AN ABSTRACT OF THE THESIS OF

Seth R. Cadell for the degree of Masters of Science in Nuclear Engineering presented on August 24, 2009.

Title: Development of a Heat Transfer Coefficient for a Single Mist Nozzle Impinging on a Heated Thin Sheet of Alloy 304 Stainless Steel using Infrared Thermography

Abstract approved:

---

Brian G. Woods Qiao Wu

A purpose built facility was designed, built and operated at the Advanced Thermal Hydraulic Research Laboratory, which is located at Oregon State University. This facility was used to heat a plate of Alloy 304 Stainless Steel to 800 °C and then cool the plate using a fine mist of water droplets. During the cooling process, the temperature of the plate was measured using an infrared camera. Using the experimental data an inverse calculation was performed to determine the heat transferred from the test plate. The experimental heat transfer data was used to develop a set of piecewise equations that describe the heat transfer from the test plate with a given flow rate over the four boiling regimes.

# Development of a Heat Transfer Coefficient for a Single Mist Nozzle Impinging on a Heated Thin Sheet of Alloy 304 Stainless Steel using Infrared Thermography

by

Seth R. Cadell

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented August 24, 2009
Commencement June 2010

Master of Science thesis of Seth R. Cadell presented on August 24, 2009.

APPROVED:

Co-Major Professor, representing Nuclear Engineering

Co-Major Professor, representing Nuclear Engineering

Head of the Department of Nuclear Engineering and Radiation Health Physics

Dean of the Graduate School

Seth R. Cadell, Author

## ACKNOWLEDGEMENTS

Countless conversations with my peers have helped while executing this work and it would be impossible to acknowledge all of them. In particular, the author would like to acknowledge a few key people who have greatly influenced this work.

Mitsubishi Heavy Industries and their engineers who have been working with the OSU staff and traveled to Oregon twice deserve commendation. This work would not have been possible if not for the insight of Dr. Keiji Mizuta and Dr. Yoshiyuki Kondo and their ability to provide the fiscal and intellectual motivation to take on this study.

Drs. Woods and Wu merit recognition for their direction (and at times intentional lack thereof) while I learned and intellectually grew during this project.

The work of Jeff Luitjens and John DeNoma was influential in the design and operation of the apparatus.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

## LIST OF FIGURES

LIST OF FIGURES (Continued)

## LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

# LIST OF TABLES

## NOMENCLATURE

| | |
|---|---|
| ATHRL | Advanced Thermal Hydraulics Research Laboratory |
| CWC | Camera Window Cover |
| DAQ | Data Acquisition |
| DAS | Data Acquisition System |
| GMT | Greenwich Mean Time |
| HTC | Heat Transfer Coefficient |
| IHCP | Inverse Heat Conduction Problem |
| JJXP | 1/4FJJXP040S303, Ikeuchi Single-Phase Spray Nozzle Part Number |
| MHI | Mitsubishi Heavy Industries |
| NERHP | Department of Nuclear Engineering and Radiation Health Physics |
| OSU | Oregon State University |
| TAHTC | Time Average Heat Transfer Coefficient |
| VI | Virtual Instrument |

In Loving Memory of

Mary Elizabeth Cadell

*Development of a Heat Transfer Coefficient for a  Single Mist Nozzle Impinging on a Heated Thin Sheet of Alloy 304 Stainless Steel using Infrared Thermography*

# *1 Introduction*

There are many applications in our daily lives where a fluid is directed at a medium with the intent to perform some function.  Some common applications are: inside a dishwasher to remove food residue from the dishware, at the local coffee shop where steam is used to heat and mix the ingredients of a latte, or at a factory to cool an object.  The industrial applications of jet sprays, or jet impingement, are numerous.  The work conducted in this study observed the rate at which a jet of fine droplets cooled a piece of steel, and then used those observations to develop a set of equations that describe the heat transferred from the plate to the water.

When a piece of steel is manufactured, molten iron is combined with various elements such as chromium, carbon, silicon, manganese, and nickel to create a specific alloy.  This molten material is then pressed and rolled into the desired shape.  The grains inside material have no general structure, and are randomly ordered.  To make the alloy stronger, the grains can be aligned through a process called annealing.  During this process the alloy is heated to temperatures that are 50 to 75% of its melting point and then cooled.  The temperature the alloy is heated to, the rate of heating, and the rate of cooling all influence the amount of change in the overall structure of the material.  For some alloys, multiple treatments exist for the same chemical composition.  Alloy 6061 Aluminum, for example, is annealed using one of two different treatments (T-4 or T-6). Both of these materials have the same chemical composition but differ in their internal structure, which provides slight variations in ductility, strength, elasticity, thermal conductivity and electrical resistance.

This study examined one of the physical components used in the annealing process, the spray nozzle. Using experimental data gathered from a purpose built apparatus; a heat transfer coefficient was developed to describe the heat transferred from a piece of steel to the surrounding environment. This coefficient can be used by metallurgists to determine the spray duration and flow rate necessary to yield the desired material properties.

While the direct purpose for generating this heat transfer coefficient is related to materials science, there are many applications in the nuclear industry where localized jet impingement is used. For example, the pressurizer in a pressurized water reactor uses direct jet impingement to regulate the system pressure. Future applications in the industry may arise where natural circulation in a passive safety system will not provide enough heat removal and jet cooling may need to be implemented.

An investigation has been conducted at the Oregon State University Nuclear Engineering Department's Advanced Thermal Hydraulic Research Laboratory (ATHRL) to develop a heat transfer coefficient for jets impinging on a thin steel plate. This thesis will provide a first attempt, using only experimental data, at creating a piecewise function to describe the heat transfer of the material as a function of flow rate, nozzle position, material properties, and position with respect to the nozzle center. It is anticipated that a piecewise function based on the fundamental boiling regime physics will more accurately describe the heat transfer characteristics than the current global correlations that exist.

To develop the heat transfer coefficient (HTC), a few critical assumptions were made. The first assumption made was that repeatedly thermally cycling the test plates would not affect their thermal characteristics. This assumption allowed multiple tests to be conducted using the same test plate, which in turn reduced the total number to test plates used in this study. During repeated testing variance in the plate's thermal properties was not observed. The second major assumption is that the test material is "thermally thin." A material whose "resistance to conduction within the solid is much less than the resistance to convection across the fluid boundary layer" is said to be thermally thin (1). A material qualifies for the thermally thin designation when the Biot number is less than 0.1 and a Biot number of 0.06 was calculated for this study. This assumption allows the measurements to be taken at the uncooled face to be treated as measurements on the cooled face. This simplification allowed a multi-dimensional problem to be reduced to a one-dimensional problem, as well as the omission of the lateral heat transfer within the plate. The third assumption is that all the error in the experimental data follows a normal distribution. This assumption is commonly accepted within experimental measurements and allows the experimenter to interpret the effects of the error.

This work provides a heat transfer coefficient for specific cases of mist jet impingements, but it contains some limitations. It is not possible to create a function of every imaginable parameter. The equations proposed in this study describe the heat transfer of a given material, at a given fluid flow rate, with a given fluid temperature, with a given nozzle to plate distance will provide a relationship that will describe the heat transfer through the four boiling regimes. Each of these givens should most likely be parameters of the HTC equations and it is planned to extend this work to incorporate more parameters into the HTC equations to provide a broader understanding of the phenomenon and developing a set of non-dimensionalized equations. However, for the specific applications at MHI, the test conditions are closely related to the prototypic environment, and the test data can be directly applied without generalization.

The following chapters will discuss and document the work conducted to develop the recommended heat transfer coefficient. Chapter 2 will discuss the relevant knowledge that exists in the literature about the techniques used and the existing heat transfer correlations. Chapter 3 will discuss the apparatus that was designed and built at OSU to collect the data necessary for this study. Chapter 4 will discuss the instrumentation installed on the apparatus and how it was calibrated. Chapter 5 will discuss the method of data acquisition used, display the experimental data gathered for this study, and discuss the sources and propagation of experimental error. Chapter 6 will contain the data analysis and resulting heat transfer coefficient. Chapter 7 will provide conclusions to the work, observations made, and an outline of future work to be done. References and appendices follow Chapter 7.

# *2 Background*

The purpose of this study is not to provide a seminal work but to build off of the efforts of others. Within this study, aspects of jet impingement, heat transfer, and boiling phenomenon will be discussed. The following sections will discuss prior work in each field that is relevant to this work. The differences between the work conducted in this study and the existing body of knowledge will be outlined.

## 2.1 Jet Impingement Overview

Any time an object needs to be heated or cooled at a rate faster than the surroundings will allow an external mechanism is needed. Similar to turning on a fan on a hot summer day, using a jet to direct a fluid at an object will increase the local heat transfer of that object. Impinging jets provide a controllable mechanism to change the temperature of a given object. Jet impingement cooling can generate much less waste water compared to submersion pools all the while producing the same cooling effects. Jet impingement has been studied for many applications, such as the steel industry in annealing alloys (2) (3) (4) (5), the food processing industry investigating the cooling of boiled eggs (6), the electronics industry to increase the cooling of integrated circuits (7) (8) (9), and the solar power industry to increasing photovoltaic electrical output by keeping the cell cooler (10).

One of the seminal works on jet impingement was written by Martin in 1977 (11). In this work Martin outlined the existing experimental data and developed basic heat transfer equations based using single and multiple arrays of jet nozzles impinging on a flat surface. He also provided equations for both slot nozzles and circular nozzles. His heat transfer equations were designed to be first order specific and used in general engineering cases. Many subsequent studies have been conducted to improve the accuracy of the equations and the understanding of the effects in changing the physical parameters. Investigations into the effects of varying nozzle to surface have been conducted (12) (13) (14). Martin's work looked at open air situations, since then studies looking at submerged jet impingement (12) (14) (15) (16) and confined jet impingement (17) (18) have been conducted. Other studies have focused on changing the plate angle (19), the plate thickness (20), and the size of the jet to the micro scale (21). The correlations provided in this body of jet impingement studies do not apply to situations where the Reynolds number is greater than 10000. The test case used in this the work the Reynolds number is 33000. While no direct correlation is available the method of handling the thermocouple data was very consistent and the general methods observed in this collection work were applied during the design phase of the apparatus. Features such as installing spring tensioned thermocouples, providing space for the lateral expansion of the plate, and the use of a "thermally thin" plate were seen in multiple impingement studies.

## 2.2 Heat Transfer

When objects of different temperatures contact each other, energy is transferred from the hotter body to the colder body until thermal equilibrium has been reached. The heat transferred between two mediums is studied in many fields and there are books available on the subject. A common text book, Heat and Mass Transfer by Incopera (1) provides the physical foundation and general relationships to produce first order approximations for determining the heat transferred between two bodies. Many authors have published work describing more specific heat transfer coefficients for specific materials (22) (23), geometries (9) (22) (24), and surface conditions (25). None of the works found describe the heat transferred from stainless steel to water droplets in each of the four boiling regimes. The limited material available on the HTC for stainless steel is listed to vary from 500 to 10,000 $W/m^2K$ (1) with the average HTC in a steam to water heat exchanger being 680 $W/m^2K$ (26). Using the boiling curve in Incropera, it was determined that the stainless steel heat exchanger heat transfer value provided the lowest bound for the heat transfer and thus the value of 680 $W/m^2K$ was used in the determination of the Biot number for the thermally thin assumption.

## 2.3 Boiling Phenomenon

Boiling is the process of water changing phase into steam. There are three distinct types for boiling: Nucleate, Transition, and Film. Each of these types are physically unique and a different amount of energy is transferred from the heater to the fluid while the fluid is at a specific point in the boiling regime. Many studies have been conducted studying the various boiling regimes, transition points and heat fluxes during each regime. A few studies that are of importance to this study specifically is the characterization of the Leidenfrost point, which is the point where transition boiling becomes film boiling. These studies characterize the minimum Leidenfrost point and the effects that surface roughness have on the point (27) (28). Other studies used as reference are a study using infrared Thermography to observe boiling (29) and a study discussing the effects of droplet size and velocity on the boiling regime (30). The information in these studies was used during the development of the HTC and will be used to justify the use of a piecewise equation, where each piece of the equation is over the temperature range of a specific boiling regime.

## 2.4 Contribution to the Body of Knowledge

Through the literature review process, a study or a collection of studies do not adequately describe the heat transfer from a thin plate to the water deposited on it from fine mist jet impingement. The focus of this study will be to develop an apparatus capable of generating the experimental data necessary to create a heat transfer coefficient (HTC) that describes the heat transferred in each of the four boiling regions for a given nozzle size, position, and flow rate. The HTC will also be developed for a specific planar material.

# *3 Test Apparatus*

The Oregon State University (OSU) Department of Nuclear Engineering and Radiation Health Physics (NERHP) conducted a thin-plate jet impingement study using both single and two-phase spray nozzles for Mitsubishi Heavy Industries (MHI). This study included numerical studies conducted in the computational fluid dynamics code Fluent® in conjunction with experimental studies conducted in the ATHRL to determine the heat transfer coefficient for a thin stainless steel plate using a given spray nozzle, at a specified nozzle spacing and spray flow rates. MHI is in the process of changing spray nozzles in their steel annealing plant. The data gained from this study and the resulting heat transfer equations will provide the metallurgists the needed information to tailor cooling of the new nozzles to provide the desired annealing of the metal.

The following sections describe the apparatus design criteria, test material, layout of the resulting apparatus, operation, and problems that were identified during shakedown testing.

## 3.1 Apparatus Design Criteria

The following specific requirements for the apparatus were outlined:

- Provide up to ±150% of the scaled jet air/water flow velocity and nominal plate temperature
- Jet activation must be user controlled
- Test plate must be uniformly heated to 800°C
- Test plate must have ten independent thermocouple measurements
- Temperature measurements must not interfere with fluid path
- Data acquisition system must record and store all pertinent test data
- Fluid flow rates will be controlled allowing constant delivery to the nozzle
- Fluid flow meters will be ±1% accurate

- Thermocouples will be ±1% accurate
- Pressure transducers will be ±3% accurate
- General instrumentation will provide measurements within ±3% accuracy over their given range
- Ensure the plate does not deform more than 2mm during thermal cycle
- Use material for the test plate with well know thermal properties and low scale build up

Additionally, four internal design requirements were imposed on the apparatus:

- Infrared camera must be protected from heat and moisture during testing
- Transition from plate heating to plate cooling must be rapid
- Apparatus must be mobile
- Apparatus must collect all water runoff

The apparatus was designed to operate in the ATHRL which provides the following utilities:

- City water supply, 10 gpm at 30 psi
- Water filtering system, mineral beds
- 220 VAC 200 Amp electrical service
- Compressed air, 1000 scfm at 100 psi
- Runoff drains installed in the floor

## 3.2 Test Material

MHI produces a proprietary carbon steel alloy in their annealing facility. To protect their interests, Alloy 304 Stainless Steel (SS304) was used for the experimental work. This material was chosen because it is readily available in the US, its thermal properties are known, and produces much less thermally induced oxidation scale compared to other carbon steels that are available. Once the heat transfer coefficients are determined using the SS304 data, the MHI engineers will modify the equations according to the material properties of their proprietary alloy. The prototypical test plate is 2mm thick, but the closest available steel thickness in the US was 14 gauge: having a nominal thickness of 1.98mm. A random sample of the procured test plate thicknesses indicates a mean thickness of 1.862mm ± 0.0048mm. This variation is within the specification range of the material. The manufacturer of the test plates states that the surface roughness is no more than 6μm.

SS 304 has a density of 7889 kg/m$^3$ between 20 and 100 °C (31). No information is available about the variation of the density with respect to temperature. It is assumed that the change in density is negligible and thus will be treated as constant.

### 3.2.1 SS304 Thermal Properties

The experiment requires the SS304 temperature to vary from 700°C to ambient temperatures. Over this wide temperature range the thermal properties of SS304 are not constant. Values for the thermal properties of SS304 over the range of the experiments can be found in Table 3.2-1. Figure 3.2-1 and Figure 3.2-2 display the experimental data, diamonds, and the regression lines that were fit to the data (32). A linear regression line was chosen because it is much more accurate than a constant term yet does not over sample the data as a higher order regression line may. Using a linear regression line over a higher order polynomial follows the concept of parsimony, where a model should have the minimum number of parameters consistent with the physical basis(33). The equations that are derived from the regression line will be used in subsequent chapters to determine the thermal properties at a given temperature.

Table 3.2-1: Table of empirically derived SS304 thermal properties (32).

| Temperature (K) | Thermal Conductivity ( W/ m K ) | Specific Heat (J/kg k) |
|---|---|---|
| 273 | 14.30 | 460 |
| 500 | 18.30 | 540 |
| 700 | 21.20 | 569 |
| 900 | 24.00 | |
| 1200 | 28.00 | 640 |
| 1670 | 33.30 | 707 |

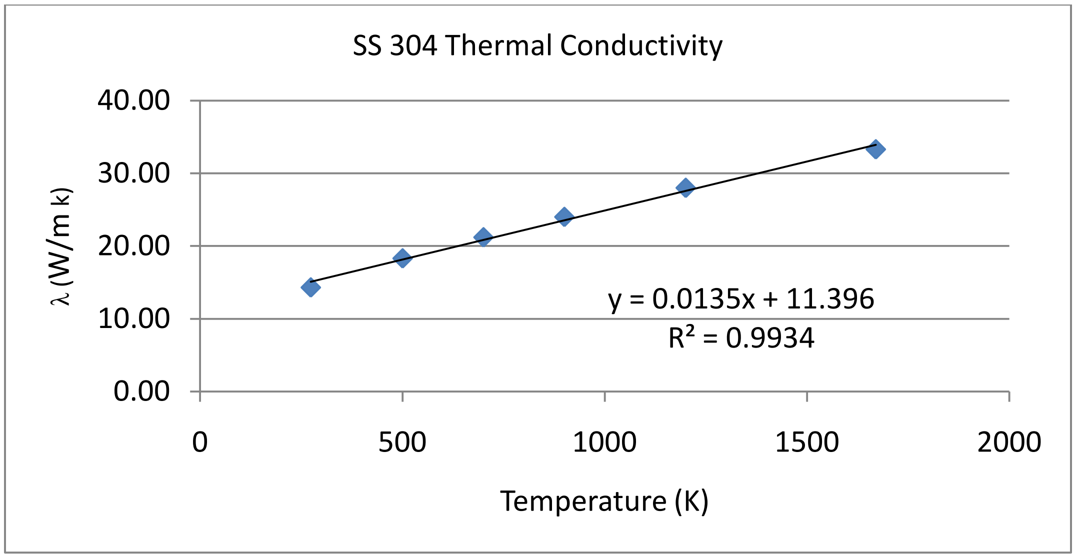


Figure 3.2-1: Plot of the SS304 thermal conductivity empirical data with a linear regression line.

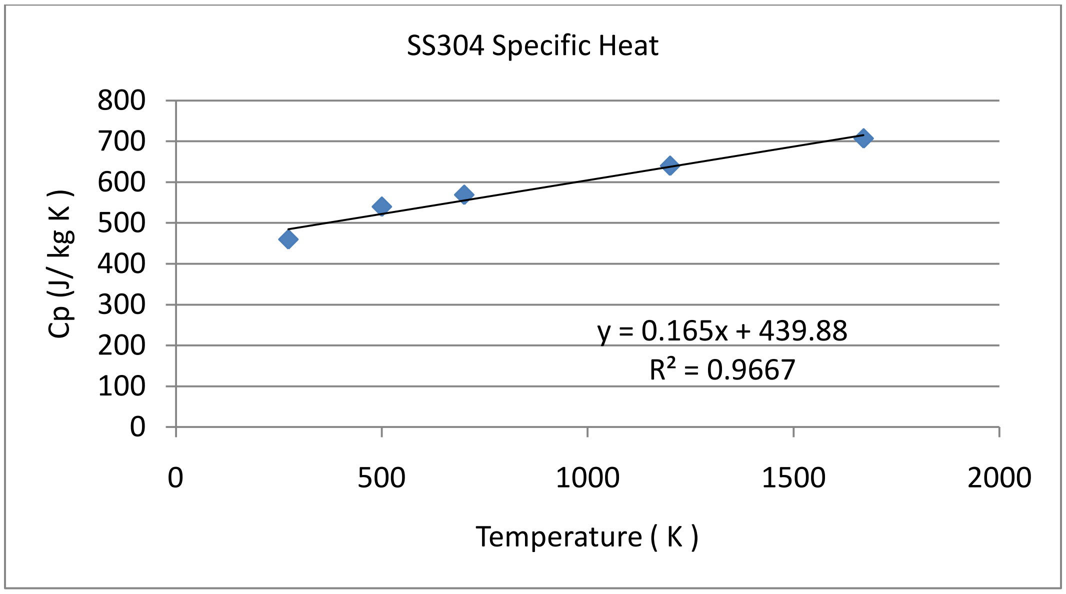


Figure 3.2-2: Plot of the SS304 specific heat empirical data with a linear regression line.

### 3.2.2 SS304 Emissivity

The emissivity of polished stainless steel is 0.57 which is not adequate for accurate infrared measurements (34). To improve plate emissivity, flat black high temperature paint was applied to the measured face of the test plate (35). During the shakedown tests, the paint was shown to provide a negligible change in thermal conductivity by comparing thermocouple measurements over complete test plate thermal cycles with and without the paint. The observed variance when comparing the data between the two test conditions was within the instrument error and substantiates the conclusion of the paint's negligible effect on plate thermal properties. The plate's emissivity with the black paint was determined to be 0.84. The method of this determination is discussed in Section 4.5, Infrared Thermography.

### 3.2.3 Heating Stresses

During the shakedown testing of the facility, it was found that the test plates were buckling and coming in contact with the non-contact heaters. The plate deflection was measured to be more than 6 cm. This amount of deflection fell outside the 2 mm maximum deflection specification provided by MHI. Various rates of heating and heater distances from the test plate were evaluated without successfully reducing the test plate deflection to within the specification. Using Solidworks® to visualize the stresses induced by heating the plate to full test temperature, it was found that the areas of high stress were present in the areas of greatest observed deflection[1]. A picture of the deformed test plate can be found in Figure 3.2-3 and a render of plate stresses can be found in Figure 3.2-4. The areas of high stress were not experimentally important and were physically removed from the test plate creating a cruciform shaped plate. Analysis of the modified plate shape showed evenly distributed plate stresses, a render of this analysis can be found in Figure 3.2-5 and a picture of a used test plate of the new design can be found in Figure 3.2-6. The modified test plate was found to deflect less than 2mm during a full temperature cycle test, meeting MHI's maximum deflection specification.

[1] This numerical work was conducted by Jeff Luitjens and presented as justification for changing the test plate shape.

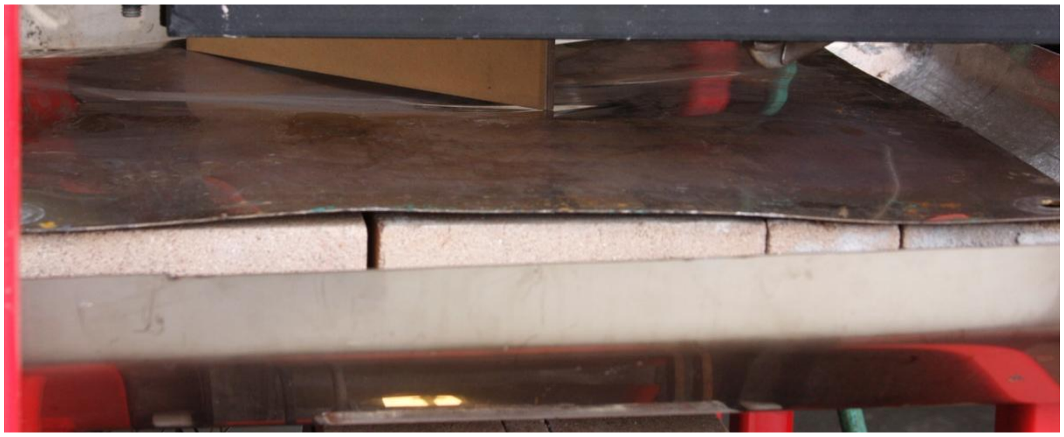


Figure 3.2-3: Picture of a heavily warped test plate.

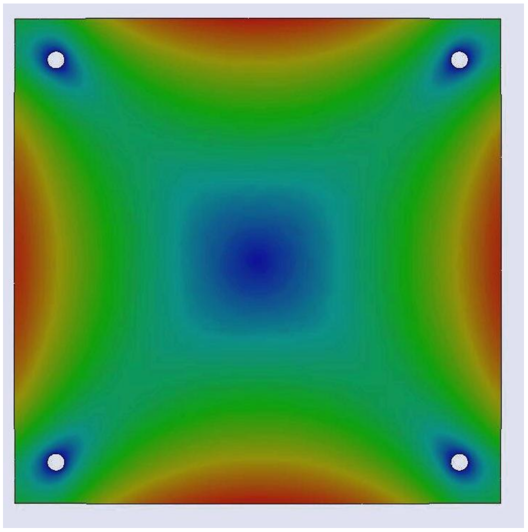


Figure 3.2-4: Render from Solidworks®, analysis of the thermally induced stresses on the square test plate. The object is shaded such that areas of low stress are blue and high stress are red.

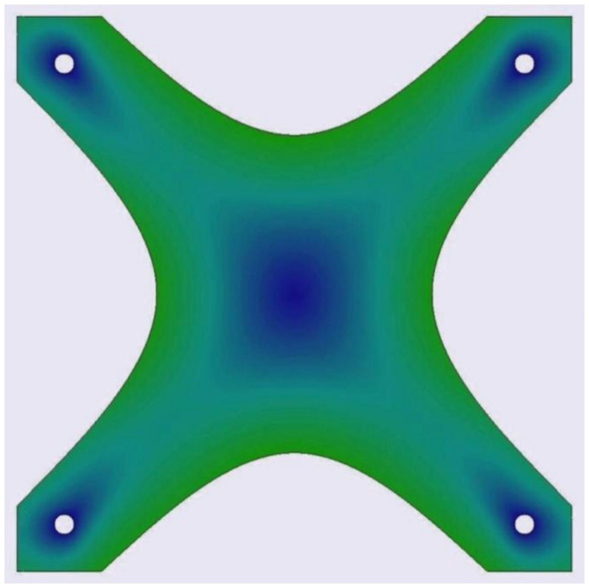


Figure 3.2-5: Render from Solidworks®, analysis of the thermally induced stresses on the cruciform shaped test plate. The object is shaded such that areas of low stress are blue and high stress are red.

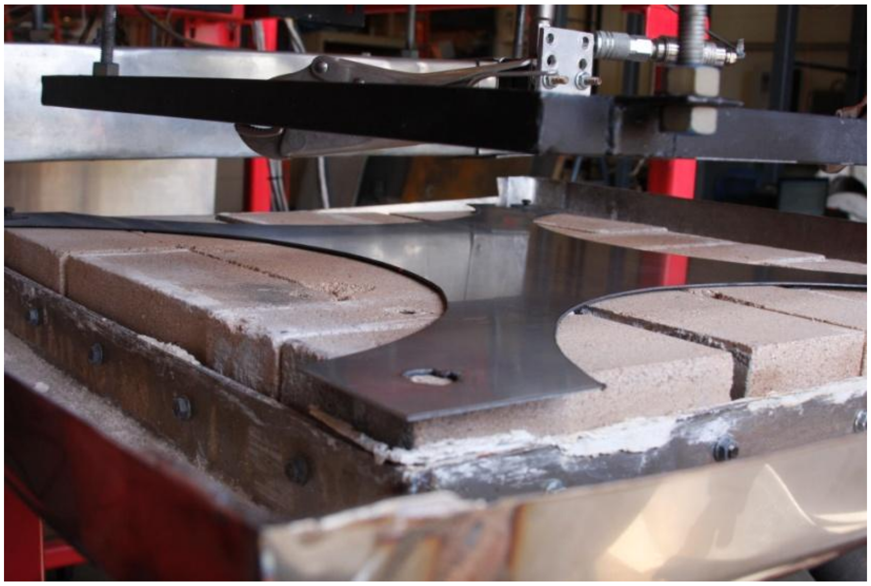


Figure 3.2-6: Picture of the modified shape after testing.

## 3.3 Ikeuchi Spray Nozzle

MHI requested that the 1/4FJJXP040S303 (JJPX) single-phase spray nozzle be one of the nozzles used in the study. This nozzle outputs a conical spray pattern with a spray angle of 65° at the standard pressure of 0.2 MPa. At that pressure, the recommended flow rate is 4.00 liters per minute of water. This flow rate is reported to provide a mean droplet size of 380 μm. This nozzle is reported to have a free passage diameter of 1.7 mm. All nozzle characteristics are taken from the manufacturer's literature (36). A picture of the typical nozzle shape, a diagram of the spray pattern, and a diagram of the spray distribution can be found in Figure 3.3-1. A picture of the test nozzle installed in the apparatus can be found in Figure 3.3-2. All experimental data included in this work will be gathered using this JJXP nozzle and hereafter will be referred to as the nozzle.

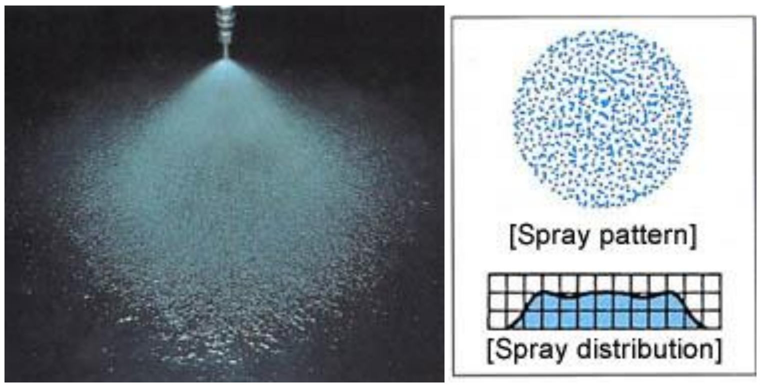


Figure 3.3-1: Picture for the spray nozzle pattern and a diagram of the nozzle shape and spray distribution, images taken from the manufacturer's website (36).

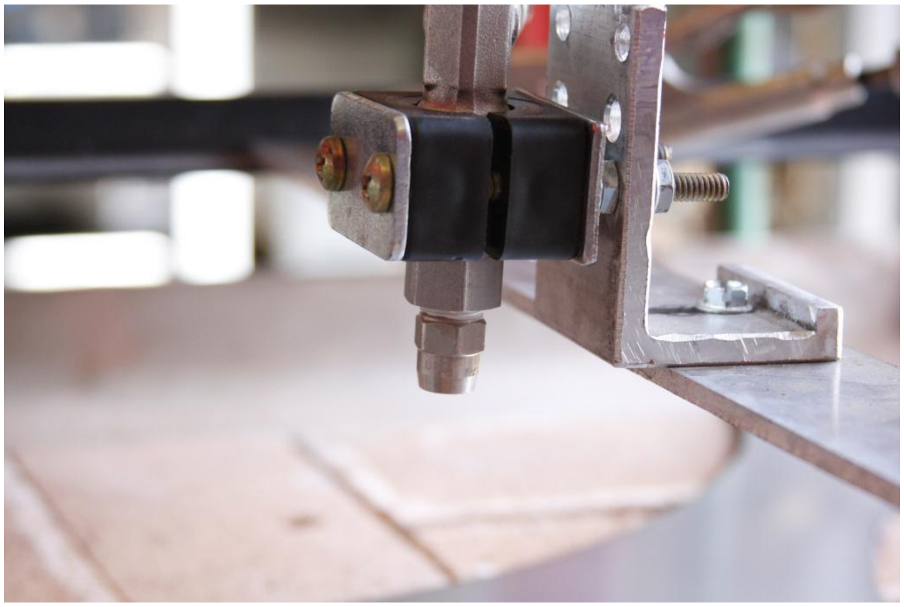


Figure 3.3-2: Picture of the nozzle installed in the apparatus.

## 3.4 Apparatus Layout

The apparatus is composed of two primary systems, heating and cooling. Both systems and the test plate are supported by the frame, which is painted red as seen in Figure 3.4-1 and Figure 3.4-2. The frame, composed of mild steel, is mounted on a set of casters allowing the facility to be mobile. The upper cross members of the frame are made using a channel shaped member which allows the lateral movement of the two trolleys that are mounted to the frame. The trolleys, one each for heating and cooling, are connected together and provide the means for quickly transitioning from the heating phase of the experiment to the cooling phase of the experiment. Pictures of the apparatus can be found in Figure 3.4-1 and Figure 3.4-2.

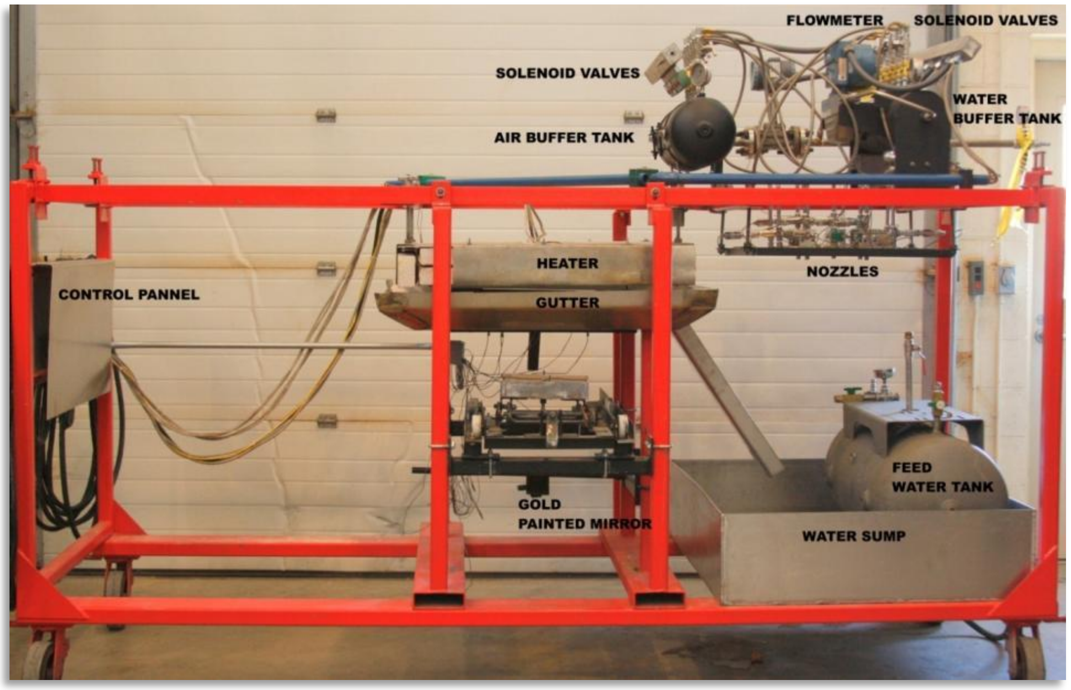


Figure 3.4-1: Side view of the test apparatus.

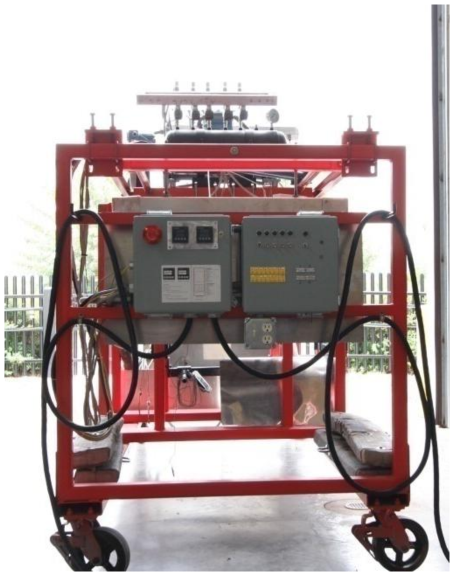


Figure 3.4-2: View of the test apparatus from the control end.

The test plate is supported by a bed of ceramic fire bricks and bolted to the frame to limit movement. The test plate attachment points have oversized holes to allow for the thermal expansion during heating. This mounting can be seen in Figure 3.2-6. The heating and cooling system are described in further detail in the following sections.

### 3.4.1 Heating System

Two radiant heaters, 14” x 30” effective heated surface area rated for continuous operation up to 1200°C, are suspended over the test plate. The heaters are mounted to the heater trolley and the mounting method provides flexibility by allowing the distance between the test plate and the heater surface to be varied from 0.5” to 4.0”. The tests discussed in this report were conducted with 1” between the heater and the test plate. The heaters operate for approximately three hours to raise the plate to 800° C. These heaters are controlled by two solid state relays and independent controllers. The heater controller measures the heater temperature at 0.25” from the surface with a K-type thermocouple. This measurement allows the heater controller to accurately maintain a constant heat up rate and a constant heater temperature. The heaters can be seen in Figure 3.4-3 and the heater controls can be seen in Figure 3.4-4. The heating system also contains two thermocouples that allow the data acquisition system (DAS) to measure and record the heater temperatures. These thermocouples are mounted next to the heater control thermocouples. It has been observed that the heater temperature measured by the heater controller and the heater temperature measured by the DAS were reliably within the measurement error of the devices.

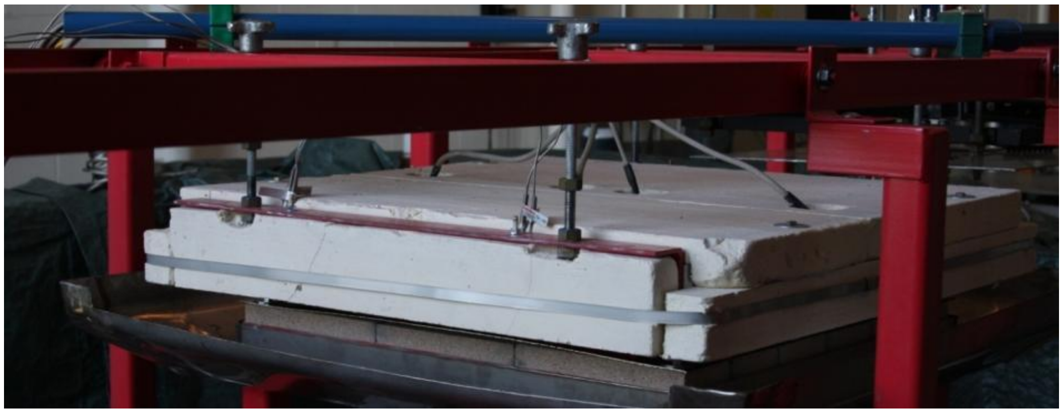


Figure 3.4-3: Picture of the heaters installed on the trolley. The heater is shown with 2" thick calcium silicate insulation.

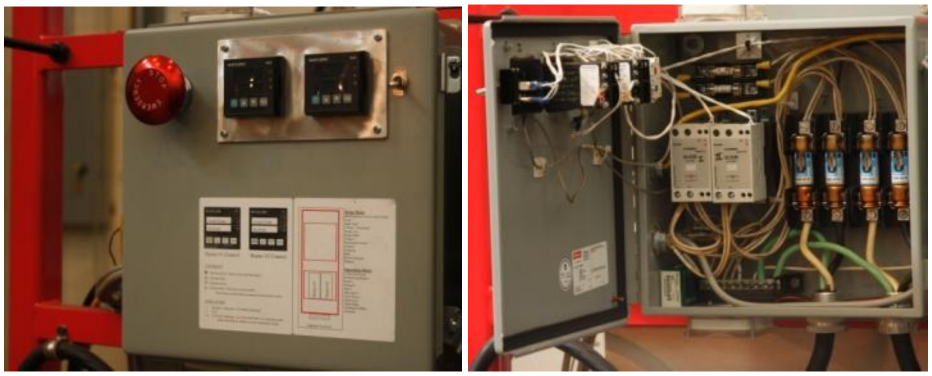


Figure 3.4-4: Picture of the heater control box. Left- exterior, Right- interior.

### 3.4.2 Cooling System

This facility was designed to provide gas and/or liquid to five test nozzles. The liquid delivery system is composed of a bulk storage tank that is held at a constant pressure, using compressed air, and delivered to the liquid distribution tank. The liquid flow is then divided between the five flow paths that lead to the test nozzles. The fluid path between the distribution tank and the test nozzle contains a solenoid valve, a balancing valve, and a static pressure transducer. The solenoid valve provides the operator the ability to actuate the impingement at a specific time. The balancing valve allows the operator to balance the fluid mass flux traveling to each nozzle, using the information provided by the pressure transducer. The gas fluid distribution system is nearly identical in function to the liquid distribution system, where the only difference is that the bulk storage tank is not held at a constant pressure, but is large enough that the volume dispensed during any given test will not change the overall system pressure by a measureable amount. This assumption is acceptable because the volume of gas used during an experiment is miniscule compared to the gas storage tank capacity, which is 1000 gallons. Both systems also have pressure regulators between the bulk storage tanks and the distribution tanks to remove any pressure fluctuations that may occur from the shop compressor cycling. A diagram for the fluid delivery system can be found in Figure 3.4-5. To provide information about each of the labels in the figure, information about each of the valves in the diagram can be found in Table 3.4-1, information about the pressure equipment can be found in Table 3.4-2, and information about the instruments in the cooling system fluid paths can be found in Table 3.4-3.

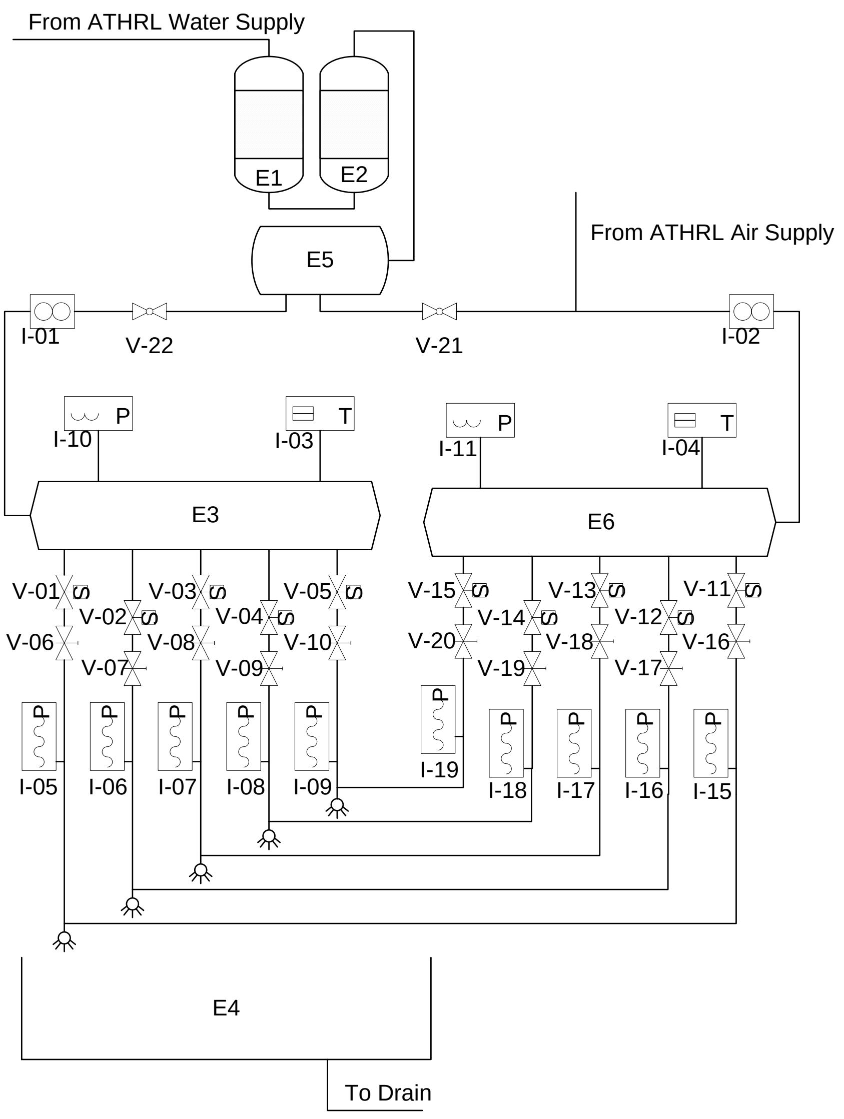


Figure 3.4-5: Schematic of cooling system: subsequent tables contain information about each of the labels.

Table 3.4-1: Table of the valves used in the cooling system where the label corresponds to the identifier in Figure 3.4-5.

| Valve List | | | | | |
|---|---|---|---|---|---|
| Label | Description | Line Size | Valve Class | Manufacturer | Model |
| V-01 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-02 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-03 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-04 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-05 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-06 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-07 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-08 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-09 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-10 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-11 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-12 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-13 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-14 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-15 | Jet Control | 3/8 NPT | Electric Solenoid | Asco | 8210G073 |
| V-16 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-17 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-18 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-19 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-20 | Flow Balance | 3/8 NPT | Manual Needle | Parker | 6F-V8LN-SS |
| V-21 | 1/4 Turn Valve | 3/4" SCH 40 Steel pipe | | | |
| V-22 | 1/4 Turn Valve | 3/4" SCH 40 | | | |

Table 3.4-2: Table of the physical components used in the cooling system where the label corresponds to the identifier in Figure 3.4-5.

| Equipment List | | | | |
|---|---|---|---|---|
| Label | Description | Manufacturer | Material | Model |
| E1 | Resin Bed | US Filter | 03263-215 | 37707 |
| E2 | Resin Bed | US Filter | 03213-113 | 37707 |
| E3 | Fluid Manifold | John DeNoma | Stainless Steel | 1 |
| E4 | Spray Catch Basin | John DeNoma | Stainless Steel | 1 |
| E5 | 30 gallon pressure tank | | Steel | |
| E6 | Gas Manifold | John Denoma | | |

Each nozzle is individually controlled by the operator using the selection switches, which can be seen on the upper half of the DAS wiring cabinet shown in Figure 3.4-6. The bulk flow rate of the liquid or gas is measured before the fluid enters the distribution tanks, painted black in Figure 3.4-7. The fluid then passes through one of five paths through the spray control solenoid, the nozzle balancing valve, past a static pressure transducer to the nozzle. The fluid temperature is measured in the distribution tanks and recorded with the flow meter and transducer data.

Table 3.4-3: Table of the instrumentation installed in the cooling system, where the label corresponds to the identifier in Figure 3.4-5.

| Instrument List | | | | | |
|---|---|---|---|---|---|
| Label | Description | Connection Size | Service | Manufacturer | Model |
| I-01 | Flow Meter | 1/2" Swagelok | S/N 703913 | Micro Motion | R025SI319U & 1FT97031C6D9U |
| I-02 | Flow Meter | 1/2" Swagelok | S/N 703967 | Micro Motion | R025SI319U & 1FT97031C6D9U |
| I-03 | K-Type Thermocouple | 1/4" Smooth | | Watlow | AFJK0FA120UK080 |
| I-04 | K-Type Thermocouple | 1/4" Smooth | | Watlow | AFJK0FA120UK080 |
| I-05 | Pressure Sensor | 1/4 NPT | S/N 67920 | Omega | PX209-060G5V |
| I-06 | Pressure Sensor | 1/4 NPT | S/N 67892 | Omega | PX209-060G5V |
| I-07 | Pressure Sensor | 1/4 NPT | S/N 67921 | Omega | PX209-060G5V |
| I-08 | Pressure Sensor | 1/4 NPT | S/N 67900 | Omega | PX209-060G5V |
| I-09 | Pressure Sensor | 1/4 NPT | S/N 67905 | Omega | PX209-060G5V |
| I-10 | Pressure Gauge | 1/4 NPT | N/A | Ashcroft | Duralife 100psig |
| I-15 | Pressure Sensor | 1/4 NPT | S/N 67905 | Omega | PX209-060G5V |
| I-16 | Pressure Sensor | 1/4 NPT | S/N 67900 | Omega | PX209-060G5V |
| I-17 | Pressure Sensor | 1/4 NPT | S/N 67921 | Omega | PX209-060G5V |
| I-18 | Pressure Sensor | 1/4 NPT | S/N 67892 | Omega | PX209-060G5V |
| I-19 | Pressure Sensor | 1/4 NPT | S/N 67920 | Omega | PX209-060G5V |

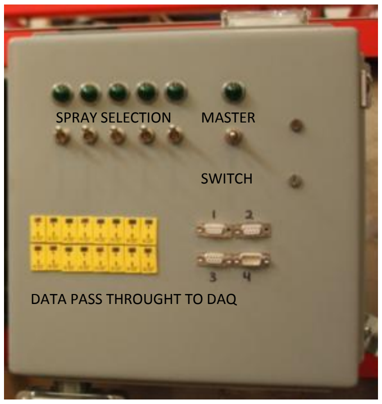


Figure 3.4-6: Figure of the fluid control and DAS wiring cabinet.

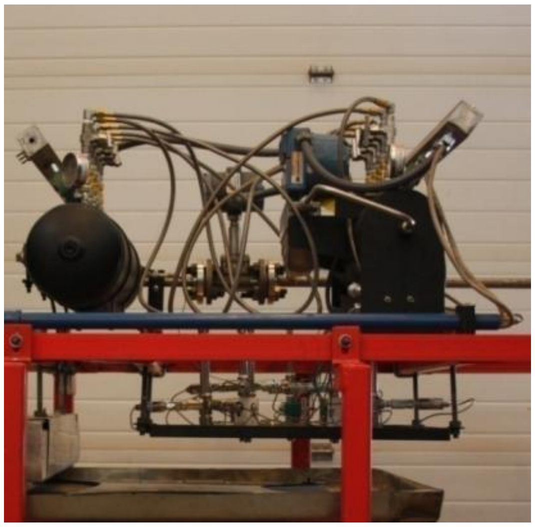


Figure 3.4-7: Picture of the gas (left) and liquid (right) distribution tanks, valves and flow meters. The tank and valves on the left distribute the gas and the tank and valves on the right distribute the liquid to the nozzles. Picture shown with two-phase nozzles installed.

## 3.5 Apparatus Operation

Once the apparatus is properly set up, the heaters are turned on and the test plate is heated to 800°C. This process takes between 3 and 4 hours depending on the temperature of the ATHRL building. Once the plate is heated, the apparatus is shifted to the cooling configuration which is discussed in the following section. When the plate cools to a given temperature, usually 700 °C, the DAS begins recording data and the jets are actuated. The plate is then cooled to 50 °C where the jets are turned off and the data acquisition is terminated. The cooling transient duration is between 12 seconds and 30 minutes depending on the fluid flow rate.

### 3.5.1 Transition from Heating to Cooling

The heaters and the cooling manifolds are mounted onto rolling trolleys that traverse laterally one meter. This lateral movement allows both the heater and cooling equipment to be suspended over the test plate. Once the plate is heated to the appropriate temperature the trolley is shifted laterally to orient the spray nozzles over the test plate. Figure 3.5-1 through Figure 3.5-5 depict this lateral movement. Physical stops are built into the frame, which are adjusted to ensure that both the heater and cooling manifold are centered over the test plate at the appropriate point in the experiment.

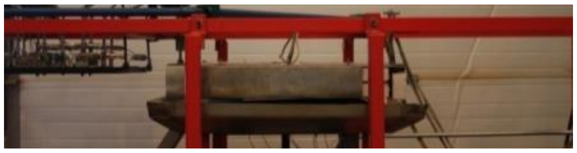


Figure 3.5-1: Trolley in the heating position, where the heater is in the center of the image and the cooling manifold is in the upper left corner of the image.

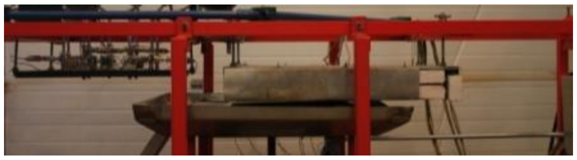


Figure 3.5-2: Trolley moving towards the cooling position.

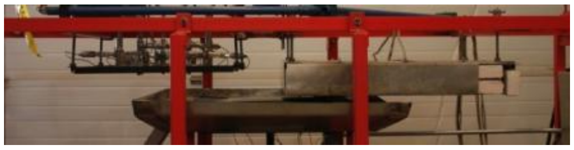


Figure 3.5-3: Trolley moving to the cooling position.

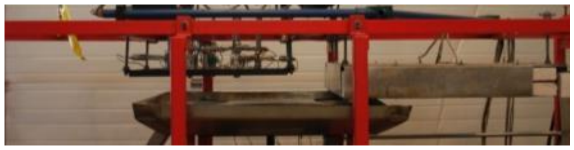


Figure 3.5-4: Trolley moving to cooling position.

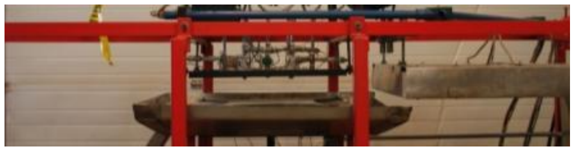


Figure 3.5-5: Trolley in cooling position, where the nozzles are over the test plate.

## 3.6 Problems Observed during Shakedown

Once the apparatus was assembled and operating, shakedown tests were conducted to ensure the proper operation of the apparatus. Throughout this testing there were three main problems that required improvements to the apparatus. The problems were excessive test plate deformation, repeated heater thermocouple failure, and non-uniform plate heating. The discussion of the plate deformation which was discussed in section 3.2.3, Heating Stresses and discussion of the other two problems will be discussed in the following sections.

### 3.6.1 Repeated Heater Thermocouple Failure

During the first two months of operating the apparatus, the thermocouples mounted on the heaters were failing. In discussions with the heater vendor it was suggested that a larger thermocouple be installed. The 3/64" diameter thermocouples were replaced with 3/16" thermocouples. The thermocouples have since operated without failure. A survey in the thermocouple literature attributed the rapid failure of the thermocouples to high temperature annealing of the sense wires (37). The wires would become very brittle and then small vibrations would break the fine wires in the 3/64" diameter thermocouples. The heavier wires would withstand the work hardening induced by the heating environments numerous thermal cycles.

### 3.6.2 Non-Uniform Plate Heating

The plate would not reach the design temperature with the camera viewing area and the heater sides open to the environment. The heater was fitted with 2" thick silicon carbonate insulation and insulation blocks were made to seal the sides of the heater during the heating. The incorporation of the insulation allowed the plate to reach the full temperature, but the measurement area of the plate was not uniformly heated. The camera window cover (CWC) was designed, built, and installed on the apparatus to provide insulation in the camera viewing window during the heating process. Figure 3.6-1 through Figure 3.6-4 depict the motion of the CWC as it lowers and is moved out of the camera viewing area. It should be noted that in this series of photographs the test plate thermocouples have been removed for clarity. It was observed that after installing the CWC, the test plate was uniformly.

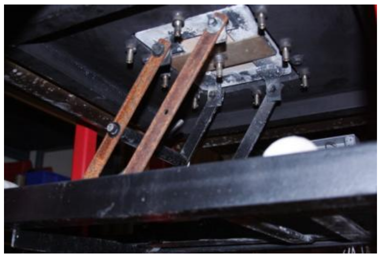


Figure 3.6-1: Camera window cover inserted.

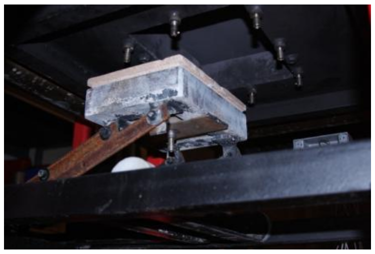


Figure 3.6-2: Camera window cover being lowered.

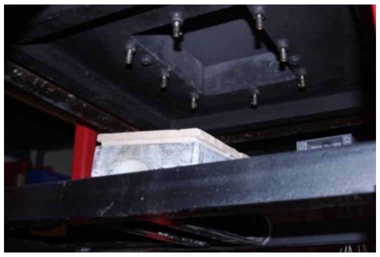


Figure 3.6-3: Camera window cover completely lowered.

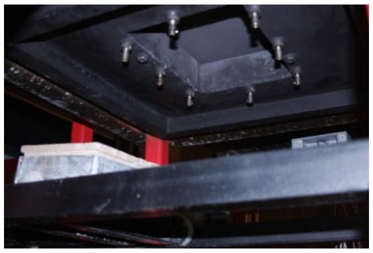


Figure 3.6-4: Camera window cover rolled out of the infrared camera's view.

# 4 Apparatus Instrumentation

To gather experimental data, a physical device such as the apparatus described in the previous chapter must have instruments installed at specific places to observe the physical state of an entity. The physical state could be a temperature, a pressure, or a position. This chapter will discuss the instruments used to gather the experimental data for this study, the method of gathering the data, and the method of calibrating each instrument. All of the instruments in this apparatus are operated within their designed range, properly calibrated, and used in a repeatable manner providing quality experimental data. The data acquisition system (DAS) is comprised of an infrared camera, ten pressure transducers, two volume flow meters, sixteen thermocouples, a data acquisition box (DAQ), and a laptop computer. The following sections discuss the pertinent details of each component.

## 4.1 Data Acquisition Equipment Relationships

Both the infrared camera and the DAQ interface with the computer. Both devices incorporate a Greenwich Mean Time (GMT) timestamp into the recorded data. This timestamp allows the two data records to be synchronized during the data extraction process. The DAQ interface measures the signal output by each of the pressure transducers, flow meters, and thermocouples. All experimental data is then collected on the laptop and stored. A flow chart of the equipment relationships can be found in Figure 4.1-1.

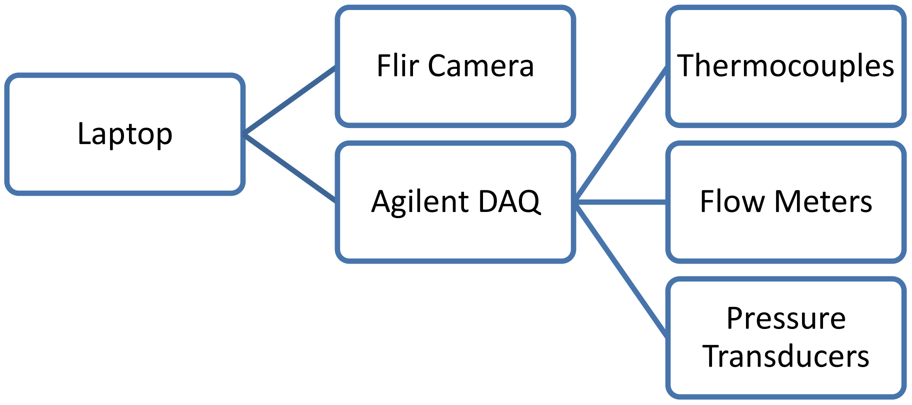


Figure 4.1-1: Diagram of the equipment hierarchy.

## 4.2 Thermocouples

In the apparatus, sixteen K-type thermocouples are installed. Thermocouples were chosen over RTDs or thermistors because thermocouples are more robust, operate over the needed thermal range, and are readily available in many form factors. Out of the thermocouple family, the K-type was chosen because it has an operating range of 0 to 1200°C, was more available in the needed form factor, and the least expensive for the needed operating characteristics. K-type thermocouples are less responsive to small changes in temperature, susceptible to high temperature drift, and influenced by electromagnet interference. These problems can be overcome by using ungrounded sheaths, annealing the thermocouple before it is calibrated, and using small gauge wire. Table 4.2-1 contains more specific information about the quantity, location, mounting method, manufacturer, part number and standard accuracy for each of the installed thermocouples.

Table 4.2-1: Table containing data about the sixteen thermocouples installed on the apparatus.

| Qty | Location | Mounting Method | Mfg | Part# | Accuracy |
|---|---|---|---|---|---|
| 9 | Plate Support | Spring Bayonet | Watlow | 10DKSRB120A | ±1.1% or 1°C |
| 1 | Camera Window Cover | Spring Bayonet | Watlow | 10DKSRB120A | ±1.1% or 1°C |
| 1 | Apparatus Frame | Clamp | Watlow | AFJK0FA120UK080 | ±2.2% or 2°C |
| 1 | Camera | Clamp | Watlow | AFJK0FA120UK080 | ±2.2% or 2°C |
| 2 | Heater | Swagelok® Fitting | Watlow | AFJK0FA120UK080 | ±2.2% or 2°C |
| 1 | Liquid Dispensing Tank | Swagelok® Fitting | Watlow | AFJK0FA120UK080 | ±2.2% or 2°C |
| 1 | Gas Distribution Tank | Swagelok® Fitting | Watlow | AFJK0FA120UK080 | ±2.2% or 2°C |

The nine thermocouples installed in the plate support and the one thermocouple installed in the CWC are mounted using an adjustable spring sheath, which keeps the thermocouple in contact with the test plate as it expands and contracts during thermal cycling. The spring sheath has a bayonet that can be used to adjust the pressure the spring applies on the thermocouple. Each bayonet attaches to a bayonet adapter that is threaded into the support plate, this mounting method can be seen in Figure 4.2-3. A diagram of the thermocouple with the spring and bayonet can be seen in Figure 4.2-2. The thermocouples used to measure the ambient and camera temperatures are clamped to members of the apparatus frame using a nylon zip tie. The remaining four thermocouples have smooth sheaths and are inserted into Swagelok® brass compression fitting. The fitting is tightened onto the thermocouple sheath which creates a leak proof connection, which is necessary in the distribution tanks, and a diagram of this fitting can be found in Figure 4.2-1.

To calibrate a thermocouple, its measurement should be compared to a known temperature. The thermocouples used in this experiment were used to measure the temperature of an ice bath, 0°C, and boiling water at near sea level, 100°C. Using the data from these two tests it was seen that all 16 thermocouples operated within their reported standard error and thus the manufacturers reported error will be used in all subsequent calculations.

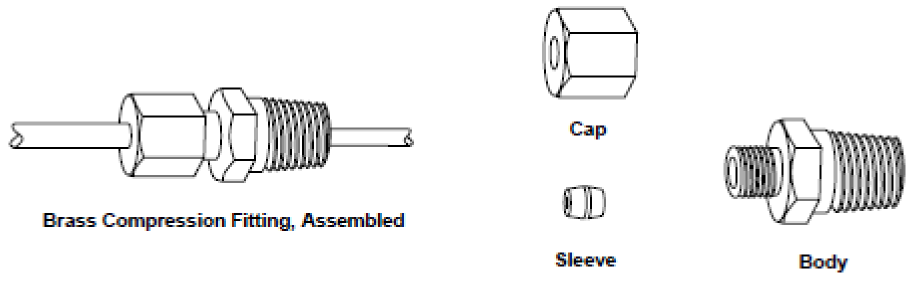


Figure 4.2-1: Diagram of the brass compression fitting mounting method allowing the thermocouple penetrate the pressure boundary of the distribution tanks without leaking.

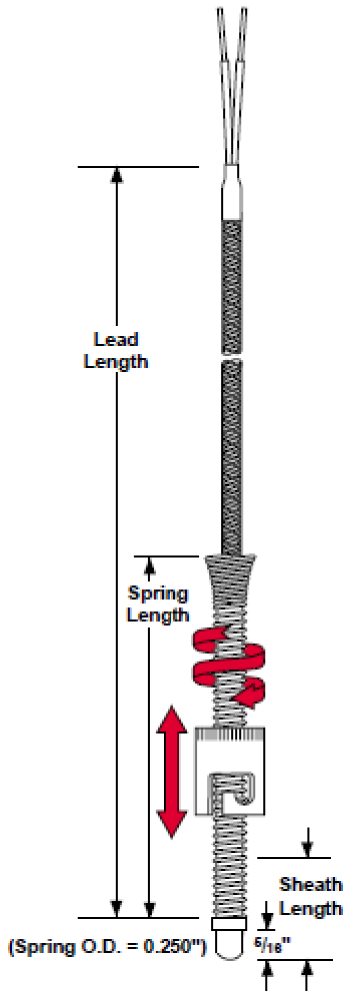


Figure 4.2-2: Drawing of the adjustable thermocouple sheath (38).

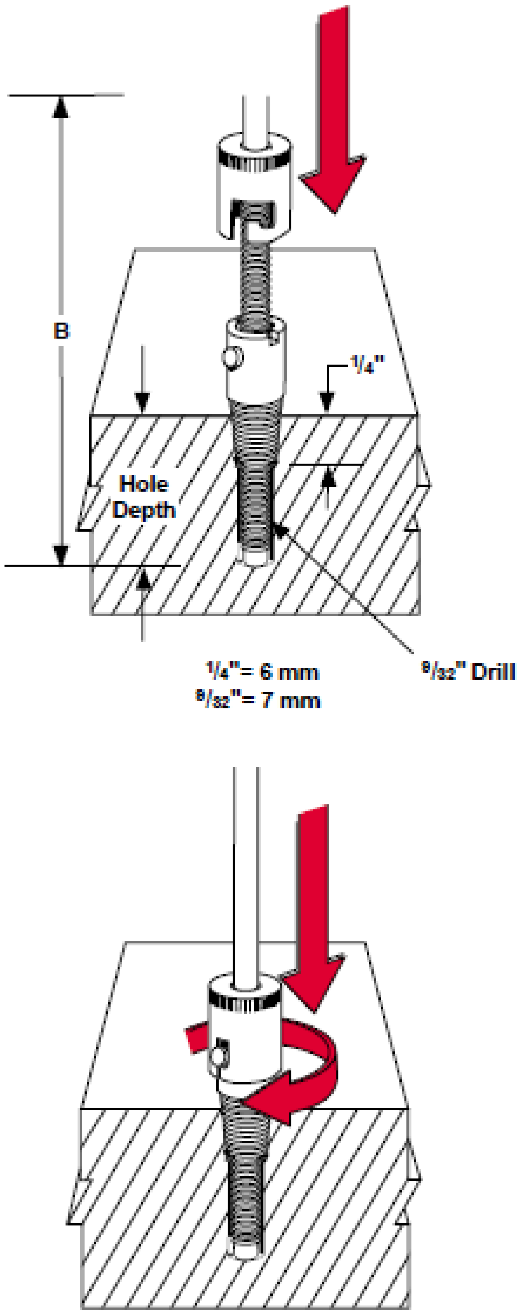


Figure 4.2-3: Diagram of the bayonet connecting to the bayonet adapter (38).

## 4.3 Pressure Transducers

A static fluid pressure measurement is made just upstream from the spray nozzles. This pressure measurement is used to ensure that nozzles have uniform fluid pressures. This measurement is made by an Omega PX209-60G5V pressure transducer and can be seen in Figure 4.3-1. This transducer provides a linear voltage output between the ranges of 0.0 and 5.0V DC which correspond to a pressure range of 0.0 to 60.0 psig. The flow meter has an accuracy of ±1.5% of the full scale output, which corresponds to ±0.9 psig. The location in the fluid path can be seen in the cooling system diagram, Figure 3.4-5. These pressure transducers were chosen because they are small, simple to calibrate, low calibration drift over time and use, inexpensive, and are sensitive to small changes in pressure. Their response time is limited due to the internal snubber which reduces the orifice that the fluid can pass through. This snubber reduces the pressure spike that occurs during the initial activation of the jets. These devices are provided with an individual NIST traceable calibration.

Figure 4.3-1: Picture of the pressure transducers used in the apparatus (39).

## 4.4 Flow Meters

The bulk flow rates of both the liquid and gaseous coolant are measured using a MicroMotion® R025 volume flow meter distributed by Emerson Process Management. This device provides a 4 to 20mA current output which correlates to a linear output between the minimum and the maximum calibrated values. The Series 1000 transmitter, capable of using the HART® communication protocol, was installed onto the R025 body. The body and transmitter combination provided the experimenter a programmable interface to vary the output window of the device. During the use of the JJXP nozzle the liquid flow meter is calibrated from 0.0 to 25.0 liters/minute. A picture of the flow meter can be seen in Figure 4.4-1. These devices were chosen because they are highly accurate (±0.025 liters/minute), currently owned by the department, and within their calibration window. The device was calibrated on 12/12/2008 and its calibration is good for two years from that date. Both the liquid and the gas flow meters were placed on the inlet of the distribution tanks. As recommended by the manufacturer, 20 diameters of straight pipe were connected to the inlet and 5 diameters were connected to the outlet of the device. These lengths of straight pipe would reduce the effects in the flow from the fittings and bends that occur in most piping systems. These meters are invasive and only provide a bulk flow rate, not a flow rate per nozzle. These two limitations do affect the overall quality of the data.

Figure 4.4-1: Picture of the R025 flow meter with the Series 1000 transmitter attached (40).

## 4.5 Infrared Thermography

Infrared Thermography is a non-invasive measurement technique that measures the energy deposited on a receiver from absorbed photos to determine the temperature of the medium that is emitting the photons. The Flir A325 infrared camera was selected for use in this application, as shown in Figure 4.5-1. The camera gathers temperature data in a 320 x 240 array of pixels. The distance between the camera and the test plate, about 0.7m, corresponds to each camera pixel measuring the area of one square millimeter on the test plate. The camera is capable of gathering images at 60 Hz and during experiments a frame rate of 10 to 30Hz was chosen depending on the transient duration. This camera was chosen because it would record data at over 30Hz for less than $30,000. Most of the cameras that would measure data at a rate of 30 frames per second or higher were over the $100,000 range.

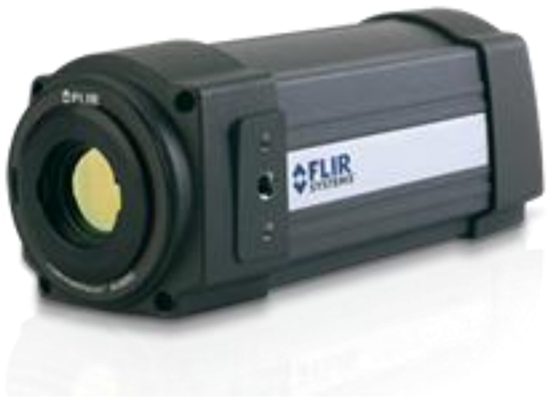


Figure 4.5-1: Picture of the Flir A325 Infrared camera (35).

To protect the infrared camera from the heat and water falling off of the test plate, a front silvered mirror is used to reflect the plate's emitted photons at the camera. The front silvering on the mirror will not absorb the photons as a back silvered mirror would. A NT-48-457 75mm x 75mm gold plated mirror sold by Edmund Optics was purchased and installed on the apparatus. This specific mirror was recommended by the camera manufacturer. An image of front silvered mirrors can be seen in Figure 4.5-2. To compensate for the error induced by reflecting the test plate's emitted photons, the plate emissivity, reflected temperature, and ambient temperature are measured with the mirror in place. With the mirror's induced error built into the background measurements the actual data recorded will not be affected by the use of the mirror, as recommended by Flir. Using thermocouple measurements to check the accuracy of the camera in the laboratory it has been seen that the use of the mirror does not induce any additional experimental error and thus the mirror's induced error is neglected.

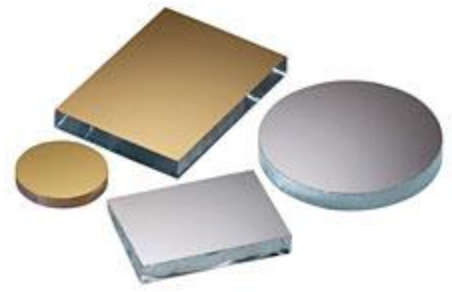
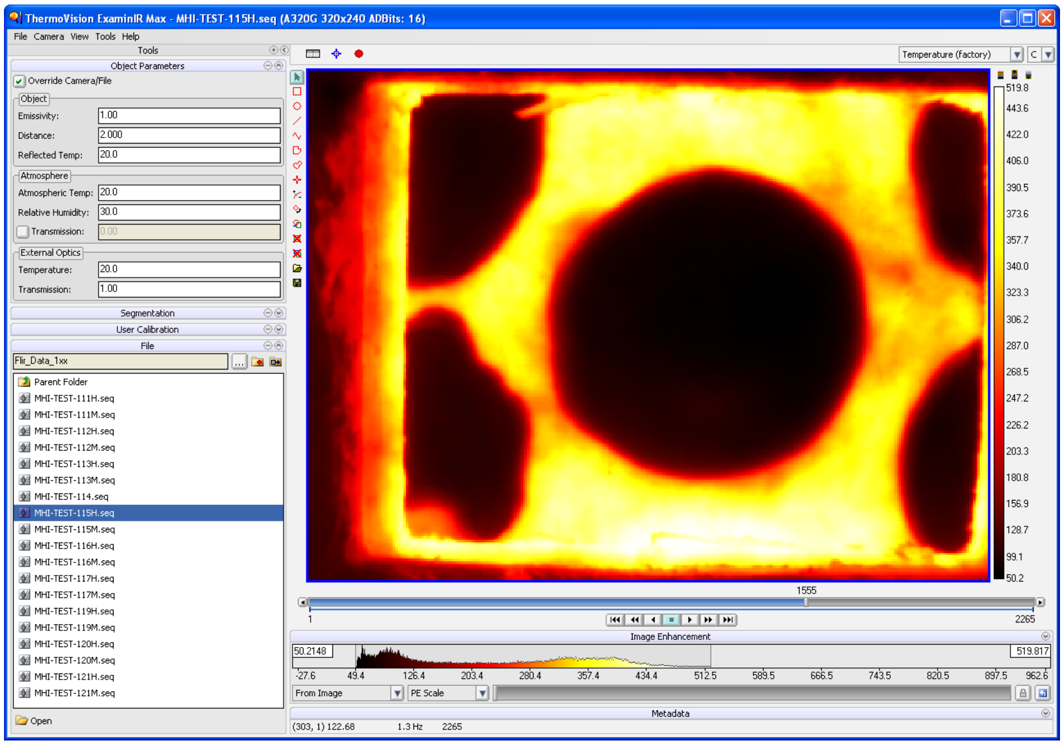


Figure 4.5-2: Picture of front silvered mirrors (41).

The photon energy is based on a particle's kinetic energy, and since temperature is a measure of the average kinetic energy of a substance, the photons borne from a heated surface will not have exactly the same energy. The energy spectrum of emitted photons follows a Poison distribution as demonstrated by Kelivn (35). It is this distribution that limits the camera accuracy to ±2% of the true temperature. The camera requires a calibration from the manufacturer as well as a daily calibration to ensure accurate measurements. The camera was calibrated 07/31/2008, which lasts for two years.

The daily calibration requires entering the ambient temperature, humidity, reflected temperature, and material emissivity, which can be seen in Figure 4.5-3. The ambient temperature and humidity can be measured with general instruments in the laboratory. The manufacturer recommends measuring the reflected temperature by placing crumpled tin foil in the viewing area and using the infrared camera to measure the average temperature. To measure the test plate's emissivity the infrared measurement must be compared to a thermocouple that is exposed to the same heat source. Figure 4.5-4 displays a comparison between the thermocouple measurement and the infrared measurement with four different emissivity settings. Figure 4.5-5 displays a comparison between the thermocouple measurement and the infrared measurement with the emissivity set at 0.84 with the included instrumental error. Once all of the instrument and environmental settings are input into the camera software reliable and accurate infrared measurements can be taken.


ThermoVision ExaminIR Max - MHI-TEST-115H.seq (A320G 320x240 ADBits: 16)

Override Camera/File [✓]

Object
Emissivity: 1.00
Distance: 2.000
Reflected Temp: 20.0

Atmosphere
Atmospheric Temp: 20.0
Relative Humidity: 30.0
Transmission: 0.00

External Optics
Temperature: 20.0
Transmission: 1.00

Flir_Data_1xx

(303, 1) 122.68 1.3 Hz 2265

Figure 4.5-3: Screen shot of the ExaminIR software with the calibration window open in the upper left corner of the window.

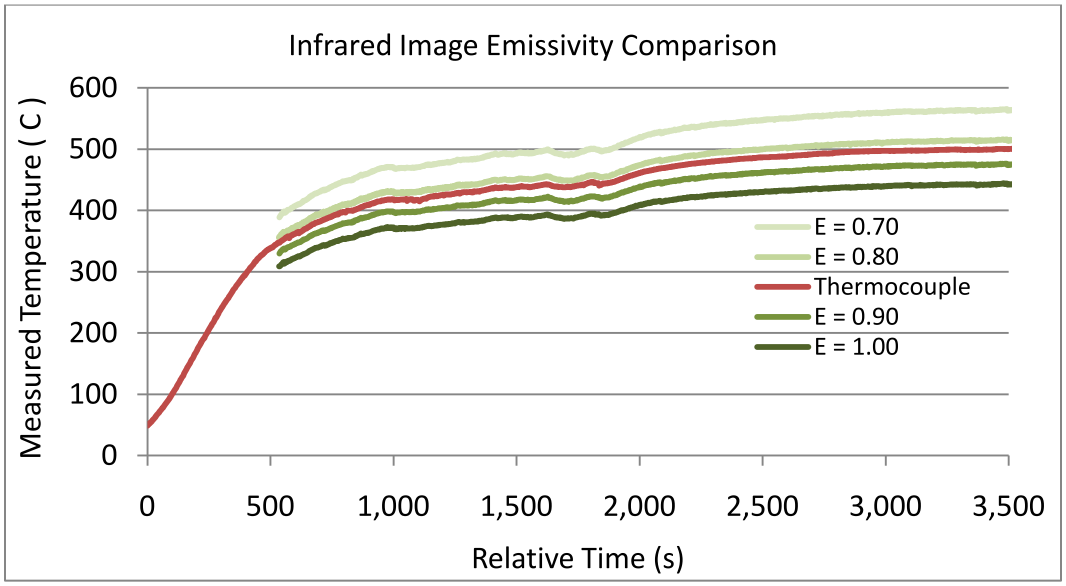


Figure 4.5-4: Comparison between a thermocouple and infrared measurement using varying emissivity values in the infrared software.

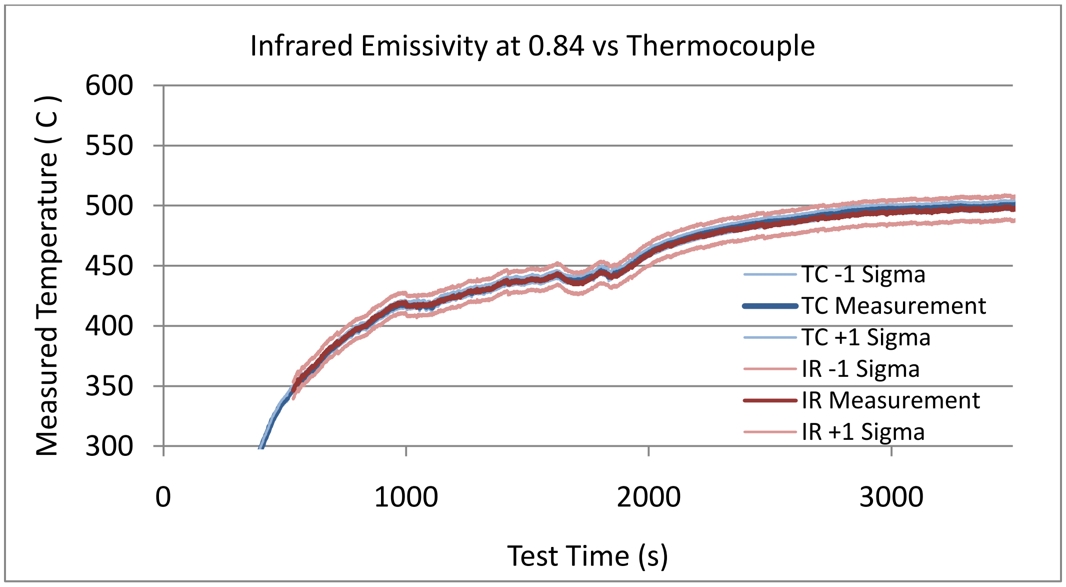


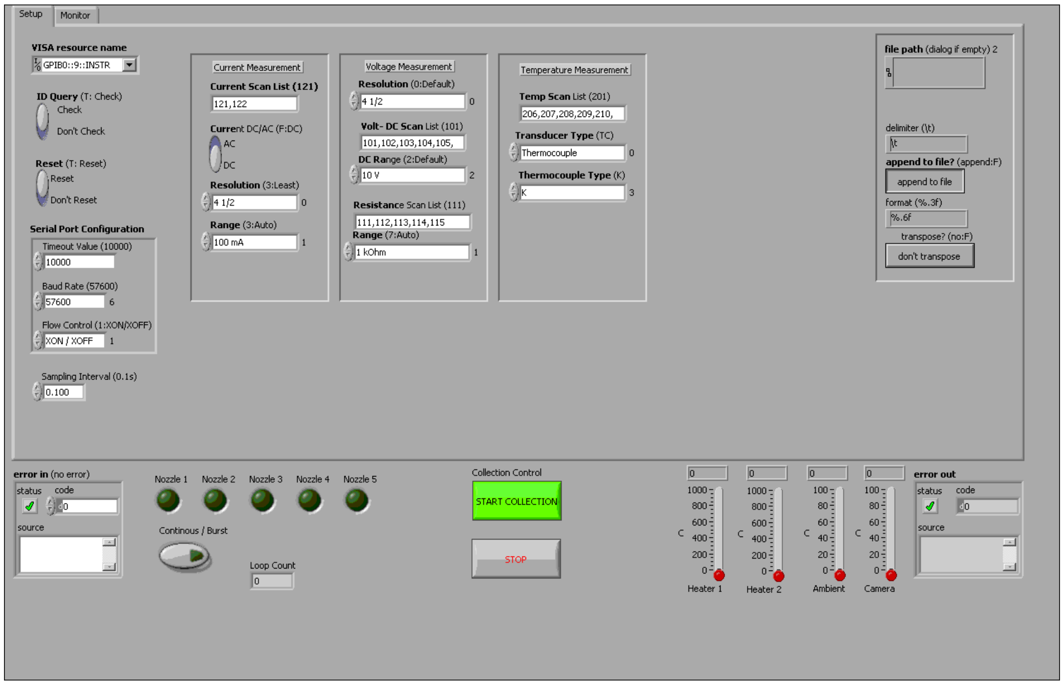
Figure 4.5-5: Plot of the resultant thermocouple and infrared measurements using an emissivity of 0.84 in the infrared software.

## 4.6 Agilent 34970 Data Acquisition Unit

The output of the transducers, flow meters, and thermocouples are measured by the DAQ box. The device used is the Agilent 34970A Switch Unit, and can be seen in Figure 4.6-1. This device is capable of sampling resistance, current, ac voltage, dc voltage and most common thermocouple types. All sensor outputs are natively measured by this system to six digits of precision with an accuracy of 0.1% of the reading. The DAQ Box is robust, proven, and capable of easily interfacing with LabVIEW™, but its sampling rate is slow. To sample all 28 instruments the DAQ box takes 35 seconds. Since the information taken by the DAQ Box will be used to determine the tests state and the overall boundary conditions a sampling rate this slow is acceptable.

Figure 4.6-1: Picture of the Agilent data boxed used in the data collection (42).

The DAQ is controlled by an application written in LabVIEW™. This application is named MHI_Interface VI, where VI is a LabVIEW™ acronym for a virtual instrument. This VI contains two windows: Setup and Monitor and can be seen in Figure 4.6-2 and Figure 4.6-3 respectively. The Setup window provides the user controls to configure the DAQ, selects the instruments to poll, select the polling order, and name the output file that will store the raw data in a comma separated value (csv) file. The output data from the LabVIEW application includes a GMT time stamp, which provides the tools to synchronize the DAQ output with the infrared camera output. The Monitor window provides a graphical plot of the ten plate thermocouples, thermometer graphics representing the heater, ambient and fluid temperatures. The window also contains graphics of six gauges that represent the flow rate and pressure in the cooling system. This interface was designed to provide an intuitive interface that was easy to determine the apparatus' state.



VISA resource name: GPIB0::9::INSTR
ID Query (T: Check): Check / Don't Check
Reset (T: Reset): Reset / Don't Reset
Serial Port Configuration
Timeout Value (10000): 10000
Baud Rate (57600): 57600 6
Flow Control (1:XON/XOFF): XON / XOFF 1
Sampling Interval (0.1s): 0.100

Current Scan List (121): 121,122
Current DC/AC (F:DC): AC / DC
Resolution (3:Least): 4 1/2 0
Range (3:Auto): 100 mA 1

Resolution (0:Default): 4 1/2 0
Volt- DC Scan List (101): 101,102,103,104,105,
DC Range (2:Default): 10 V 2
Resistance Scan List (111): 111,112,113,114,115
Range (7:Auto): 1 kOhm 1

Temp Scan List (201): 206,207,208,209,210,
Transducer Type (TC): Thermocouple 0
Thermocouple Type (K): K 3

file path (dialog if empty) 2
delimiter (\t): \t
append to file? (append:F): append to file
format (%.3f): %.6f
transpose? (no:F): don't transpose

error in (no error): status ✓ code 0 source
Loop Count: 0
Heater 1: 0 (0–1000); Heater 2: 0 (0–1000); Ambient: 0 (0–100); Camera: 0 (0–100)
error out: status ✓ code 0 source

Figure 4.6-2: Image of the Setup window in the MHI_Interface VI.

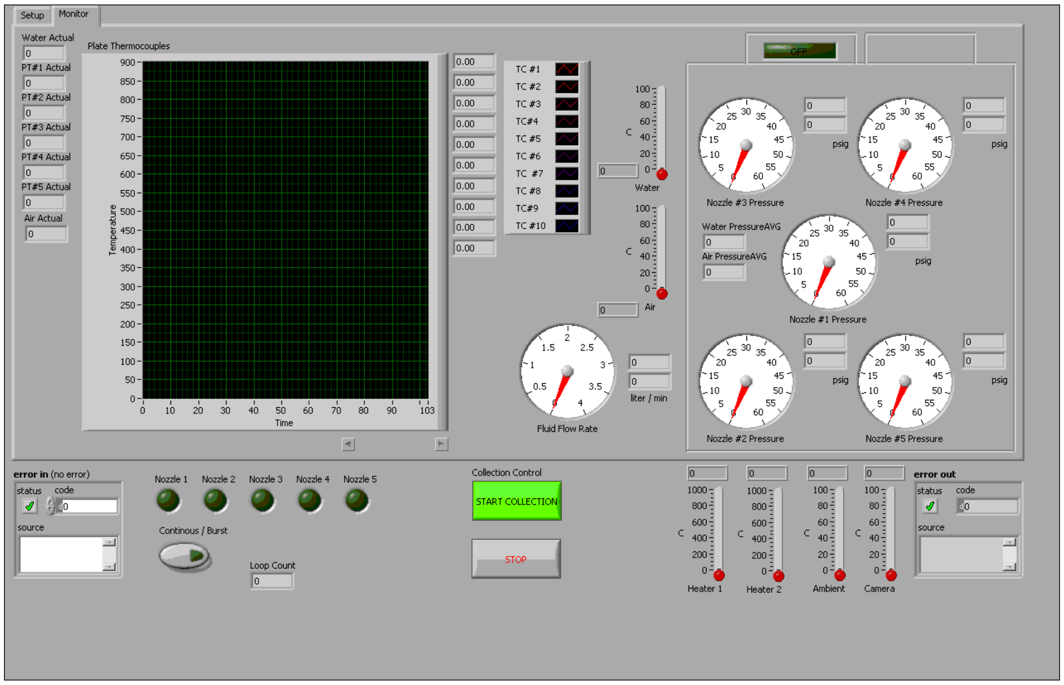


Figure 4.6-3: Image of the Monitor window in the MHI_Interface VI.

To calibrate the DAQ box and the VI the physical instruments were replaced with calibration instruments. The pressure transducers were replaced with power supply that output a carefully regulated voltage. This voltage was then read by the DAQ box and passed to the VI. The value output at the instrument could then be compared to the value reported by the VI. The flow meters were replaced with a controlled current source and the output value was compared to the reported value. To calibrate the temperature measurements, the thermocouples were replaced with a Fluke 714 Temperature Calibrator which will replicate a thermocouple. With all of the tests, the output values from the calibration tools were the reported values of the VI, within the tolerances of the equipment.

## 4.7 Thermal View Visualization Software

The ExaminIR software, purchased from Flir, does not provide any built in 3D visualization tools. Also, the current release version does not include the ability to export the temperature of multiple nonadjacent pixels over the transient. To process the data both of these tools are needed. The Thermal View program was written by the author, using the C++ language and the OpenGL graphics Application Programming Interface. A screen shot of the Thermal View interface can be seen in Figure 4.7-1 and a screen shot of the Thermal View control window can be seen in Figure 4.7-2 and the source code can be found in the Appendix.

This program allows the user to load a set of infrared camera data, in the form of csv files, and display images of the measured temperature field in multiple views simultaneously. The isometric view provides a three dimensional rendering of the temperature field where the horizontal plane represents pixel position and the vertical axis represents temperature. The pixels of each temperature measurement are colored using the heated object scale as well as displaced vertically to enhance the viewer's ability to visualize the temperature of a given frame. Thermal View will allow the user of cycle through many frames of temperature data, at a user controlled frame rate. This animation feature allows the user to view the three dimensional thermal transient in real-time. Adjacent to the isometric view, front, side and top views are presented to provide the user with views where one dimension is compressed. These compressed views provide the ability to see global trends in the thermal transient that may not be obvious in the isometric view.

Thermal View is also built to export a series of data points along a user selected line. When the “Show Export Points” check box is selected, green points are drawn over the data which allows the user to visually align the export points with the areas of interest observed within the data plots. The “Export” function also controls the generation of a 512 bit x 512 bit x 512 bit texture file that is generated. This texture file is then read into the program glMan, written by Dr. Bailey, which is used to generate the cutting plane visualization of the data as presented in 5.2.2 Test Data. These export functions allow the user to move beyond the visual inspection of the data to a numerical analysis of the data, as discussed in Chapter 6.

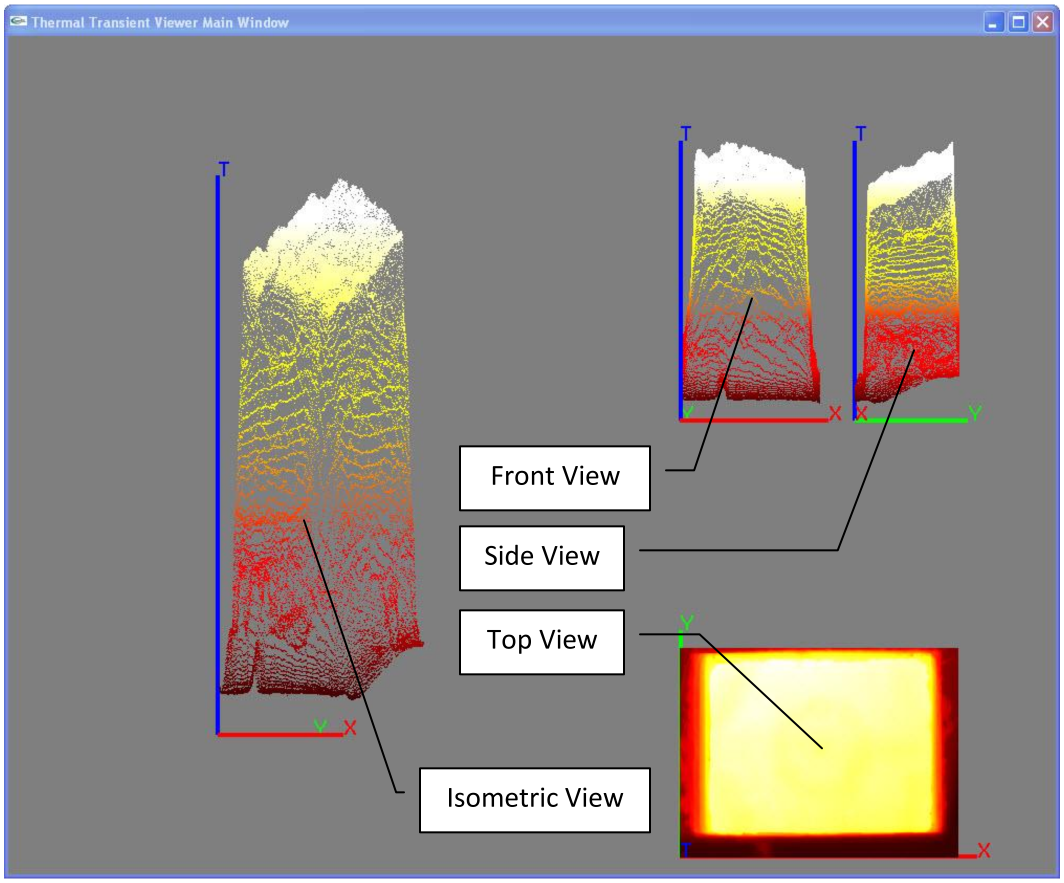
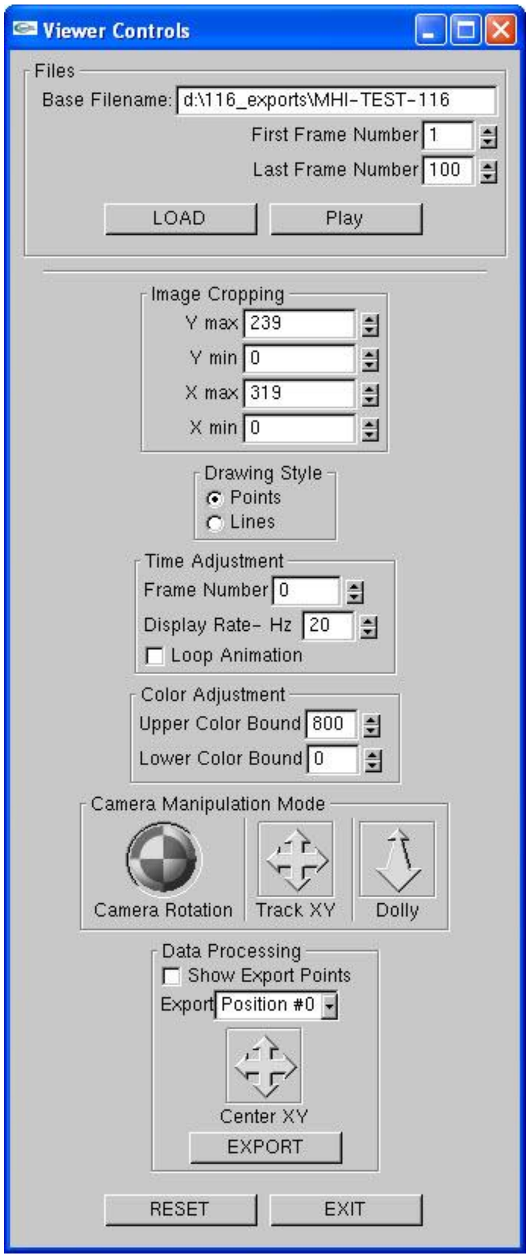


Figure 4.7-1: Screenshot of the Thermal View interface window, providing the user four views of the data field: Front, Side, Top, and Isometric.


Viewer Controls

Files

Base Filename: d:\116_exports\MHI-TEST-116

First Frame Number 1

Last Frame Number 100

LOAD | Play

Image Cropping

Y max 239

Y min 0

X max 319

X min 0

Drawing Style

[✓ Points] Lines

Time Adjustment

Frame Number 0

Display Rate- Hz 20

☐ Loop Animation

Color Adjustment

Upper Color Bound 800

Lower Color Bound 0

Camera Manipulation Mode

Camera Rotation | Track XY | Dolly

Data Processing

☐ Show Export Points

Export Position #0

Center XY

EXPORT

RESET | EXIT

Figure 4.7-2: Screenshot of the Thermal View user control window.

# *5 Experimental Data and Error*

With the apparatus constructed and operating as required, a test method and procedure were developed. The tests were conducted using this procedure and the data was processed provide meaningful information. During the data processing, the error incorporated into the measurements must be characterized before conclusions can be drawn from the data. This chapter will first outline the testing methodology used to gather the experimental data. Then a set of experimental data will be presented with the initial and boundary conditions. The instrumental error present in the data will be discussed and characterized. This chapter will close with a discussion on the propagation of the raw experimental error into the final product of the data, the desired HTC.

## 5.1 Method of Acquiring Experimental Data

To produce useful experimental results the experimenter must conduct the experiment in a manner that will allow the assessment of the *degree of goodness* of the data (43). The *degree of goodness* is a measure of how accurately the experimental data matches the true measurement. By combining the data uncertainty with the experiment's repeatability, the ability to conduct the same test again with the same results, a person drawing information from the data will have an understanding of how much information can be extracted for a given data set. For example, a set of data displaying the relationship of temperature versus voltage of a thermocouple is fit with two regression lines: linear and cubic. Both lines are well within the uncertainty of the data set, no argument based on the data will allow the selection of the "better" regression line and thus using the guidelines of parsimony the linear regression would be chosen. If the linear regression was not contained within the error of the measurements and the cubic regression fit within the error of the measurements the cubic would be the regression of choice. Equation Chapter (Next) Section 5

To ensure that each data point was gathered with reasonably low uncertainty, the instruments were chosen and calibrated as discussed in Chapter 4, Apparatus Instrumentation. To ensure that the experiments were conducted in a repeatable manner a well prescripted test methodology was used. This methodology was adapted into a test procedure and allowed the experimenters to follow the same process of each test. As the experimenters were conducting the experiment, a procedure check list was filled out. All observations, problems, and associated file names were written on this document. The completed test procedure was signed by all experimenters involved in conducting the test and then stored will the apparatus's permanent records. The associated digital files containing the raw data are then transferred onto the data processing computer and onto a backup hard drive.

The aforementioned method was developed with the intent to produce complete and quality data. Depending on the institution the method of producing quality data requires different documentation methods. The data collected in this study was not required to meet the documentation requirements of the ASME NQA-1 nuclear data quality requirements, but the burden of documenting calibration, test, and process methods and records that provide information traceability were implemented, which allows the generation of quality data from this apparatus. In the context of this study, quality data is defined as data having a high *degree of goodness*. The users of this data should be supplied with enough information and a complete set of boundary conditions to provide confidence in legitimate conclusions drawn from the data set. To this end, the gathering, processing, and analyzing the data must be handled in deliberate and careful manner.

The setup for a test must be conducted in a repeatable manner. All cables must be plugged into the proper receptacle. Switching the connection of two thermocouples could provide an improper sense of the apparatus functioning for instance if the heater and camera thermocouples were switched the operator would believe that the heater was broken and the camera had melted causing the experiment to be aborted. To ensure repeatability, the infrared camera must be set up in a careful manner. The camera is not rigidly attached to the apparatus, for its protection, and so the process of aiming and calibrating the camera must be done with care for each test. The reflected background temperature, ambient temperature, and distance from the measured object must be determined, recorded, and entered into the camera software before each test. Figure 5.1-1 displays an infrared image of camera’s view when properly setup and aimed. Using this view the center of the spray nozzle can be recorded which will allow the person processing the data to know the relationship between the nozzle center and camera image.

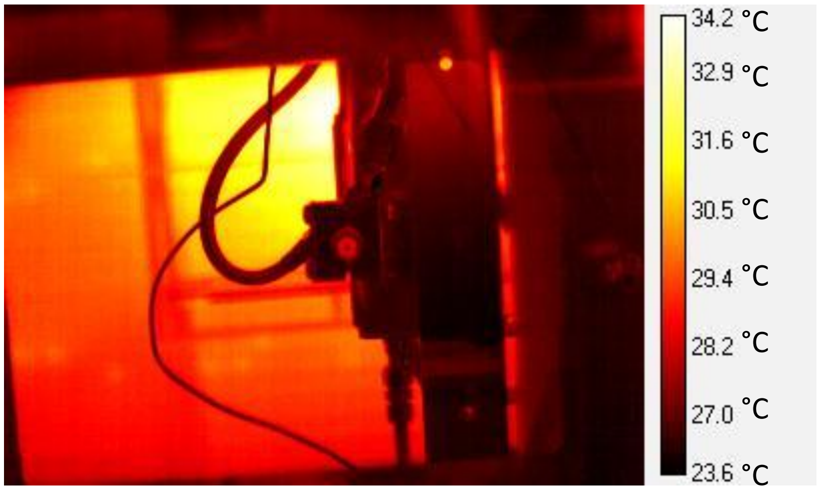


Figure 5.1-1: Infrared image looking through the camera window at the spray nozzle, note this image was gathered before the test plate was installed in the apparatus.

Once the data was experimentally gathered it needed processing. There were two sources of data to draw from, the data gathered by the Agilent box and the data gathered by the Flir camera. The Agilent data was saved in the csv format, which is imported into MS Excel. The data gathered by the infrared camera was stored in a proprietary format exclusive to the ExaminIR program. To process the camera data, the data was exported to csv files, where one csv file corresponded to each camera frame. The time of jet activation is stored in the DAQ data, reported in seconds, and is translated into units of 10 μs to coordinate with the time stamp in the camera data. After synchronizing the time stamps, the cooling transient can be extracted from the camera data using the Thermal Viewer software written by the author. With the camera data properly processed, the actual analysis can be conducted, which is discussed in detail in Chapter 6, Data Analysis.

## 5.2 Data from the Test Case

The data used in this study was collected during an experiment that was conducted during August 8th and 9th of 2009. The objective of this test was to conduct consecutive tests using the same setup parameters. Having multiple runs with the same input parameters will allow a basic assessment of the facility's ability to reproduce a given data set. Due to problems with timing only one run was successfully completed on August 8th and the three runs were completed successfully on August 9th. The August 8th run was collected with the infrared camera using its high calibration range. This test will be referred to as High 1 from here on. For the tests on August 9th, one was collected with the infrared camera in the high calibration and the other two were collected with the camera on the medium calibration. These tests will be referred to as High 2, Medium 1, and Medium 2 respectively. Collectively, these four data sets encompass the facility's operating range. The test parameters, data, and variance will be discussed below.

### 5.2.1 Test Parameters and Statistics

The following list provides the apparatus settings and boundary conditions the test case data for the test conducted 08/10/2009 and 08/11/2009:

| | |
|---|---|
| Nozzle Type: | ¼ FJJXP040S303 |
| Nozzle Qty: | 1 |
| Water Flow Rate: | 4.0 liters per minute |
| Camera Range: | High (1200 to 200 °C) Runs High 1 and 2 |
| | Medium (350 to 0 °C) Runs Medium 1 and 2 |
| Test Plate Number: | MHI-TP-030 |
| Nozzle – Plate Separation: | 83 mm |
| Nozzle Orientation: | Flow Path normal to test plate |
| Initial Plate Temperature: | 700 °C |

The fluid temperature, fluid pressure ahead of the nozzle, and fluid flow rate was measured and recorded by the DAS. Table 5.2-1 contains the mean values for each of these parameters for each of the four tests. The standard deviation of each set of data is also included in the table, which provides a measure of the quality of the data. Plots of the fluid temperature, fluid pressure, and fluid flow rate for each of the four tests with respect to time can be found in Figure 5.2-1, Figure 5.2-2, and Figure 5.2-3, respectively. It should be noted that the large standard deviation in the fluid pressure and flow rate data is expected since the cooling system is not at steady state during the cooling transient. The activation of the jet causes a pressure spike at the nozzle which causes the system pressure to oscillate for the first minute of operation. This pressure wave can be observed by examining Figure 5.2-2.

Table 5.2-1: Test Fluid parameters

| | Fluid Temperature ( °C ) | | Fluid Pressure (PSI) | | Fluid Flow Rate (L/min) | |
|---|---|---|---|---|---|---|
| | Mean | Std Dev | Mean | Std Dev | Mean | Std Dev |
| High 1 | 29.89 | 0.05 | 26.684 | 7.273 | 3.877 | 0.534 |
| High 2 | 28.91 | 0.07 | 28.365 | 3.533 | 3.922 | 0.607 |
| Medium 1 | 28.88 | 0.06 | 27.682 | 4.086 | 3.848 | 0.528 |
| Medium 2 | 29.18 | 0.05 | 28.201 | 4.250 | 3.824 | 0.523 |

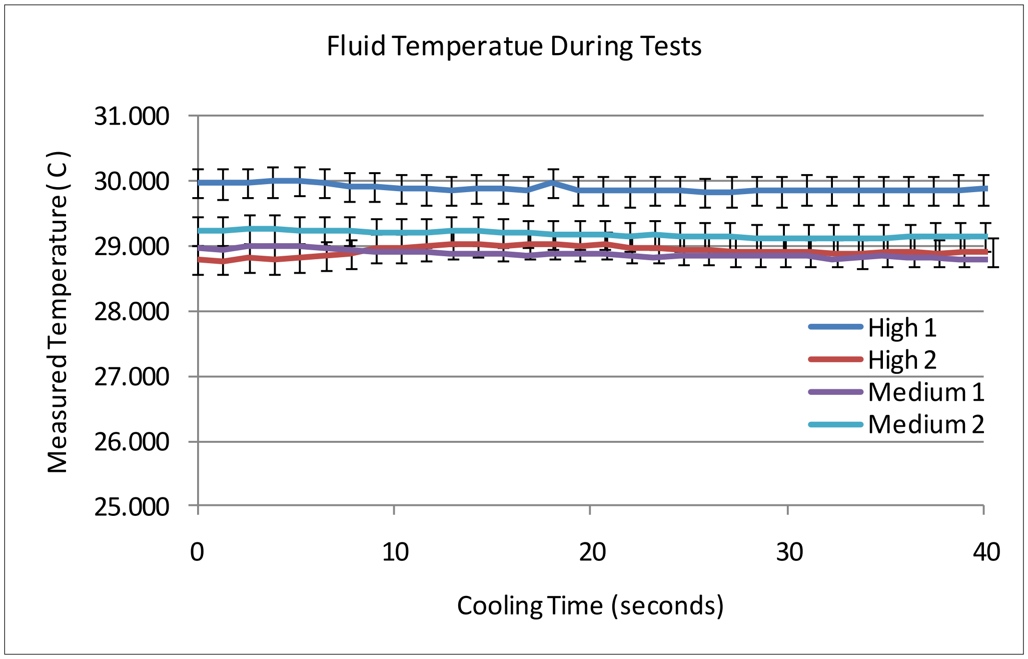


Figure 5.2-1: Plot of the fluid temperature during the 4 tests.

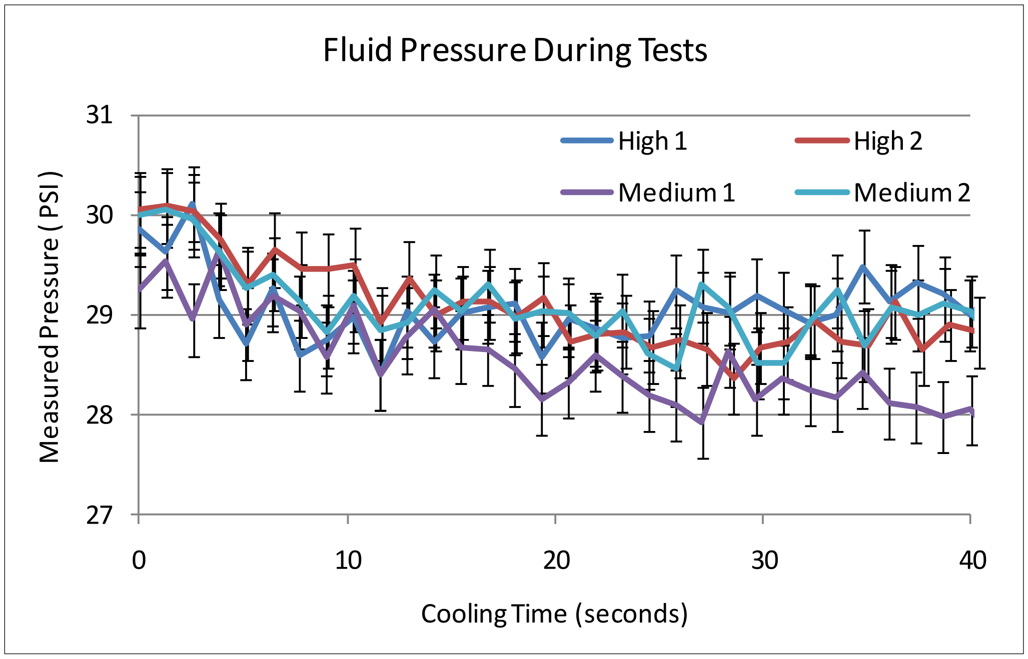


Figure 5.2-2: Plot of the fluid pressure during the four tests.

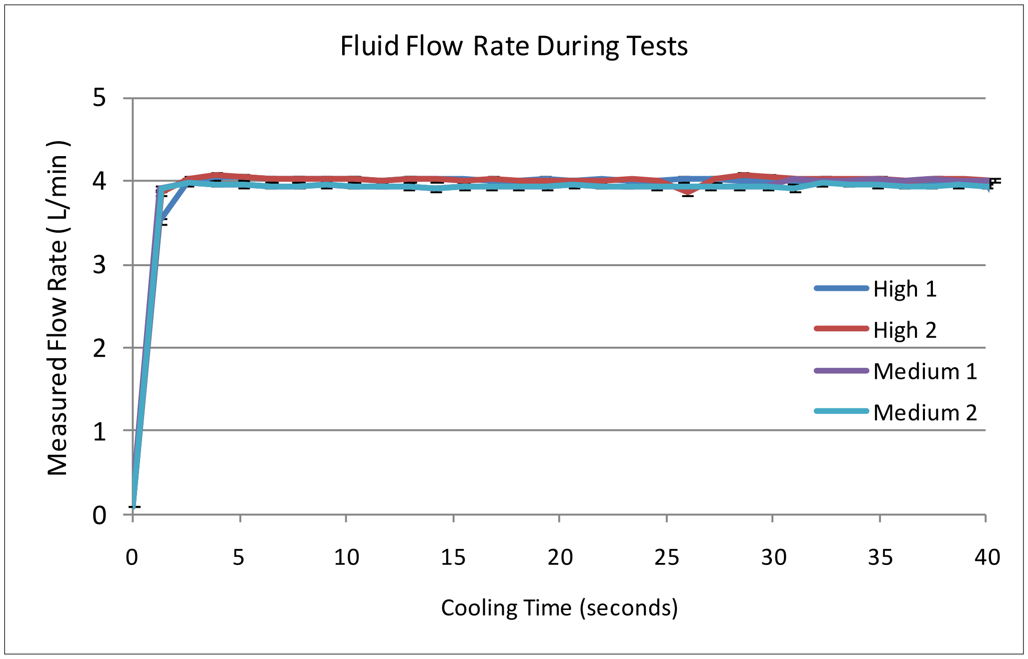


Figure 5.2-3: Plot of the fluid flow rate during the four tests.

### 5.2.2 Test Data

During the cooling transient there are 33.5 million unique temperature measurements gathered by the infrared camera and simply presenting a table of the data would not prove useful. In an effort to visualize the transient, a series of images was created using three cutting planes passing through the temperature field; see Figure 5.2-6 through Figure 5.2-26. This series of images are composed of three orthogonal planes that pass through the data field and a diagram of the cutting plane orientation can be seen in Figure 5.2-4. The series images uses the same geometry as depicted in the figure and are created where the X and Y axis represent space and the Z axis represents time. Figure 5.2-6, represents time = 0, when the jet was initiated and progress with one image per second until time = 20 seconds when the jet was terminated.

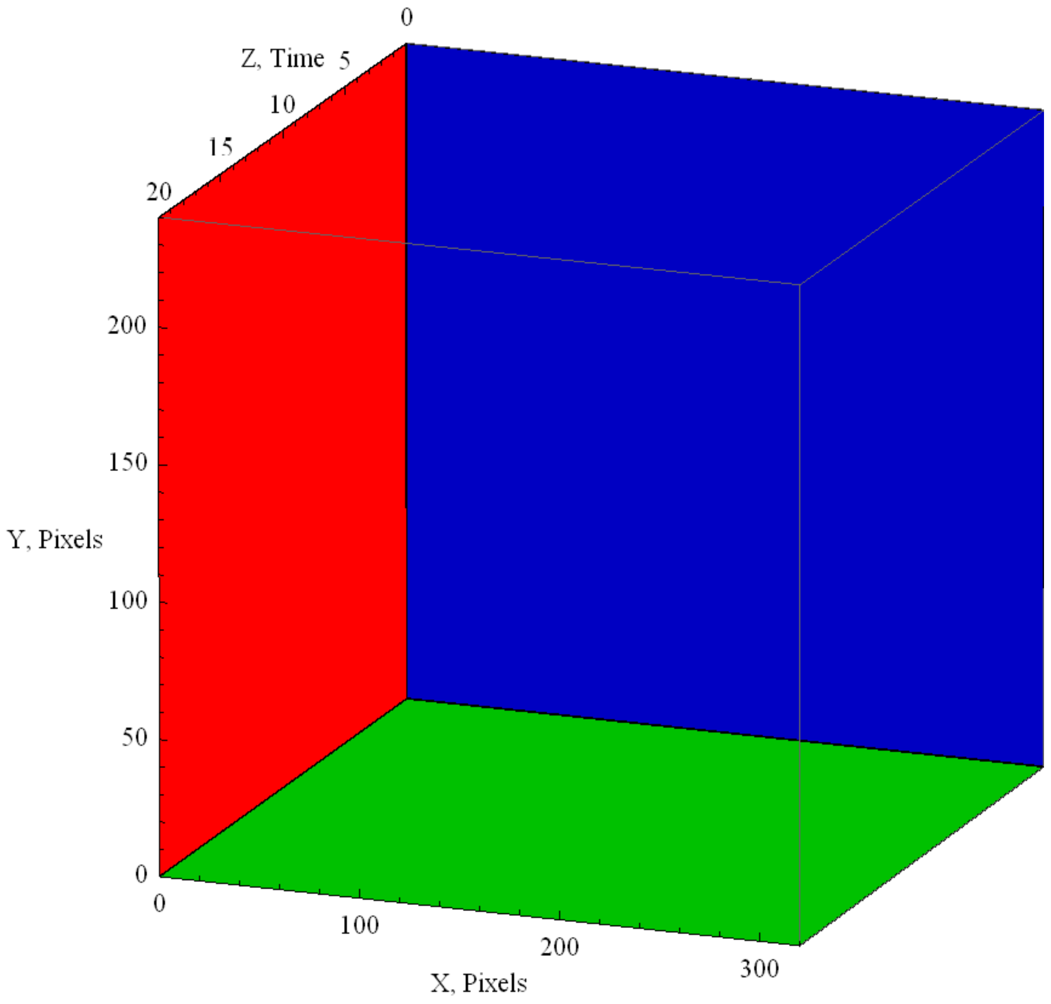


Figure 5.2-4: Diagram of the cutting plane orientation: XY Plane (blue) changes with respect to time, YZ Plane (red), and ZX Plane (green) change with respect to position.

The color applied to each pixel in the cutting planes correlates to a temperature measurement at its given location. This color was set by sampling the temperature data and applying the heated object coloring scale to imply the temperature at that given point. This coloring method was designed to represent the colors of a piece of metal as it is heated from a solid to a liquid. At room temperature the metal is darkly colored and as it is heated the metal starts to glow red, then yellow, and finally white in the liquid state. A diagram of this color scale can be found in Figure 5.2-5.

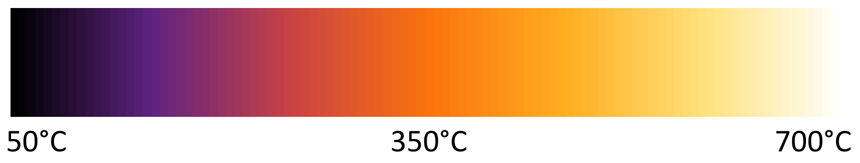


Figure 5.2-5: Diagram of the heated object coloring scale, color at far left represents 50°C and far right represents 700°C and the temperatures are linearly distributed across the color range.

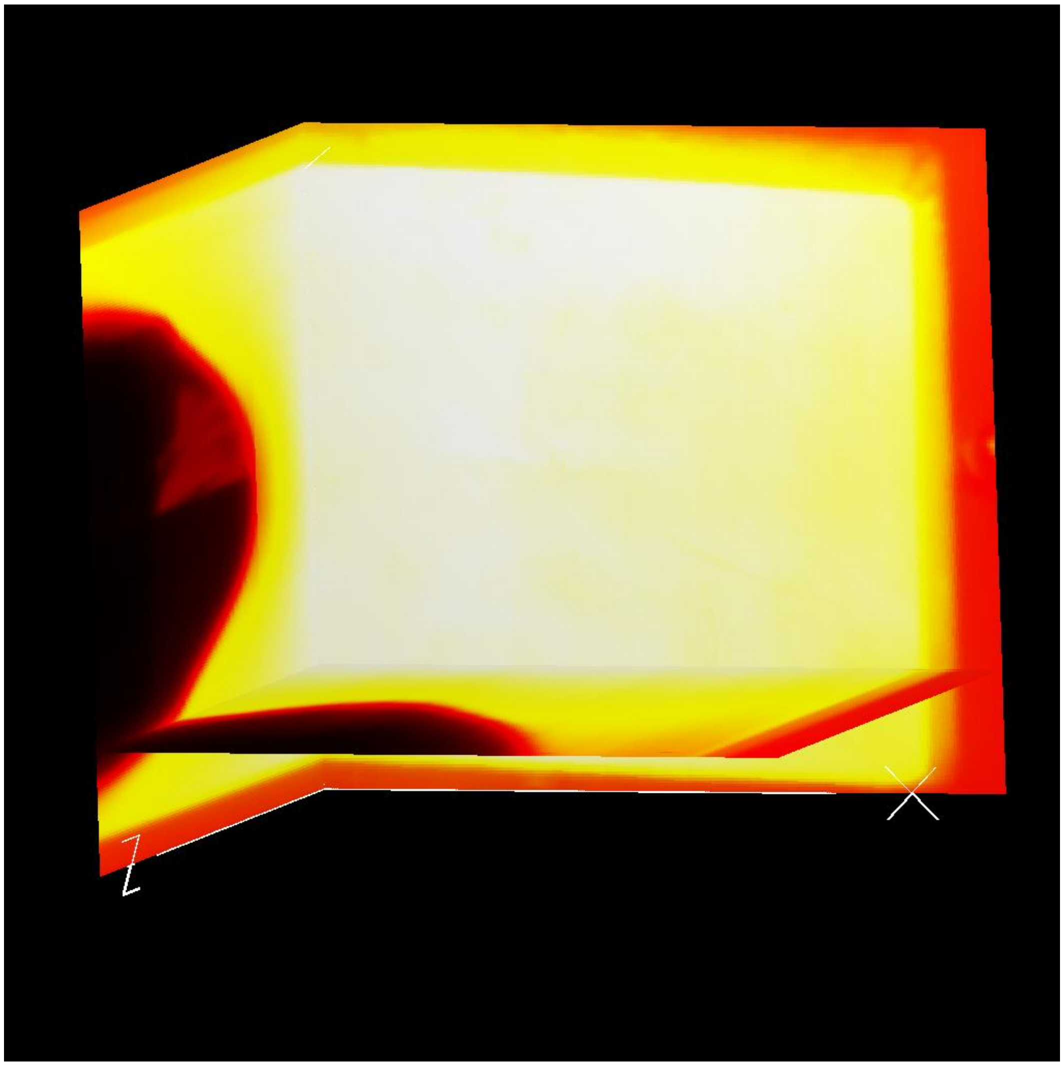


Figure 5.2-6: Render of the temperature field at t=0, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

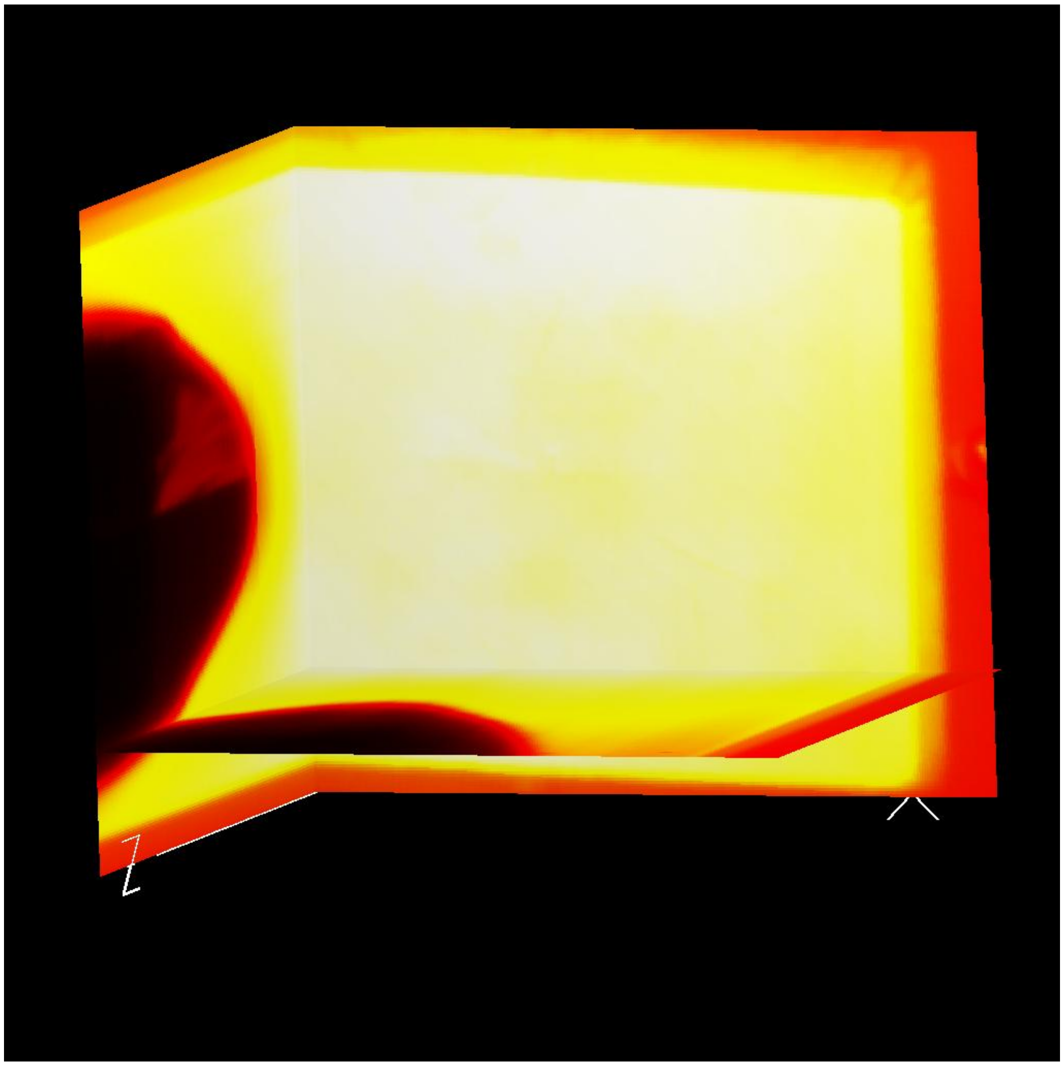


Figure 5.2-7: Render of the temperature field at t=1, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

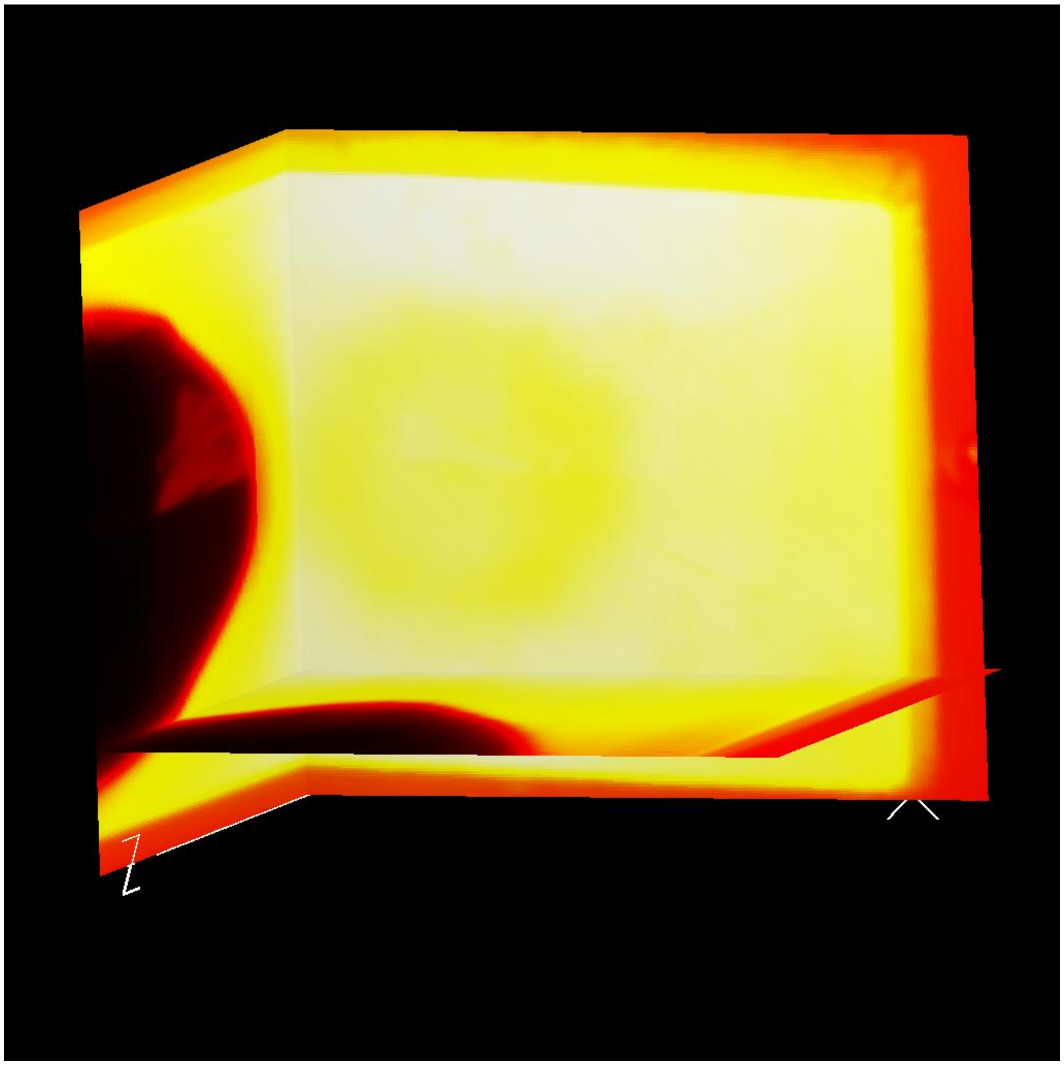


Figure 5.2-8: Render of the temperature field at t=2, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

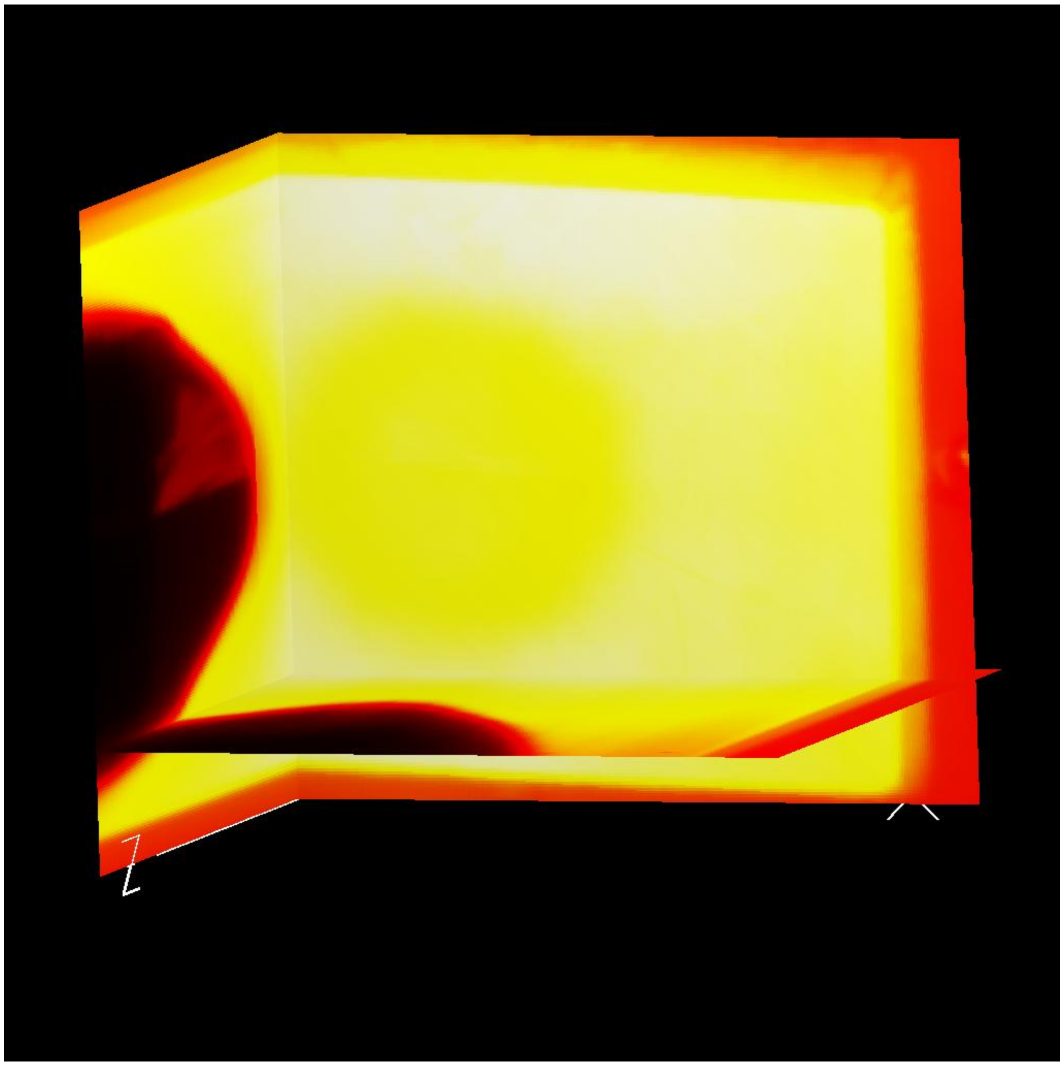


Figure 5.2-9: Render of the temperature field at t=3, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

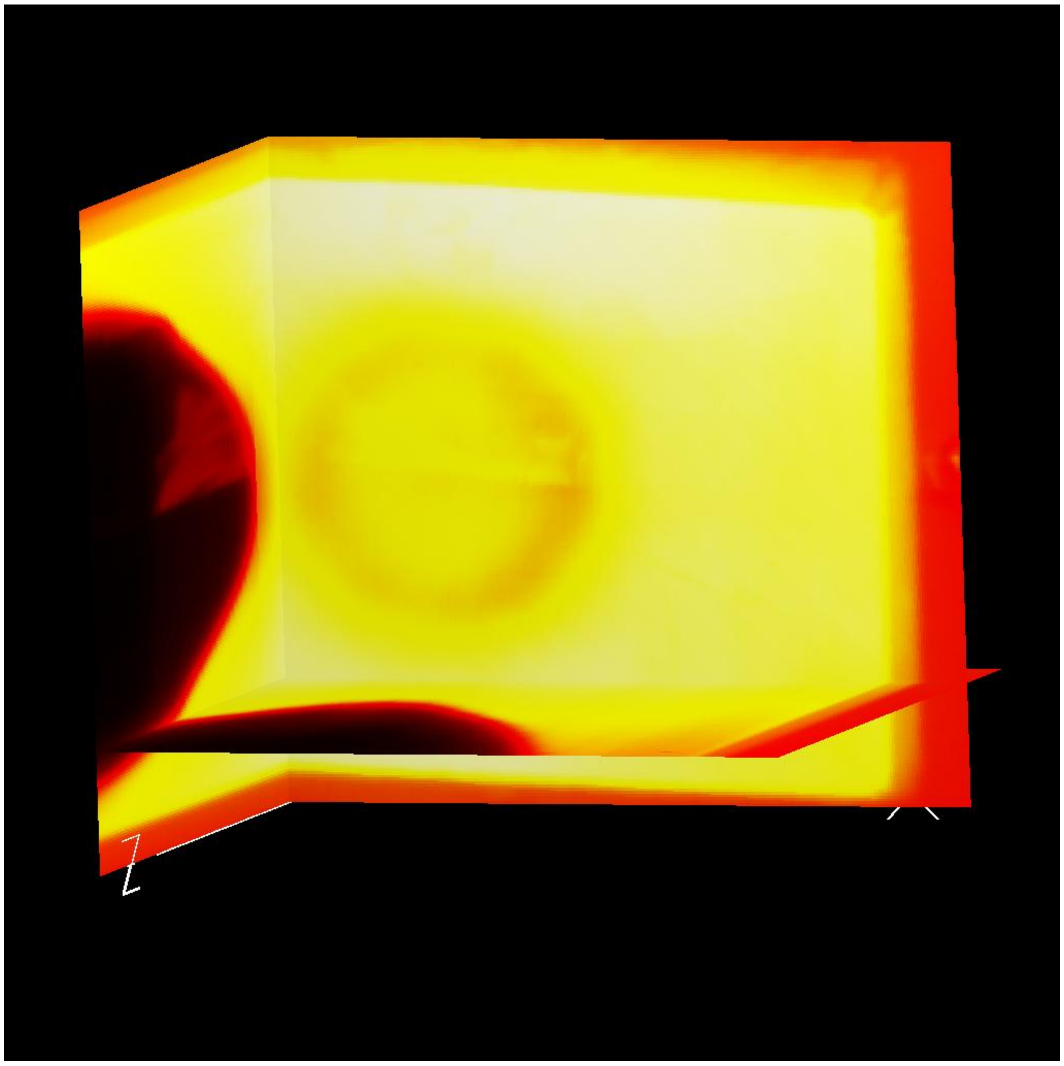


Figure 5.2-10: Render of the temperature field at t=4, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

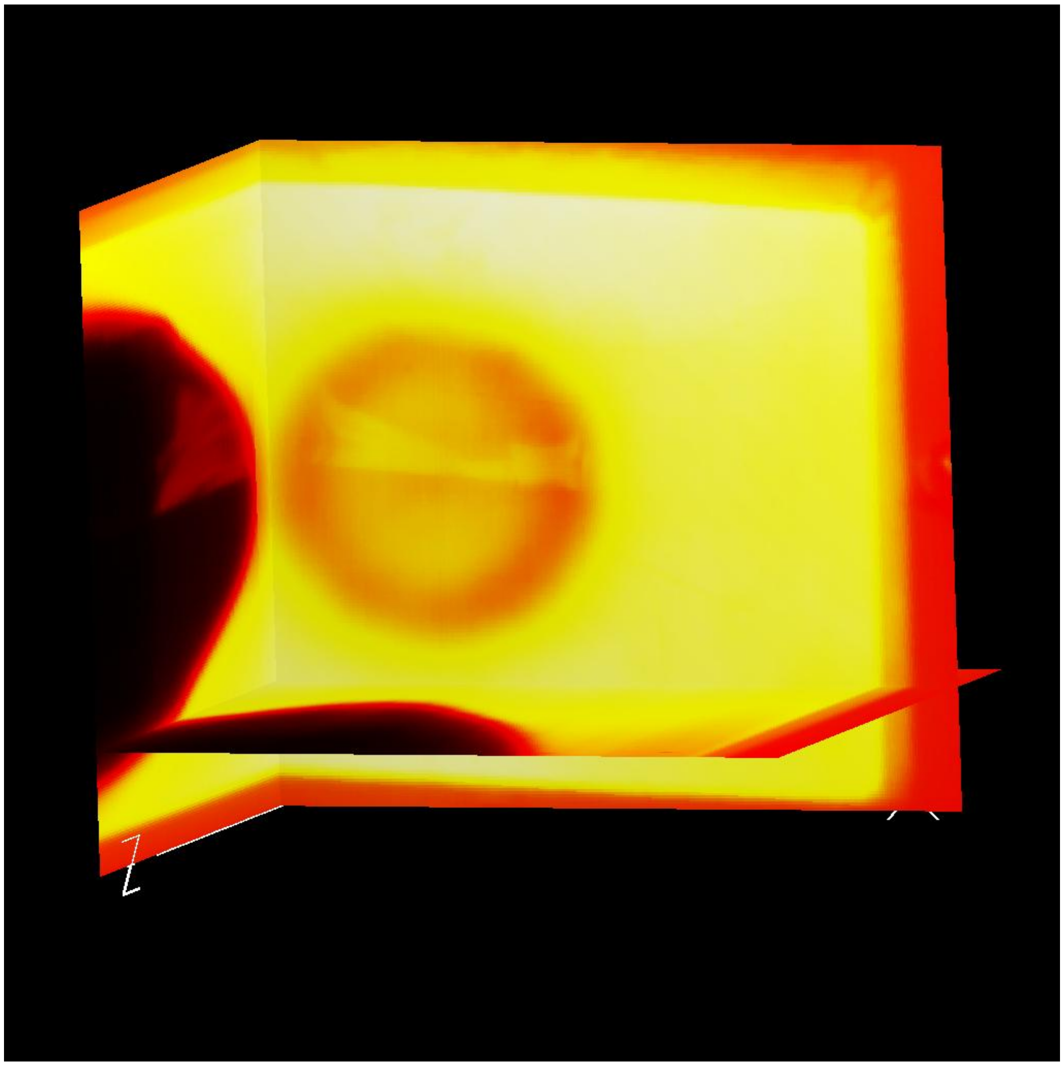


Figure 5.2-11: Render of the temperature field at t=5, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

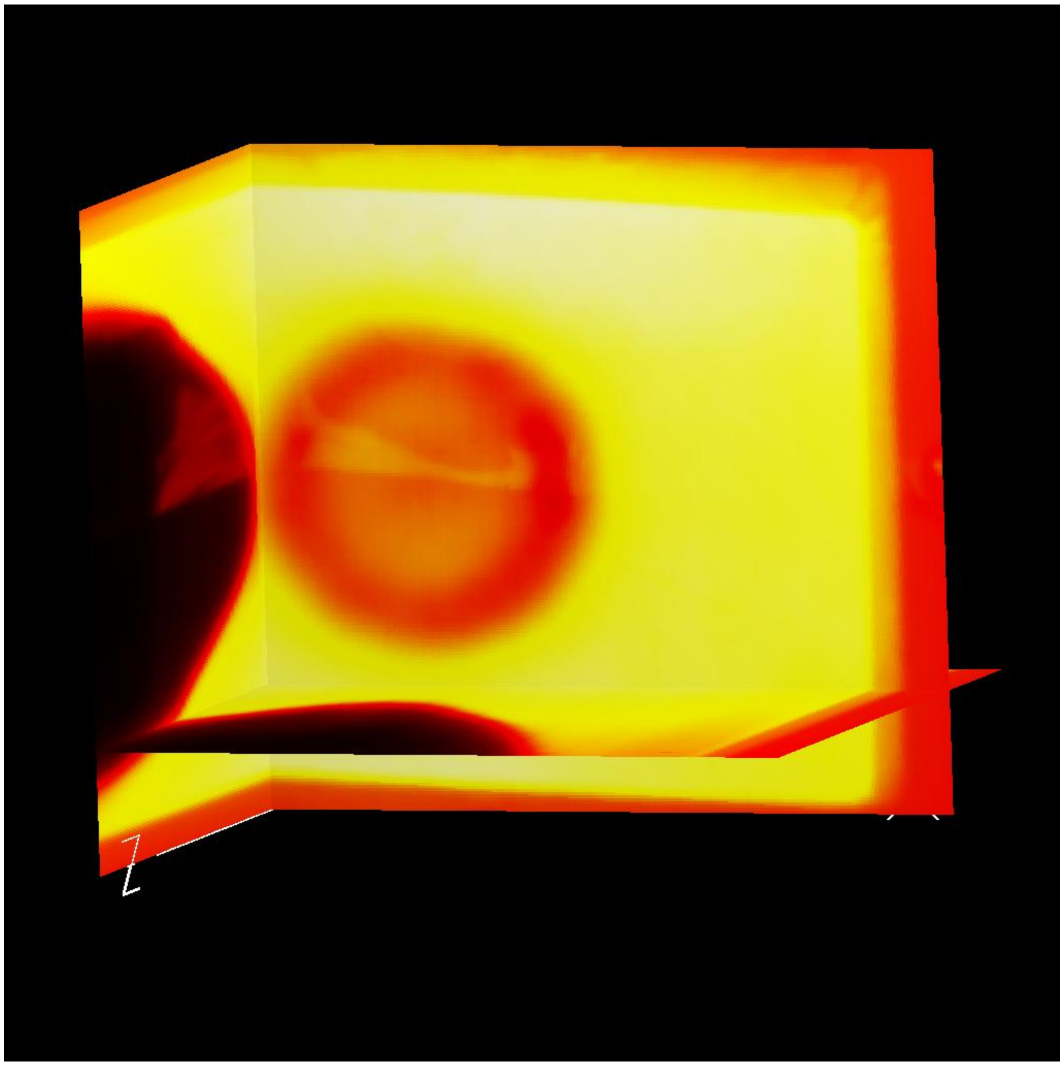


Figure 5.2-12: Render of the temperature field at t=6, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

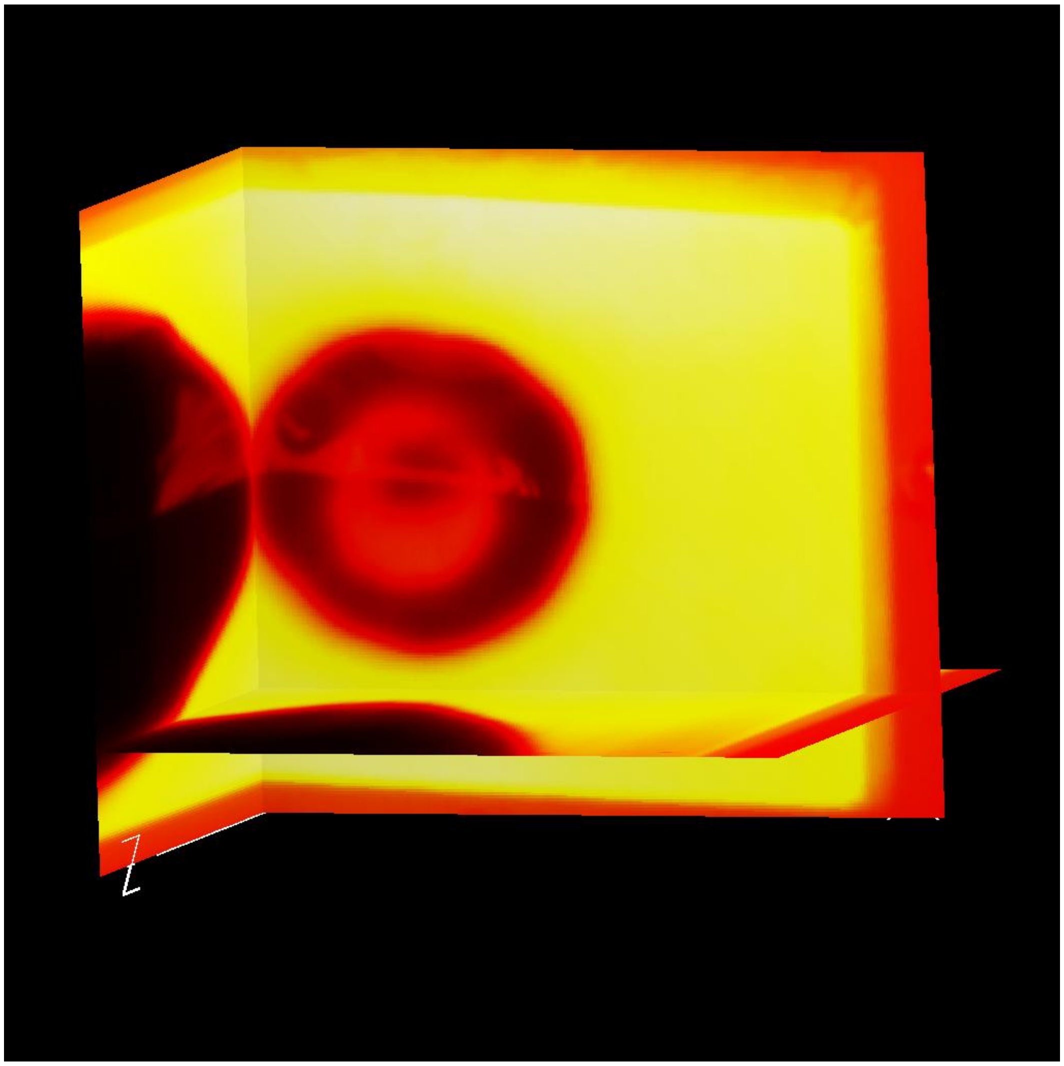


Figure 5.2-13: Render of the temperature field at t=7, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

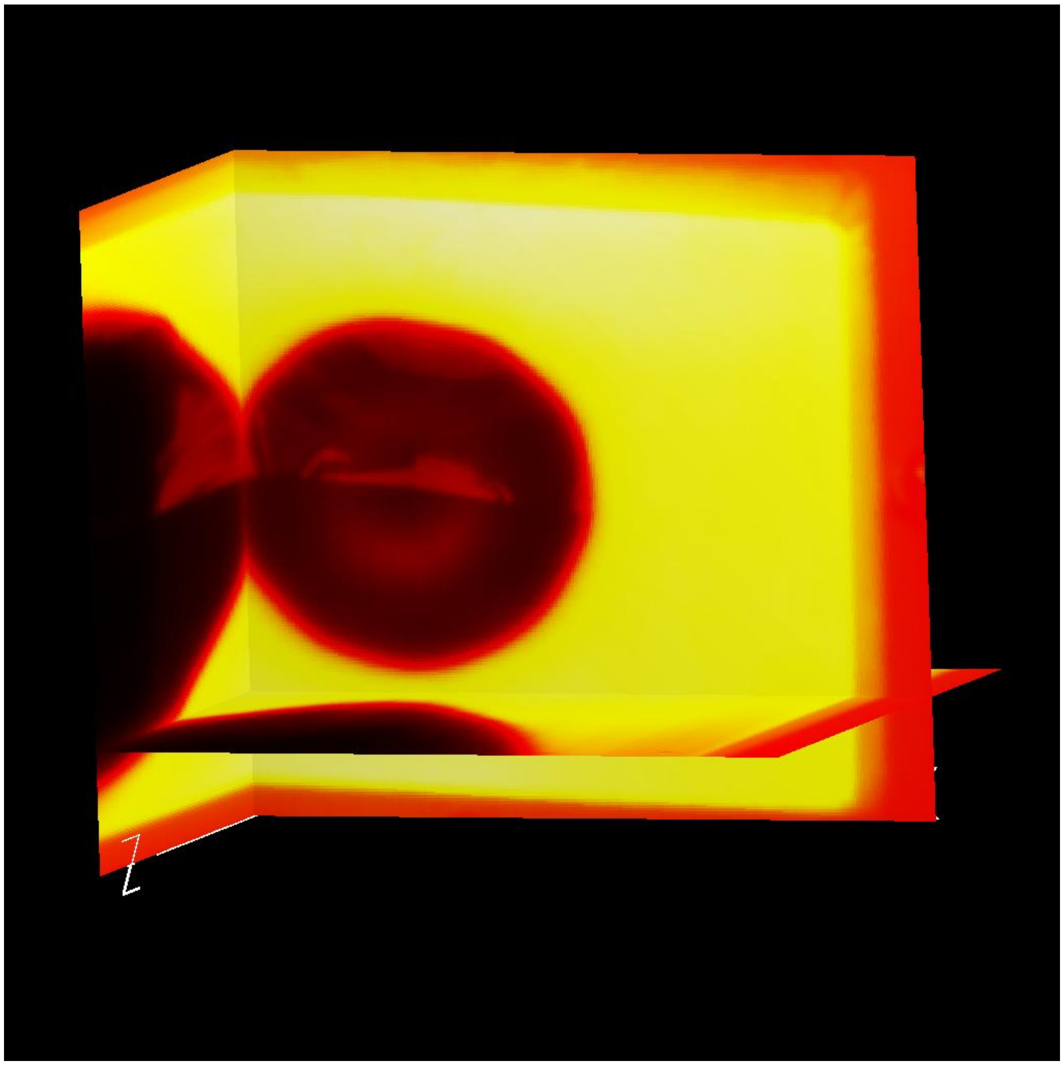


Figure 5.2-14: Render of the temperature field at t=8, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

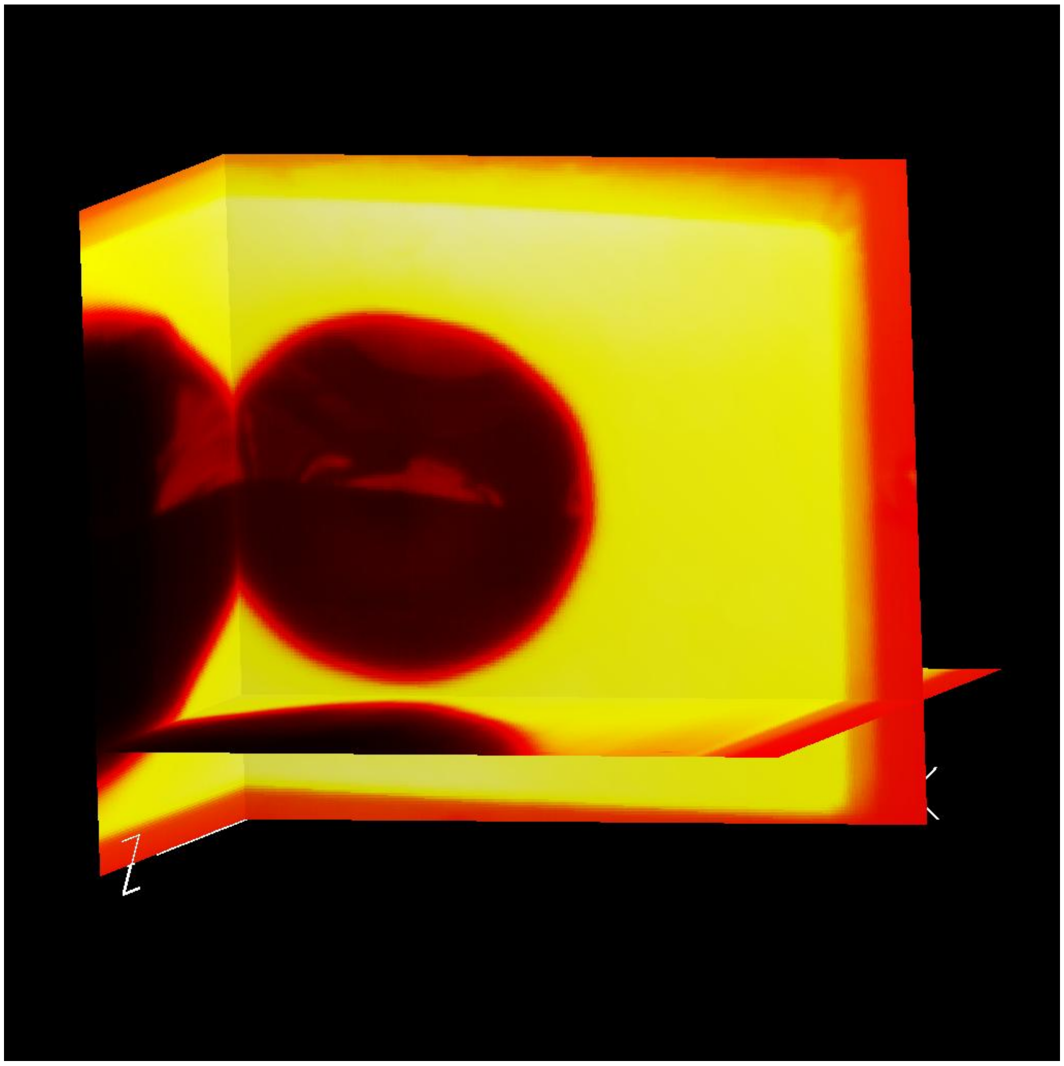


Figure 5.2-15: Render of the temperature field at t=9, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

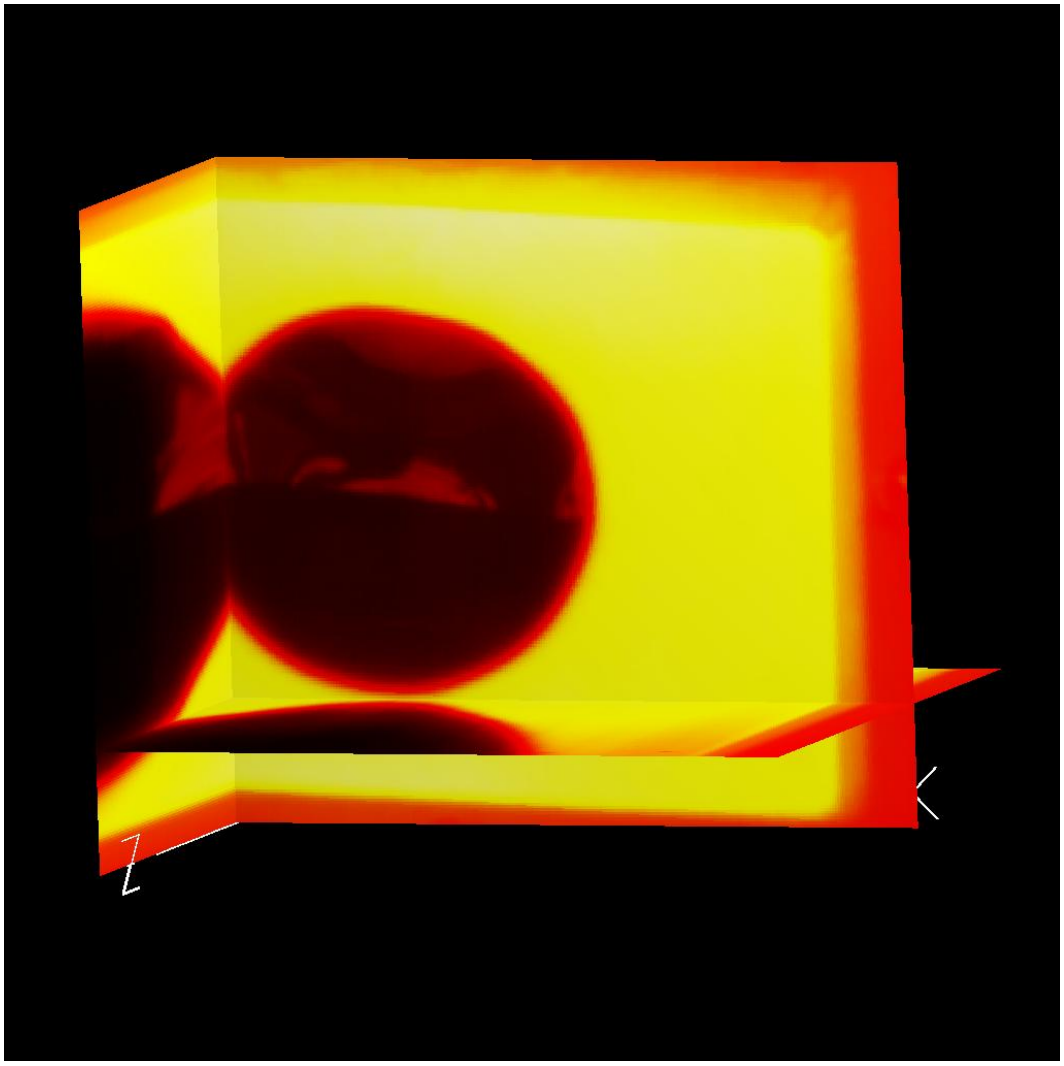


Figure 5.2-16: Render of the temperature field at t=10, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

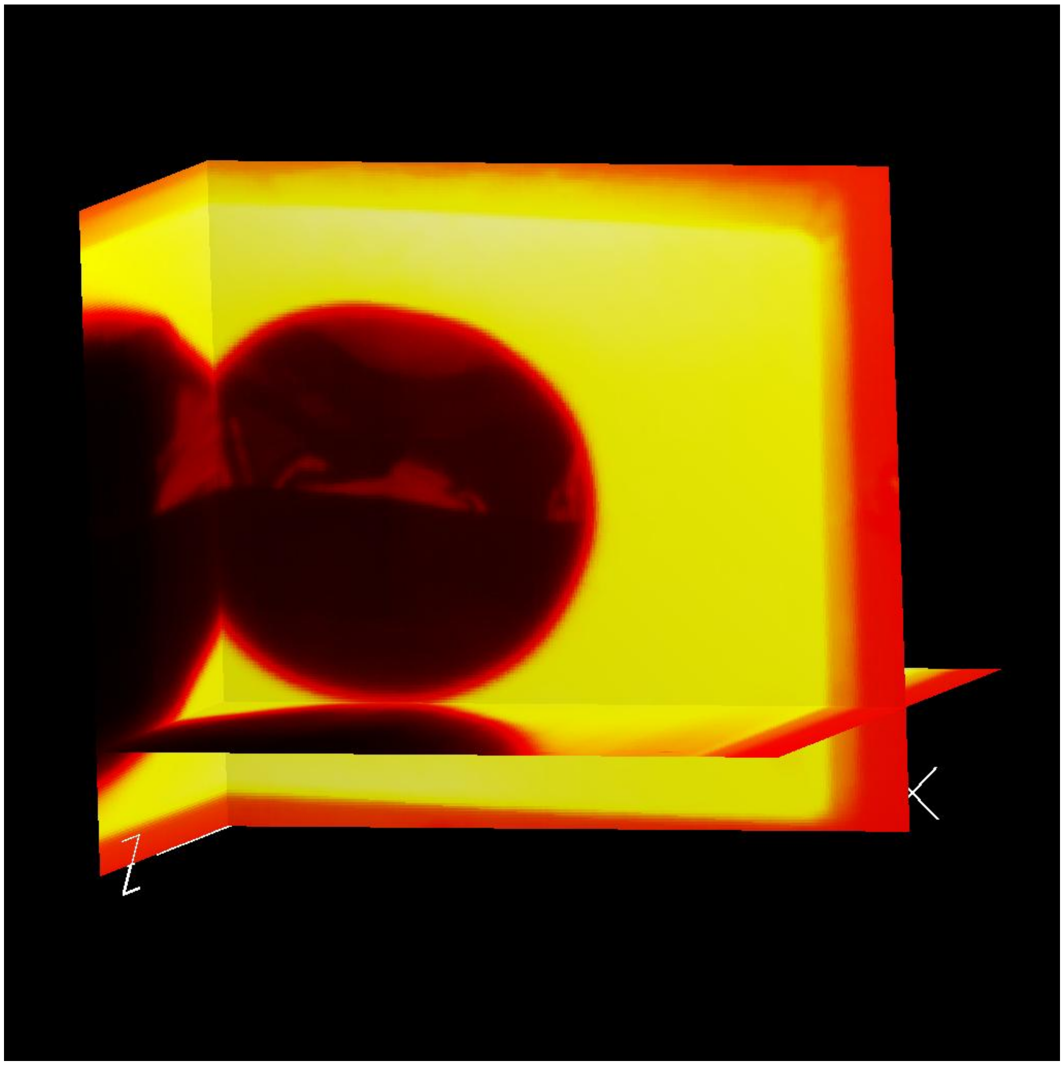


Figure 5.2-17: Render of the temperature field at t=11, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

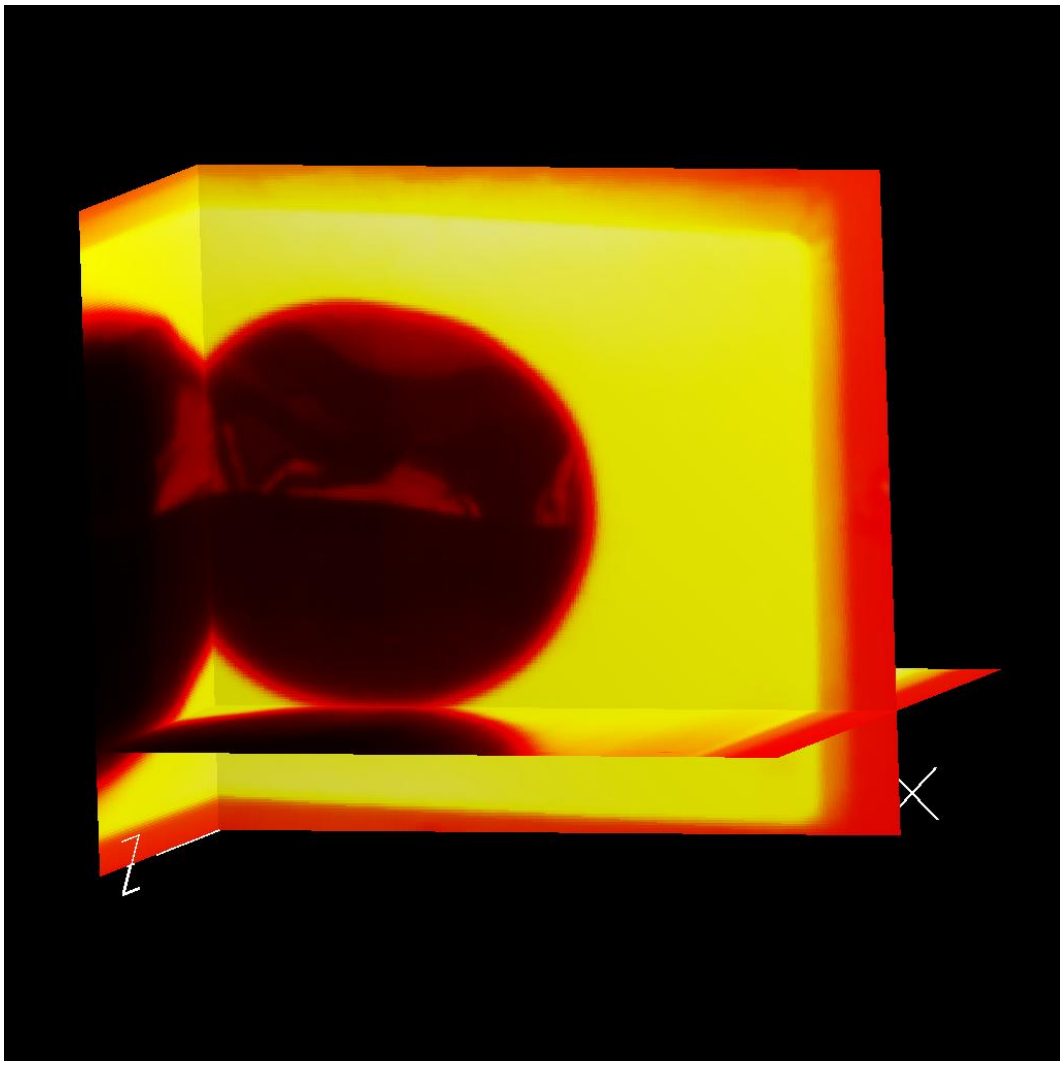


Figure 5.2-18: Render of the temperature field at t=12, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

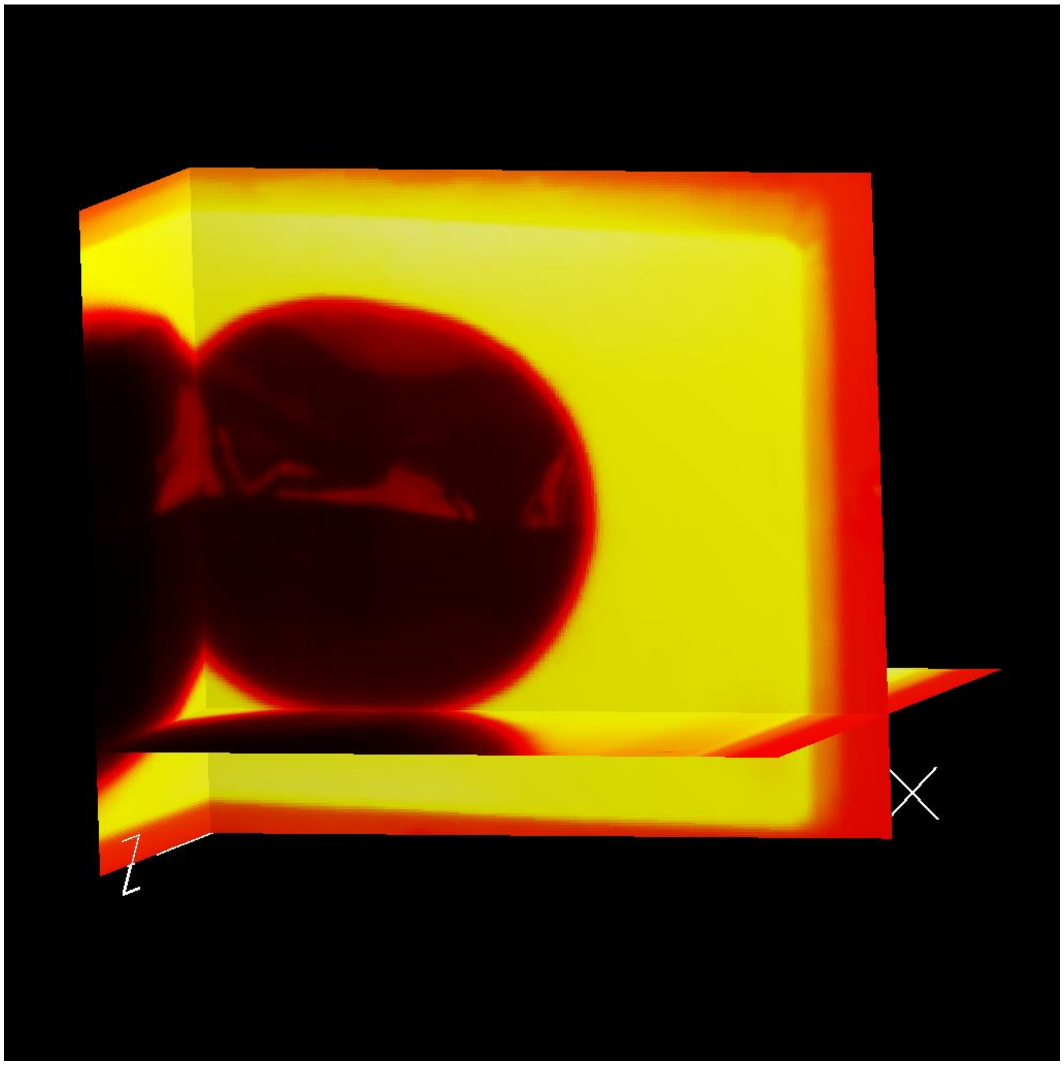


Figure 5.2-19: Render of the temperature field at t=13, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

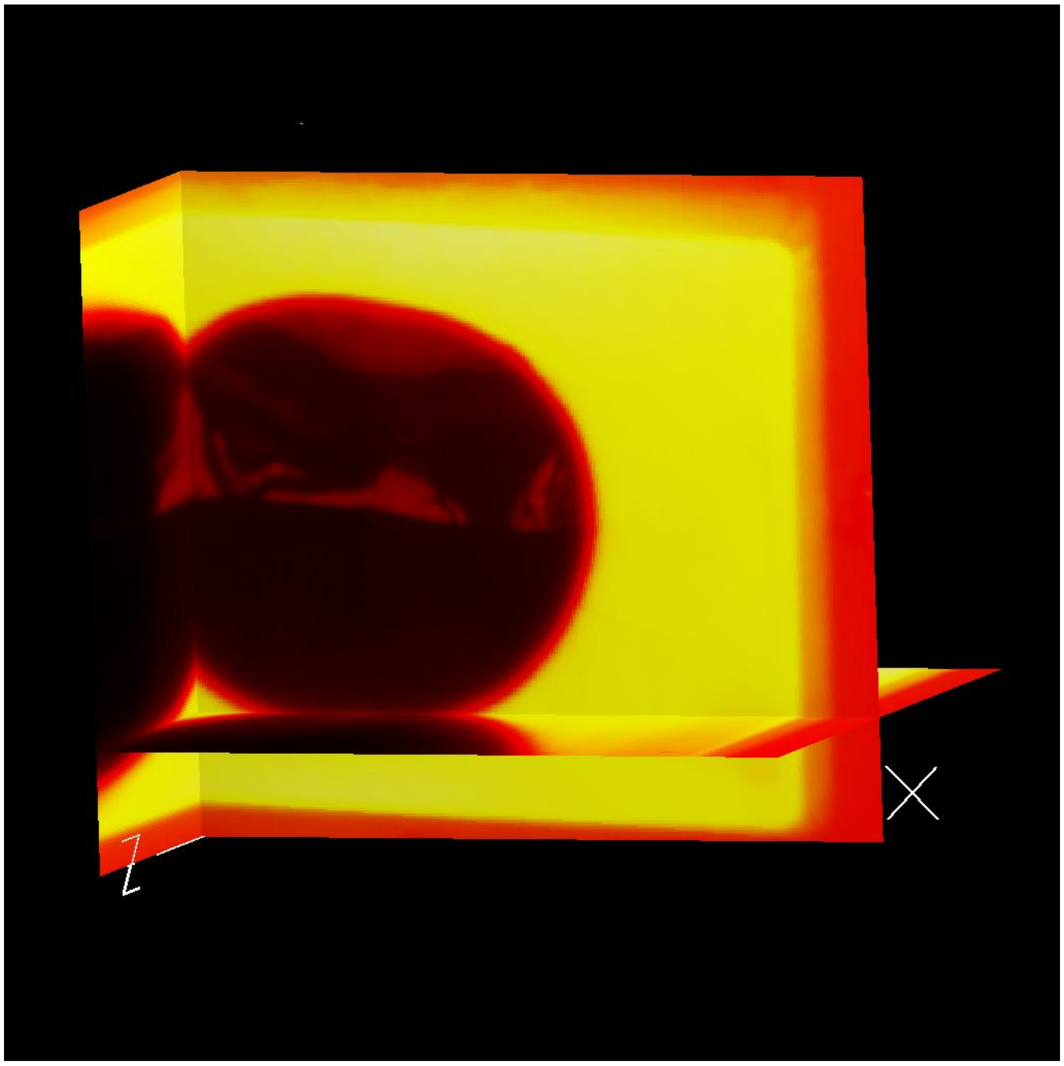


Figure 5.2-20: Render of the temperature field at t=14, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

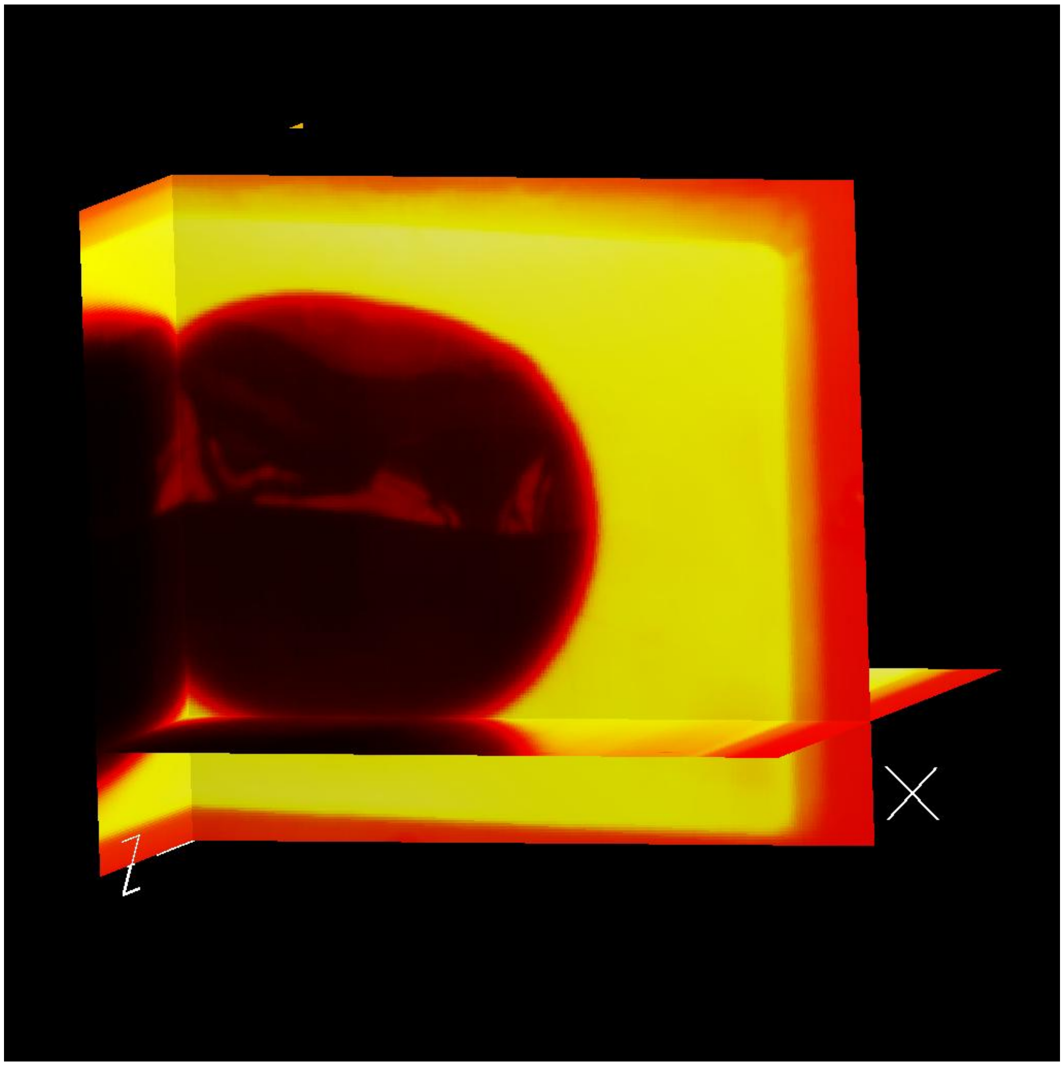


Figure 5.2-21: Render of the temperature field at t=15, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

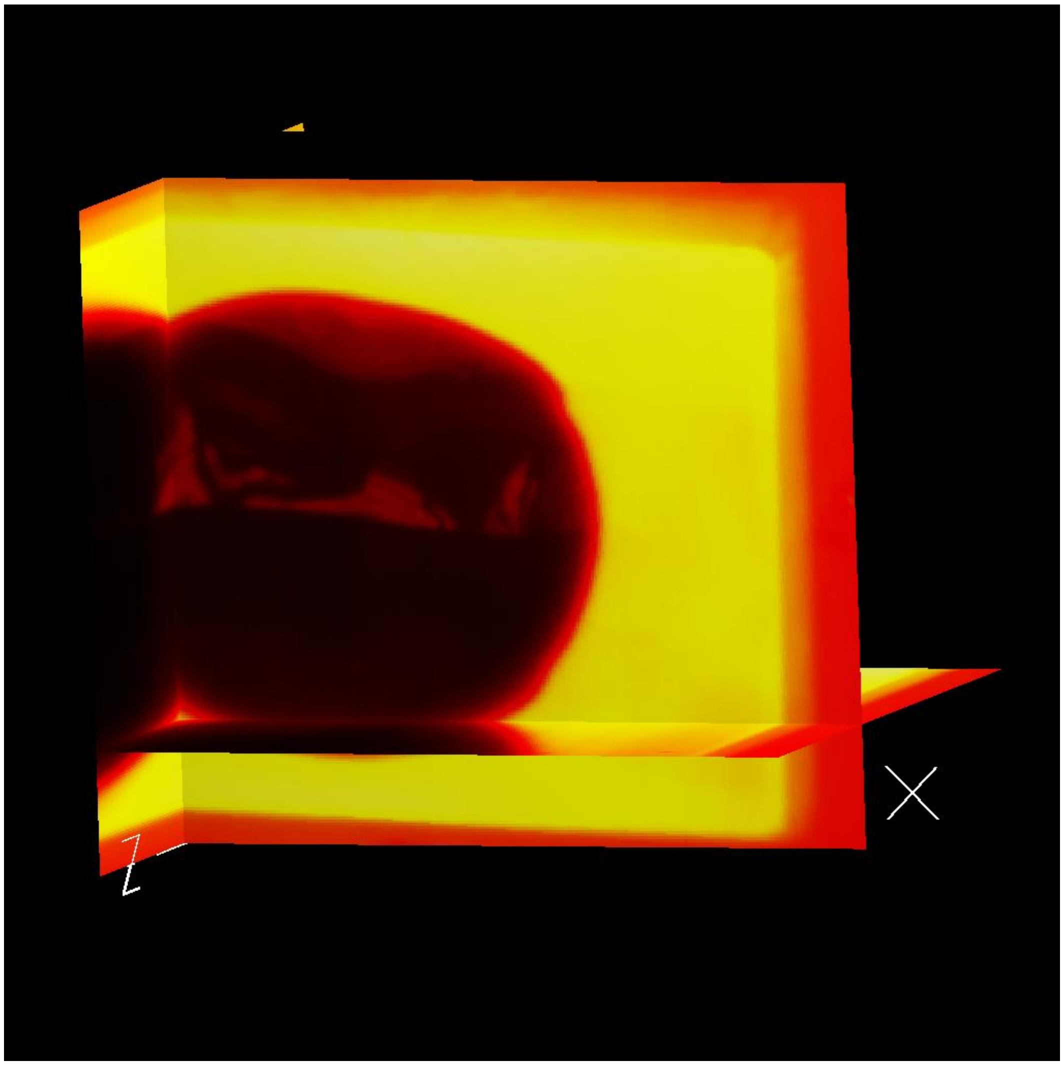


Figure 5.2-22: Render of the temperature field at t=16, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

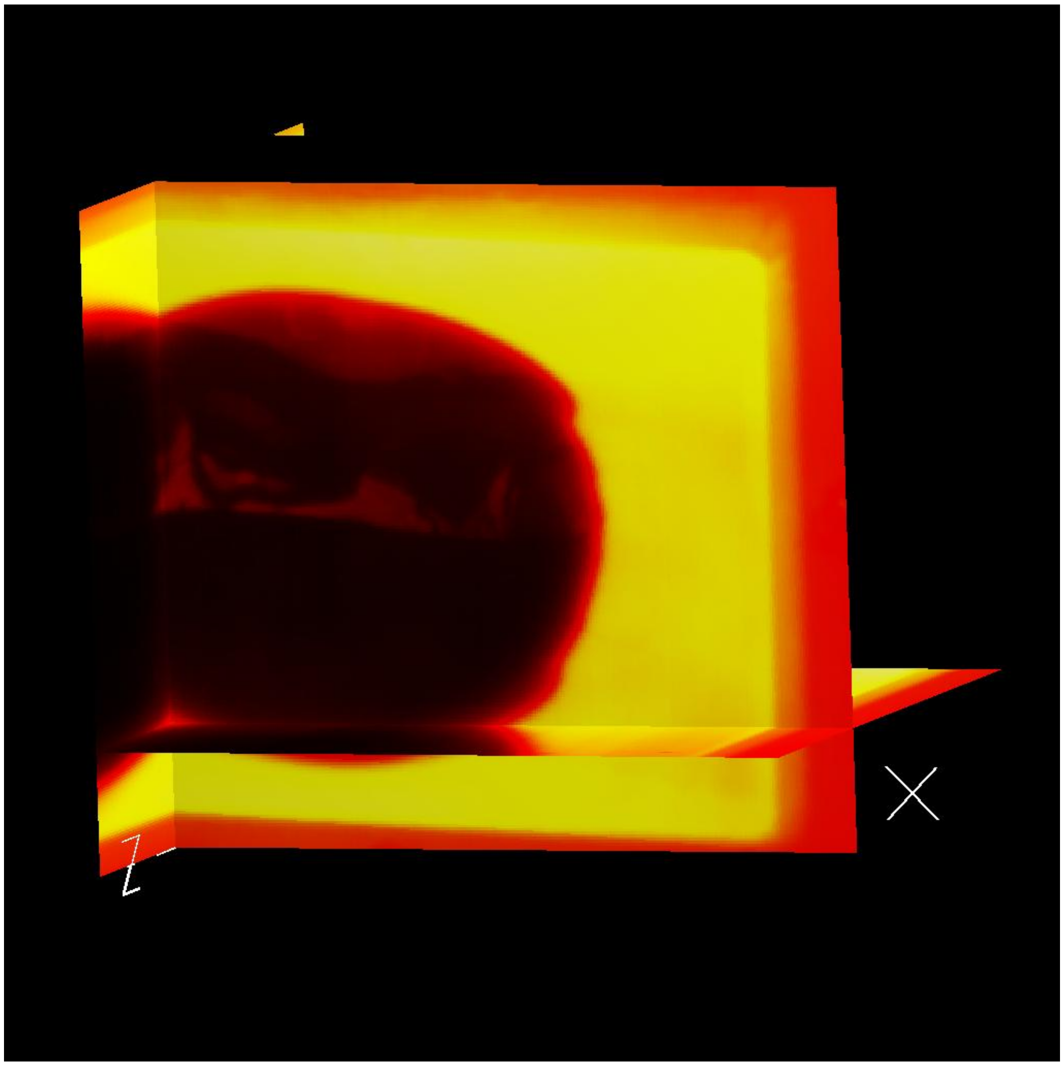


Figure 5.2-23: Render of the temperature field at t=17, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

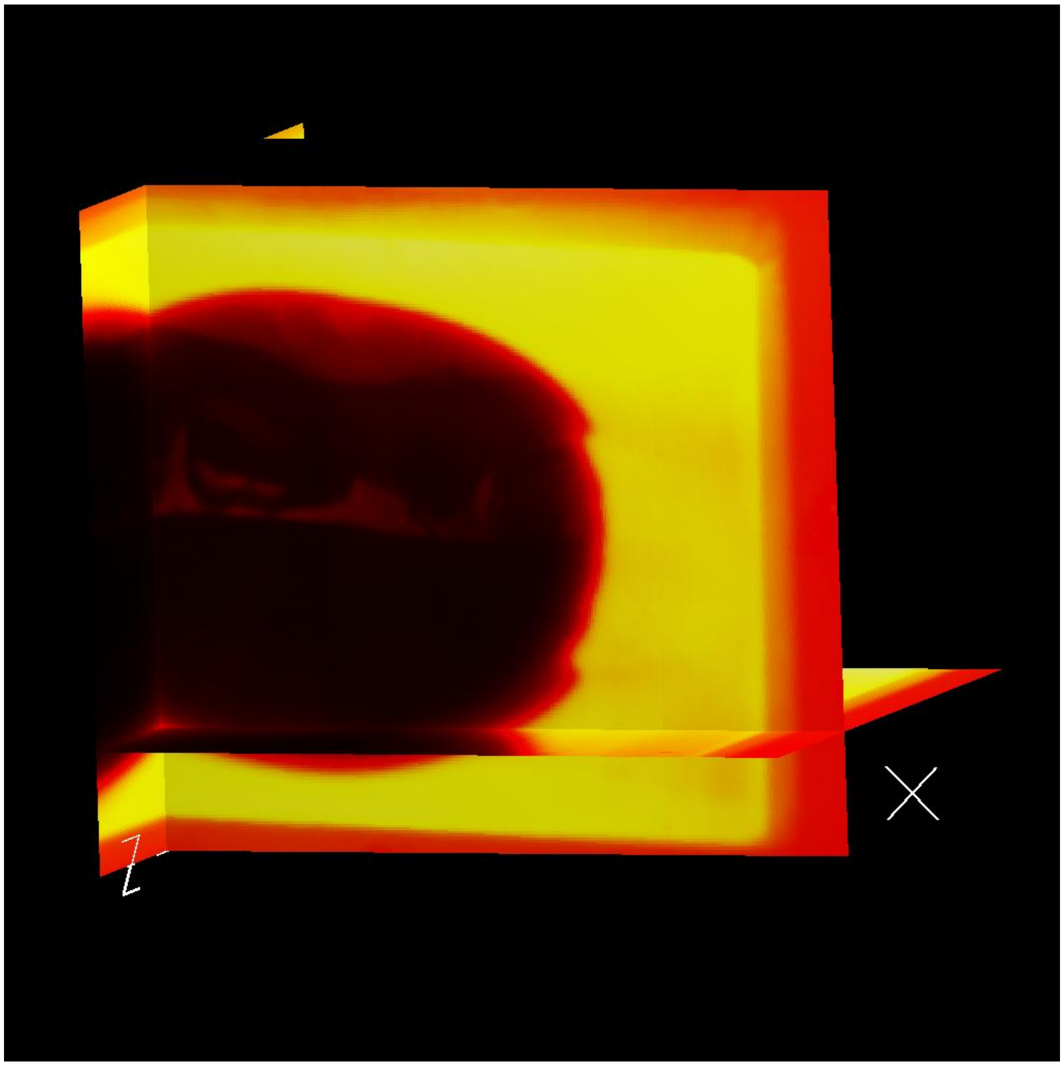


Figure 5.2-24: Render of the temperature field at t=18, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

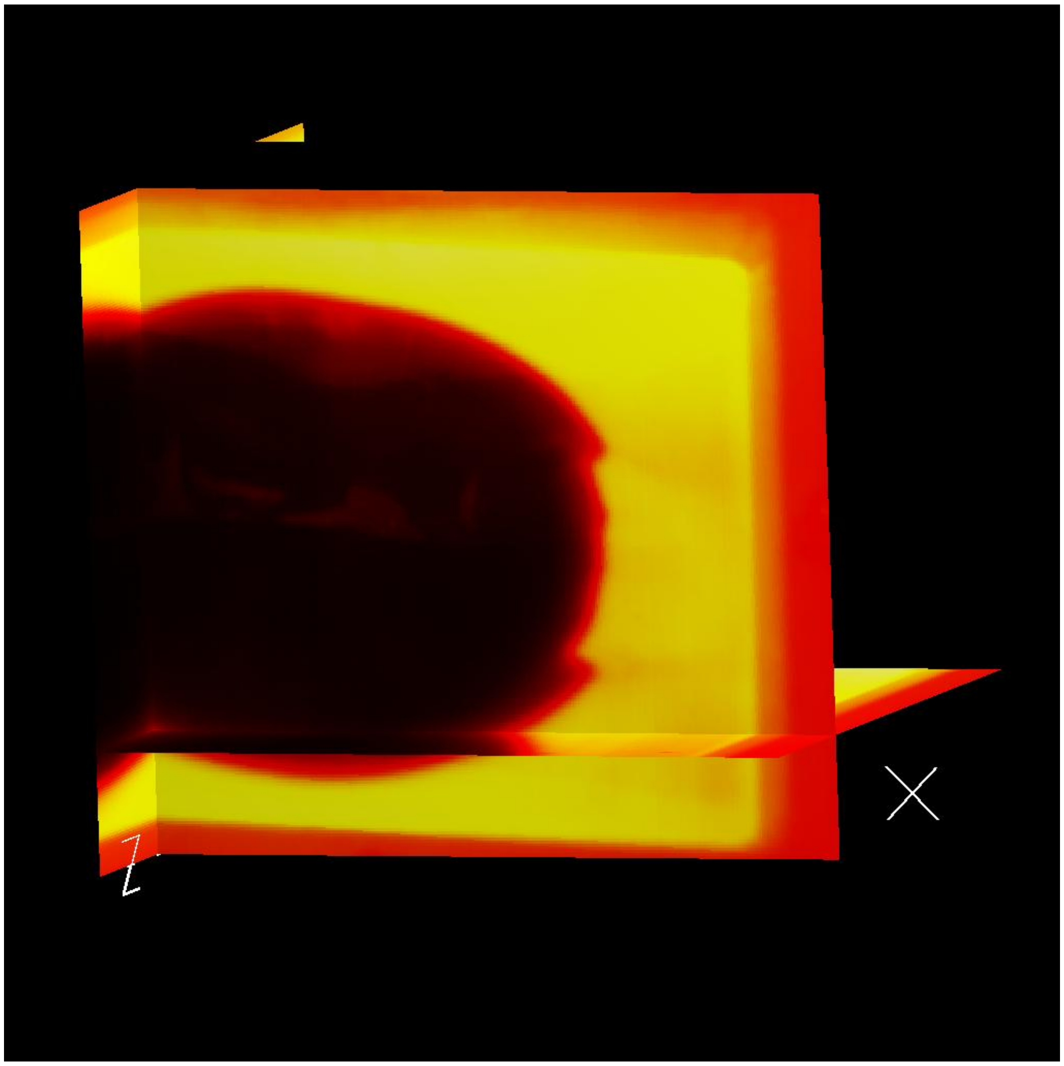


Figure 5.2-25: Render of the temperature field at t=19, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

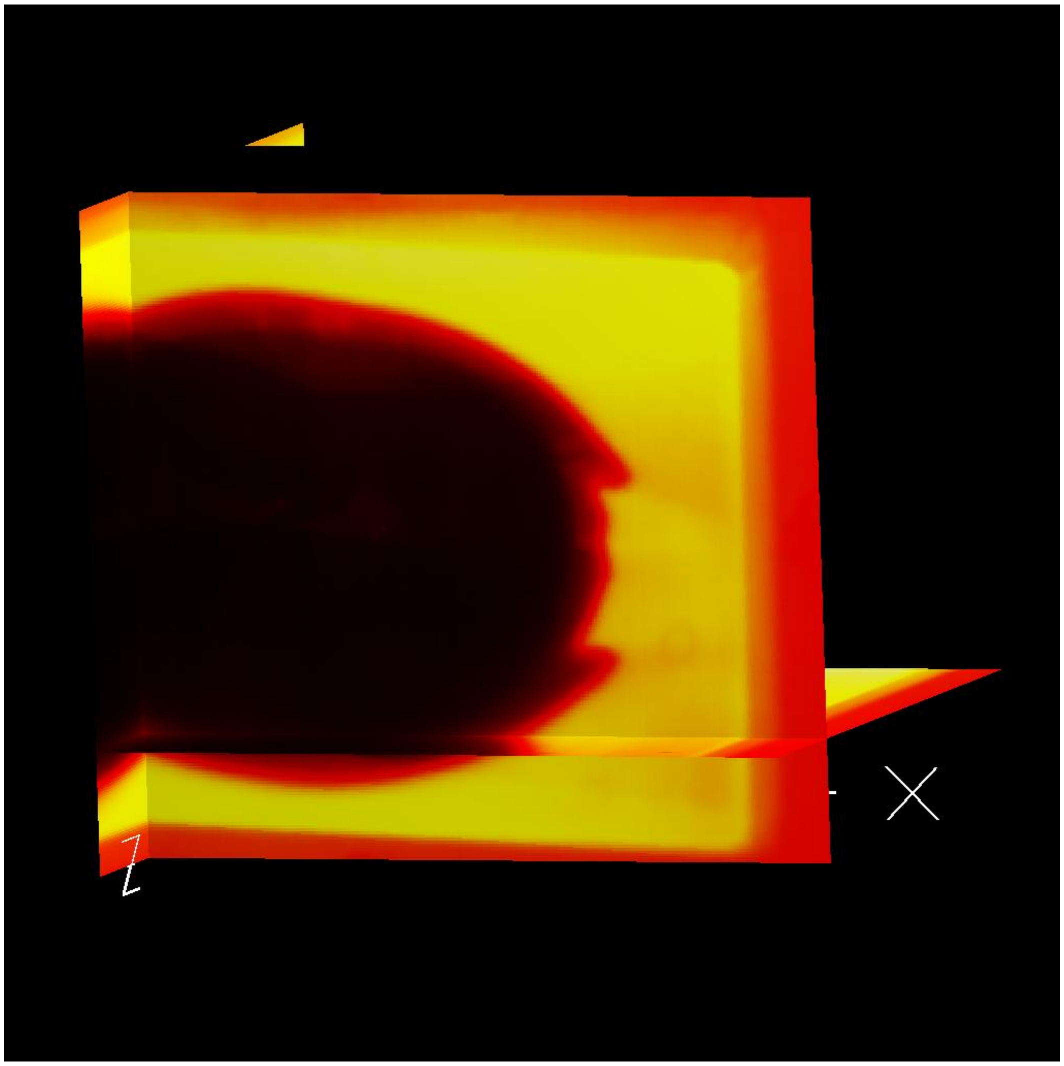


Figure 5.2-26: Render of the temperature field at t=20, using three cutting planes and the heated object coloring scale. The X and Y axes represent space and the Z axis represents time.

## 5.3 Basic Cooling Transient

Using the basic heat conduction equations, a piece of steel heated to 700°C would cool at a rate proportional to the difference between the plate and air temperature. This rate would produce a temperature plot that looks fairly logarithmic. Since the cooling medium is not simply the surrounding environment, but actually water droplets that sprayed onto the plate the plot of the plate temperature with respect to time is more complicated. A plot of the typical cooling shape can be seen in Figure 5.3-1. As the plate cools, the fluid in contact with the plate passes through the four boiling regimes. When the jet spray is first actuated film boiling is observed. As the plate cools, the fluid enters the transition boiling regime, ~300°C. Then, when critical heat flux is achieved the fluid enters the nucleate boiling regime, ~180°C. As the plate approaches the fluid temperature, the free convection regime is entered. During these four regimes the heat transferred from the plate to the cooling fluid is varies greatly and thus makes the determination of the heat transfer coefficient difficult because the phenomenon does not follow a uniform trend. This phenomenon will be discussed further in the following chapter.

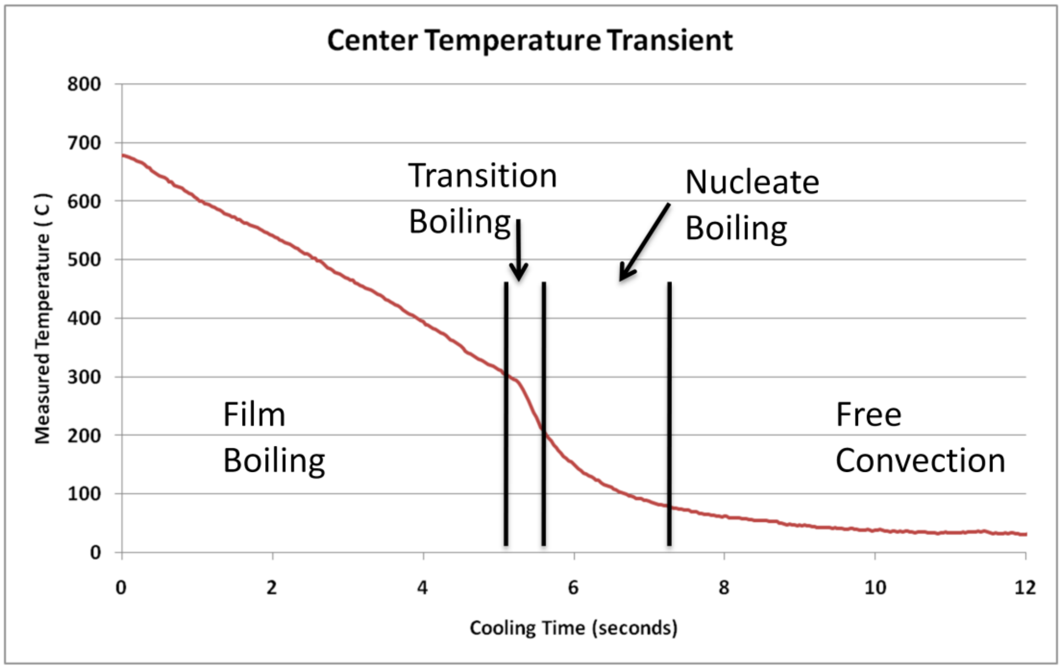


Figure 5.3-1: Plot of the cooling trend with the four boiling regimes labeled.

## 5.4 Error in the Measurement

As with all experimental studies, the desired outcome is to measure a characteristic of nature. The problem with experimentation is that the act of taking the measurement changes the phenomenon that is being measured. This problem means that the indication the gauge outputs is not the true value of the phenomenon, but is the measured value for the phenomenon. The measurement error is defined as the difference between the true value and the measured value. Thus in this document when the error is discussed it is not a reference to a broken instrument but the difference between the measurement and reality.

In determining the total measurement error for the system, the error induced by the sensor must be combined with the error induced by the DAQ. This was computed by taking the square root of the sum of the squares as directed by Knoll (44):

$$Error_{Combined} = \sqrt{\ Error_{DAQ}\ ^{2} + \ Error_{Instrument}\ ^{2}}\ , \tag{5.1}$$

where $\text{Error}_{\text{Instrument}}$ is the error in the instrument's measurement of the phenomenon and $\text{Error}_{\text{DAQ}}$ represents the error in measuring the instrument. The calculated combined measurement error can be found in Table 5.4-1. Since the infrared camera interfaces directly with the laptop, the error in the infrared measurement is simply the measurement of the device, ±2% of the reading.

Table 5.4-1: Table of the individual and combined measurement errors (34) (38) (40) (45).

| **Error Sources** | |
|---|---|
| Flowmeter | 0.500% |
| Pressure Transducers | 1.500% |
| K- type thermocouple | 1.1% |
| DAQ Current Measurement | 0.014% |
| DAQ Voltage Measurement | 0.0019% |
| DAQ Temperature | 1.0% |
| Combined Flow | **0.500%** |
| Combined Pressure | **1.500%** |
| Combined Temperature | **1.5%** |

## 5.5 Experimental Replication

A component of the *degree of goodness* is the ability to replicate a given set of data (43). In this context, replication is very different than repetition. Repetition is the process of doing experiment a second time, where replication is the process using the same input parameters to conduct an experiment multiple times producing similar results. Four experiments were conducted using the same parameters, where the infrared camera was in its high calibration two of the instances and medium calibration the other two instances. Data from the four tests were compared in two manners: a point comparison and area average comparison. It should be noted that the tests labeled High 1 and 2 were taken on different days and the average fluid temperature was 1.2 °C lower in High 1 than in High 2. The tests labeled Medium 1 and 2 were taken sequentially within a two hour period and the fluid temperature did not vary significantly between the tests. All other parameters for the test were within the error of the instrument and will be considered similar for all four tests.

The first method of comparison was done by plotting the cooling transient for the four tests as measured by an arbitrarily chosen camera pixel, 134x122. The data gathered using the high camera calibration sits relatively well one on top of the other. During the period where the measured temperature is above 350 °C the High 2 plot falls within the error of the High 1 measurement. Below the 350 °C point the two plots depart from each other. The reason for this departure is unknown, but could be a function of the differing coolant temperature. The plot Medium 2 sits within the Medium 1 plot's measurement error. These two plots display a high level of replication. This plot can be seen in Figure 5.5-1.

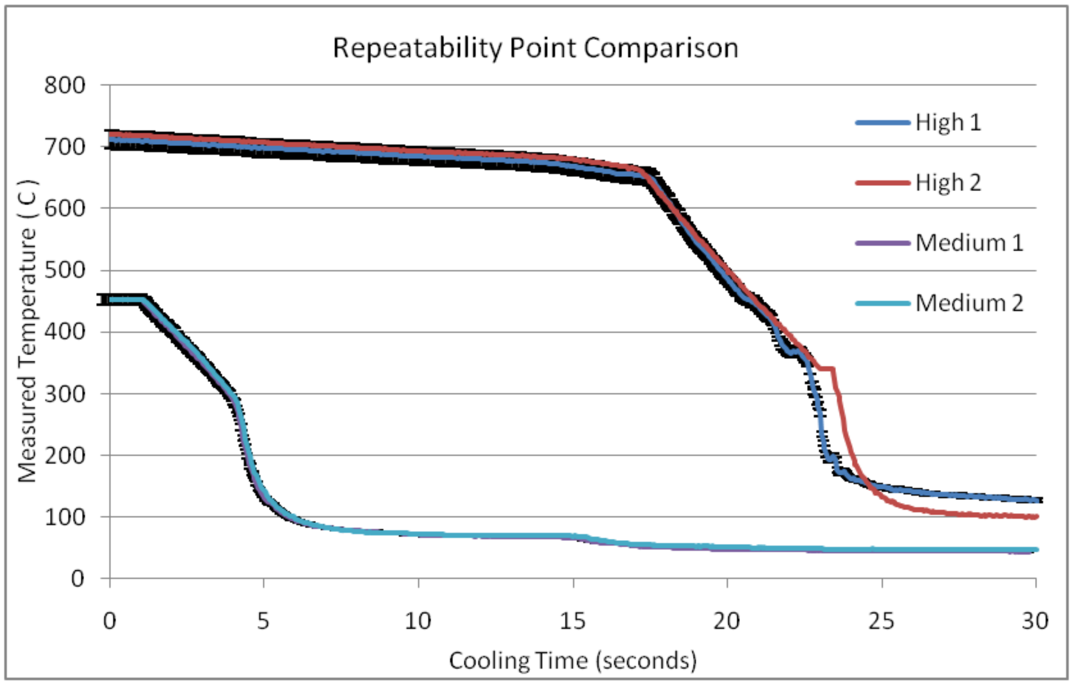


Figure 5.5-1: Point comparison of four tests with the same initial parameters, where the error bars for test High 2 and Medium2 were omitted for clarity.

The second method of comparison was done through plotting an average temperature over 404 pixels during the cooling transient. This arbitrarily circular area was chosen such that the sampling area and the nozzle spray pattern were concentric. This area physically represents a circle with an area of 0.6 square inches. Both the High 1 and 2 samples and Medium 1 and 2 samples display a high level of replication. Both sets fall within the error of the actual measurements. Again there is more variance between the High 1 and 2 than the Medium 1 and 2 comparisons. This is most likely due to the difference in coolant temperature of the high tests. The coolant temperature would change the heat transfer rate once the local temperature has dropped below the Leidenfrost point, this transition happens at about 18 seconds. The slope of the two High test are slightly different, and a steeper slope would correlate to higher heat transfer, which would happen with lower coolant temperatures. As can be seen in a plot of the area averaged comparison, Figure 5.5-2, the slope of the High 1 line between 18 and 23 seconds is slightly steeper.

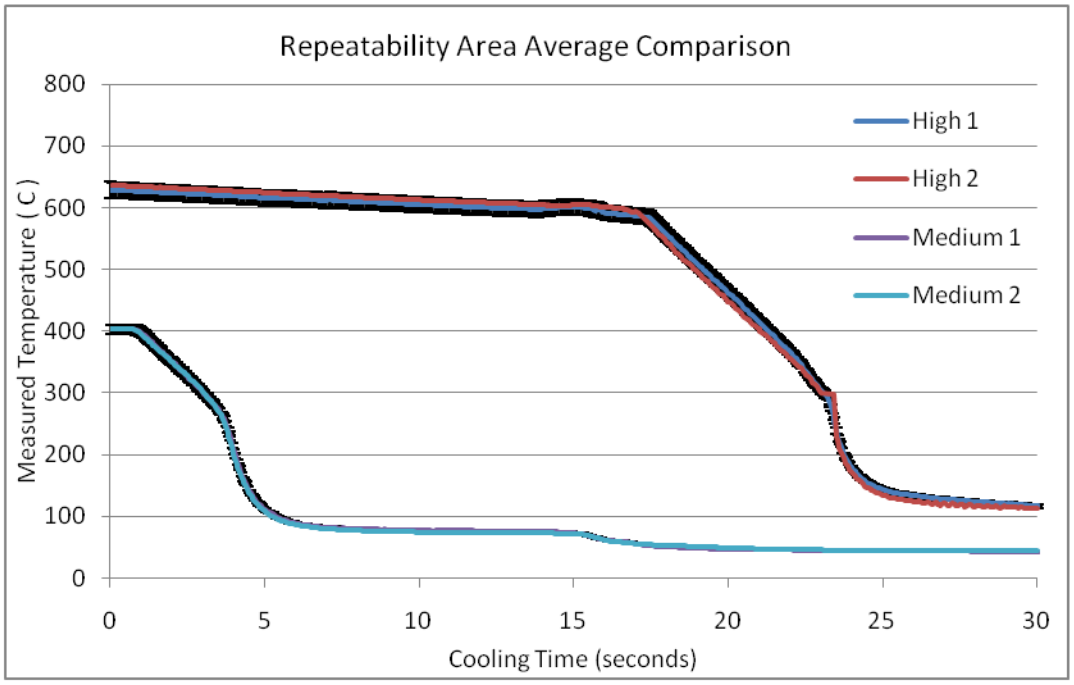


Figure 5.5-2: Area average comparison of four tests with the same initial parameters, where the error bars for test High 2 and Medium2 were omitted for clarity.

# 6 Data Analysis

The energy transferred from one object to another in the form of heat cannot be measured directly. Instead temperature measurements must be taken during a cooling transient and then the heat transferred can be estimated using lumped the energy balance equation:

$$h * A_S (T(t) - T_\infty) = \rho V c \frac{dT}{dt} , \qquad \text{Equation Section 6(6.1)}$$

where h is the heat transfer coefficient of the material, $A_S$ is the heat transfer surface area, T(t) is the temperature at a given time, ρ is the material density, V is the material volume, c is the specific heat of the material, and dT/dt is the change in temperature with respect to the change in time. Using the data presented in Chapter 5, this equation could be used to calculate h for about 33 million of the measurements, which it too much information to be useful.

To ascertain useful information from such a large database a method of extracting the data, determining the important trends and calculating h is needed. The following sections discuss the radial compression of the data, the calculation of h from the data, the relationship between the boiling phenomenon and the data, and the equations for h derived from the data.

## 6.1 Radial Data Compression and Added Uncertainty

A radial area average of the experimental data was taken over the thermal transient. This average was taken at the center of the nozzle impingement site over a circular area with a radius of 50 mm. The variance in the averaged data was less than the systematic error in the experimental data and thus will be ignored making the assumption that the area underneath the impingement area is at a uniform temperature during any given time step. Plots of the data can be seen in Figure 6.1-1 and Figure 6.1-2. The plots contain a total area average, labeled “All” and an average of the data along the edge of a circle that is a given distance from the impingement center, labeled “Avg1” through “Avg10”. These averages are taken on concentric circles with radii starting at 5 mm, for “Avg1”, and incrementing in 5 mm intervals to 50 mm at “Avg10”.

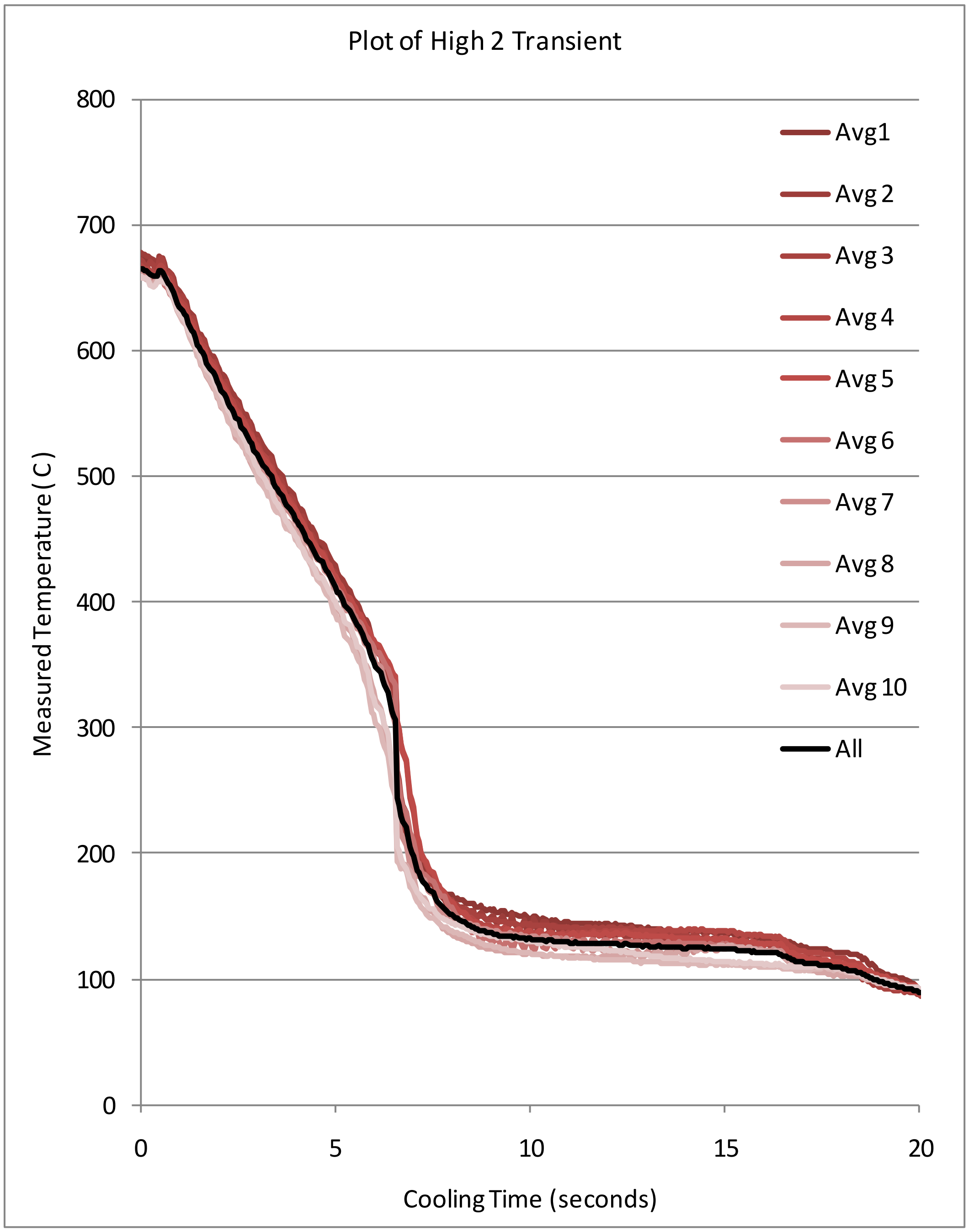


Figure 6.1-1: Plot of the condensed High 2 data.

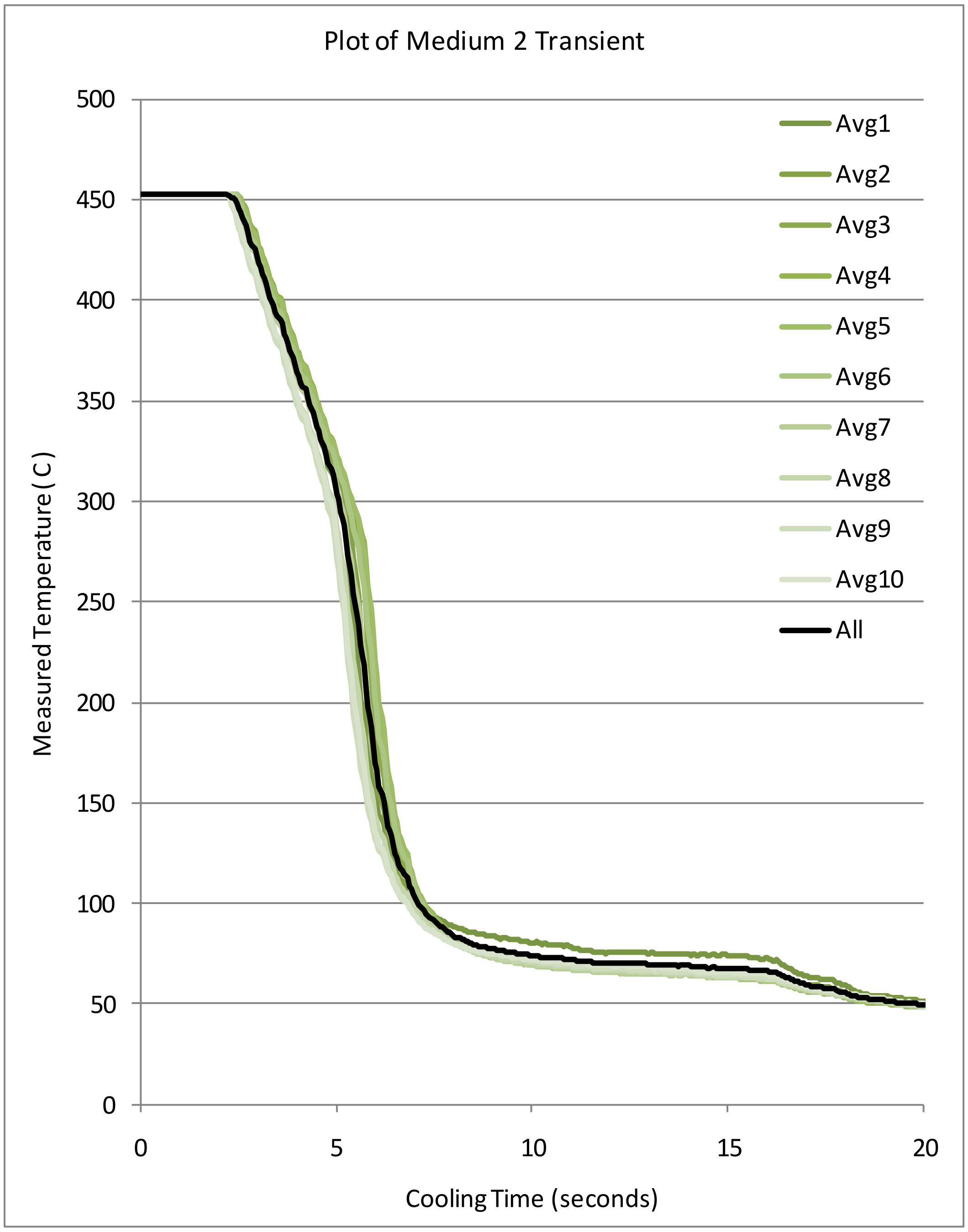


Figure 6.1-2: Plot of the condensed Medium 2 data.

## 6.2 Inverse Heat Transfer Calculation

Using the assumption that the heat transfer rate is constant during each time step, the differential equation (6.1) can be solved and used to calculate the time averaged heat transfer coefficient (TAHTC) at each average measurement with the formula:

$$\overline{h(t)} = -\ln\left(\frac{T(t) - T_\infty}{T(0) - T_\infty}\right) * \frac{\rho c L_C}{t} \quad , \tag{6.2}$$

where T(0) is the temperature at time zero, $L_C$ is the characteristic length and t is the current point in time. An oscillatory pattern was observed in the calculated data, as can be seen in Figure 6.2-1. Looking more closely at the calculated heat transfer, the first derivative with respect to time was calculated, a plot of the results can be seen in Figure 6.2-2, which displays a heavy oscillatory pattern about the general cooling trend. To reduce this oscillation a moving average was applied to the data. Comparisons were made using no average, and moving averages with three, five, and seven points centered on the current time step. The large oscillations were moderately removed from the data using the three averages independently of each other, but the loss in fidelity using the five or seven term averaging schemes did not merit the benefit of using them. The three term moving average was chosen to determine $T_{AVG}(t)$, the equation is:

$$T_{AVG}(t) = \frac{T(t-1) + T(t) + T(t+1)}{3} \quad , \tag{6.3}$$

where T(t) is the temperature of the current time step, T(t-1) is the temperature of the most recent time step, and T(t+1) is the temperature of the next time step. Plots of the first derivative of the calculated TAHTC using the various moving averages can be seen in Figure 6.2-2 through Figure 6.2-5. The resultant plot of the calculated TAHTC for the High 2 data and Medium 2 data can be found in Figure 6.2-6 and Figure 6.2-7, respectively.

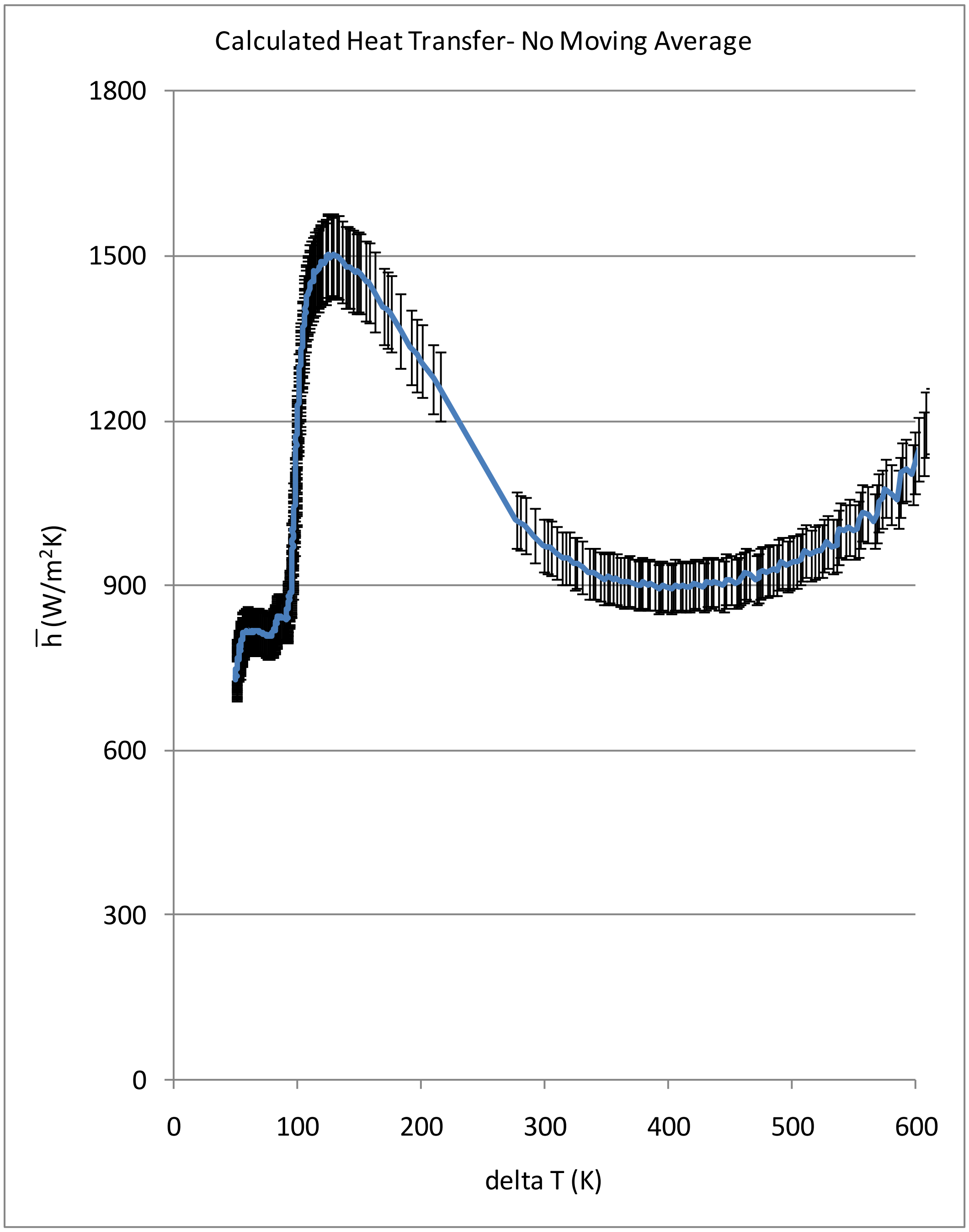


Figure 6.2-1: Plot of the calculated TAHTC, with the error bars included based on the error inherent in the measurement.

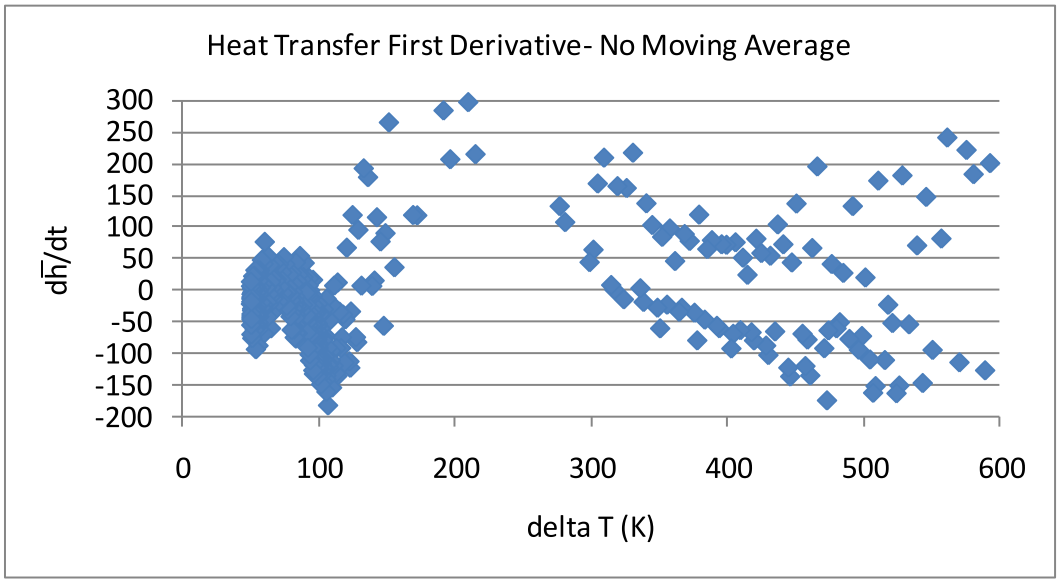


Figure 6.2-2: Plot of the first derivative of the TAHTC, with respect to time, using no moving average: error bars omitted for clarity.

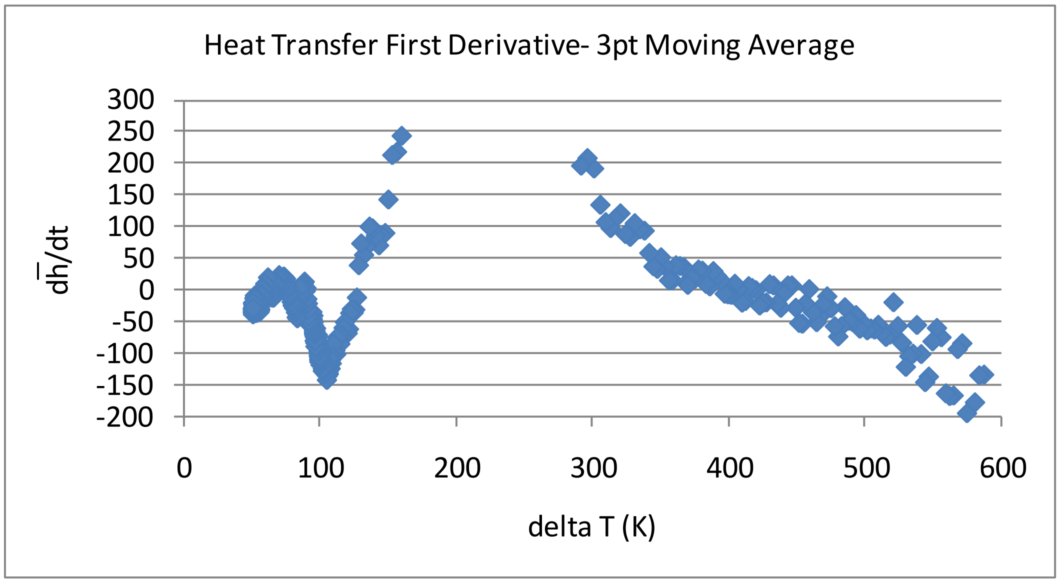


Figure 6.2-3: Plot of the first derivative of the heat transfer, with respect to time, using a three point moving average: error bars omitted for clarity.

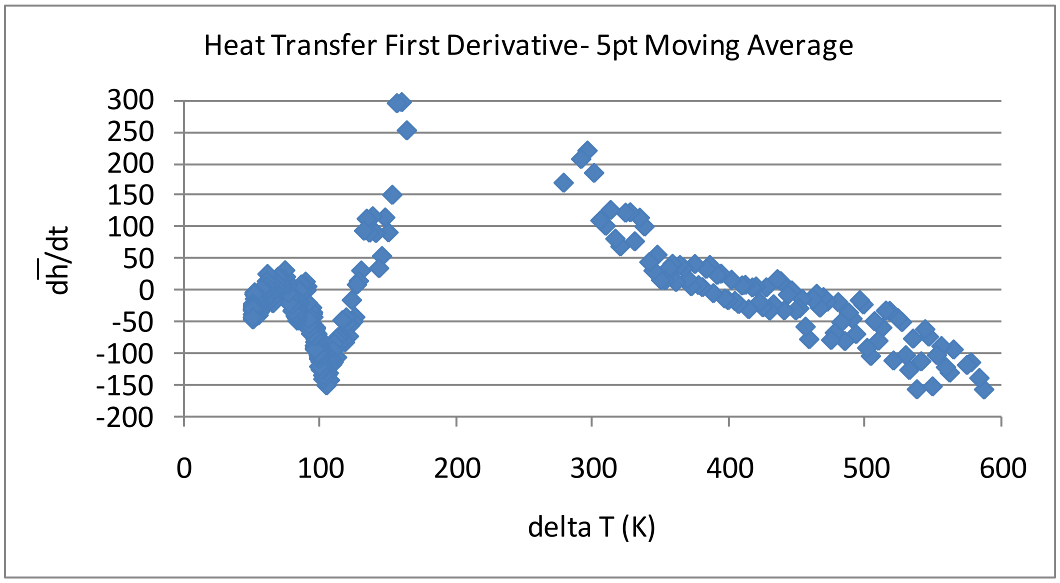


Figure 6.2-4: Plot of the first derivative of the TAHTC, with respect to time, using a five point moving average: error bars omitted for clarity.

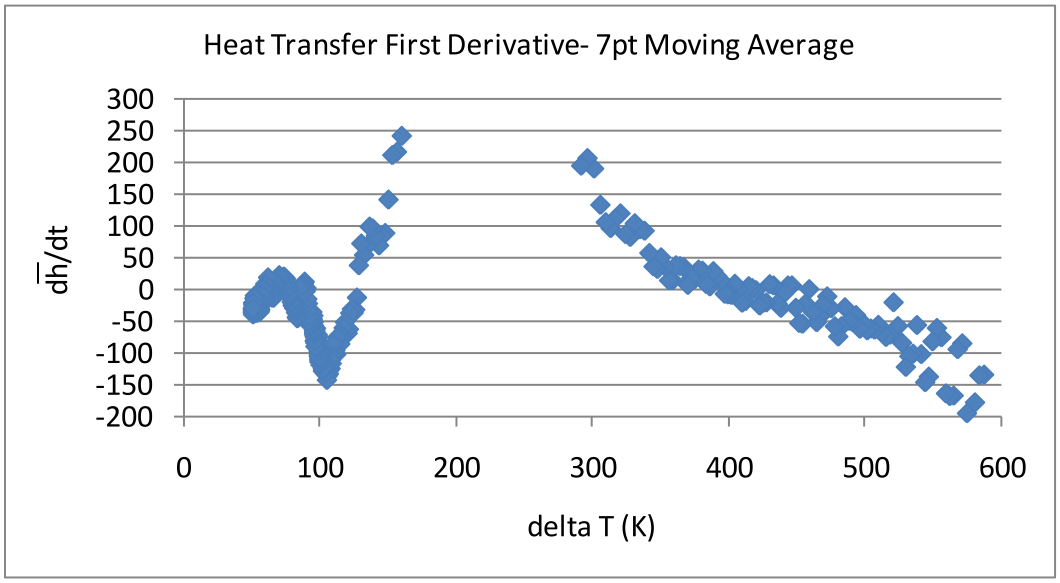


Figure 6.2-5: Plot of the first derivative of the TAHTC, with respect to time, using a seven point moving average: error bars omitted for clarity.

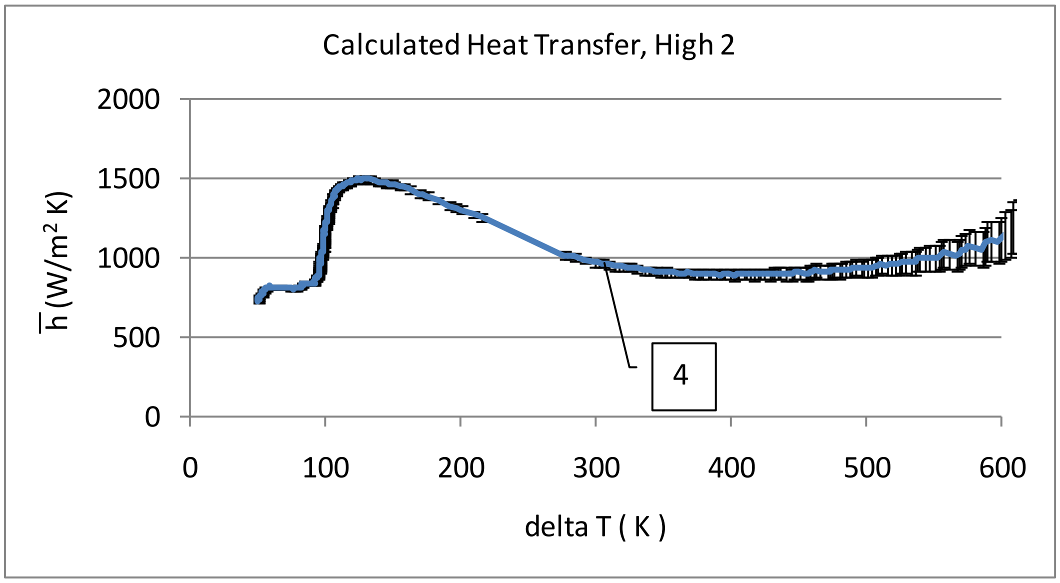


Figure 6.2-6: Plot of the calculated TAHTC using the high 2 data with a three point moving average. The Leidenfrost transition point is denoted by label 4.

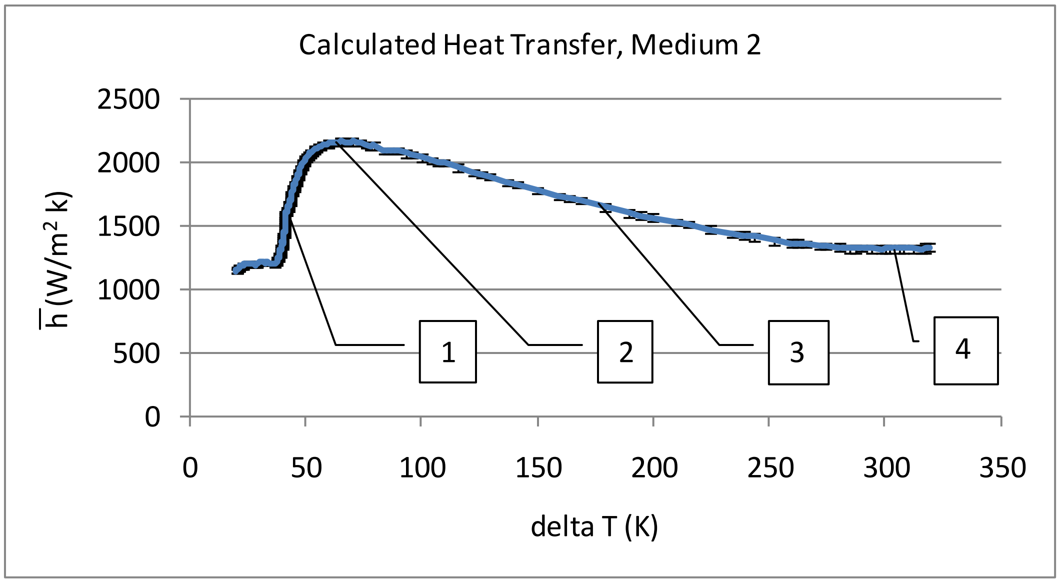


Figure 6.2-7: Plot of the calculated TAHTC using the medium 2 data with a three point moving average applied. The transition points of interest are labeled: 1 = Free Convection, 2 = Nucleate Transition, 3 = Critical Heat Flux, 4=Leidenfrost Point.

To determine the error in the calculated heat transfer coefficient, the input parameters to equation (6.2) were varied to determine the greatest effect of the uncertainty. The variance in the characteristic length, temperature measurement, and the time were evaluated to see which variable caused the greatest change in the resultant calculation. As expected, the quantity with the greatest error caused the greatest variation in the heat transfer calculation; this was the temperature measurement error. All of the figures plotted in this chapter are done so using the temperature measurement error to determine the size of the error bars. The effects of variance in characteristic length and time were negligible compared to the effects of the temperature variance and are thus omitted from further determinations.

## 6.3 Boiling curve HTC Relationships

The heat transferred from a heated piece of material to an impinging fluid is dependent on the different between the fluid and plate temperature. This difference in known as the superheat temperature commonly denoted as θ or ΔT. The shape of the plot in Figure 6.2-6 closely resembles the shape of the plot in Figure 6.3-1, where the transition points from one boiling regime to another are denoted. Using the discrete calculation of the TAHTC, developed in the previous section, the transition points A, B, C, and D, as denoted in Figure 6.3-1, will be determined. The temperature at these points will then be used to set the boundaries for a piecewise equation developed in the following chapter.

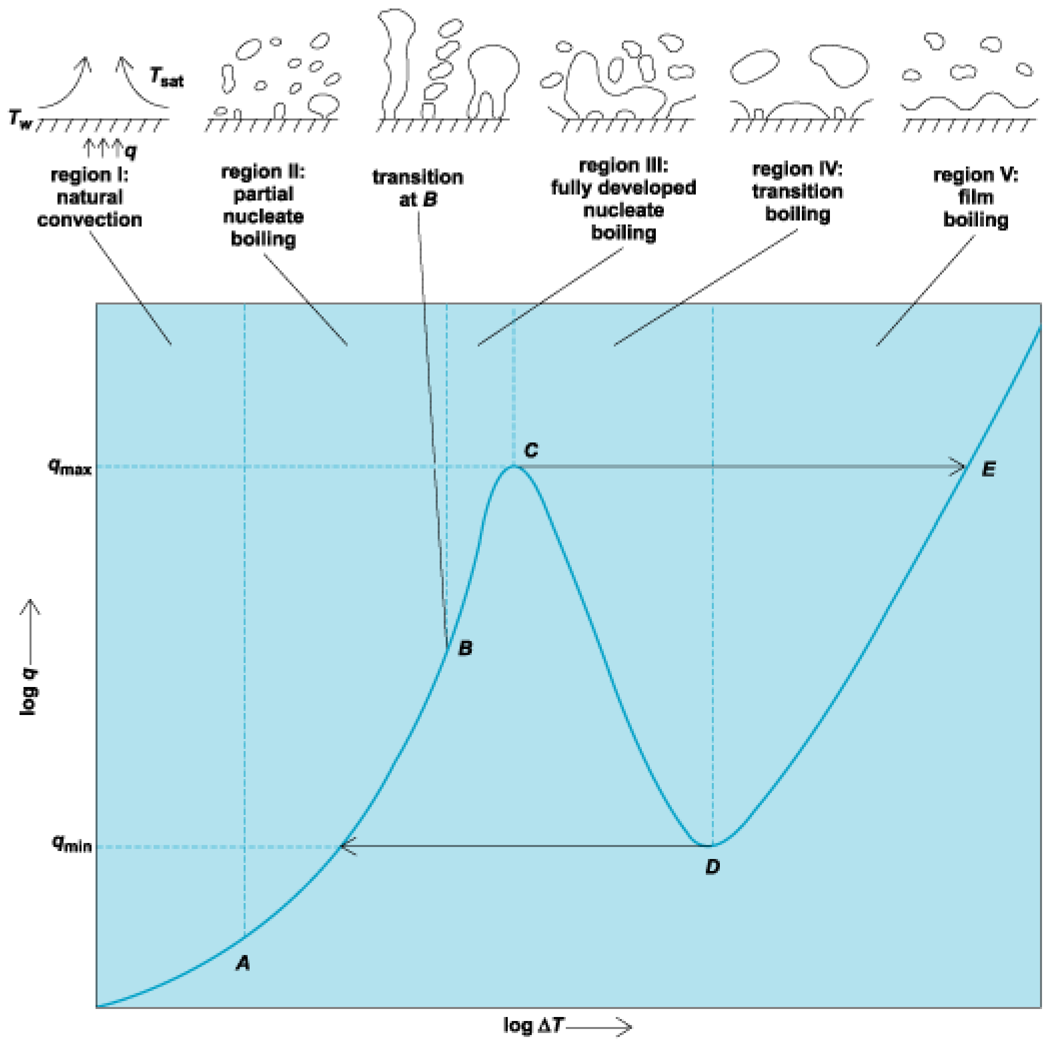

$T_w$
$T_{sat}$
$q$
region I: natural convection
region II: partial nucleate boiling
transition at B
region III: fully developed nucleate boiling
region IV: transition boiling
region V: film boiling
$q_{max}$
$q_{min}$
A
B
C
D
E
log q
log ΔT

Figure 6.3-1: Drawing relating the heat flux, super heat temperature and the boiling regimes (46).

### 6.3.1 Transition A Determination

Figure 6.3-2 displays a plot of the first derivative with respect to time of the discrete TAHTC calculation and the four transition points of interest are labeled. The transition between free convection and nucleate boiling, labeled A in Figure 6.3-2, will be chosen as the lower bound for the HTC. This decision is made because the test plate measurements become influenced by the surrounding material.

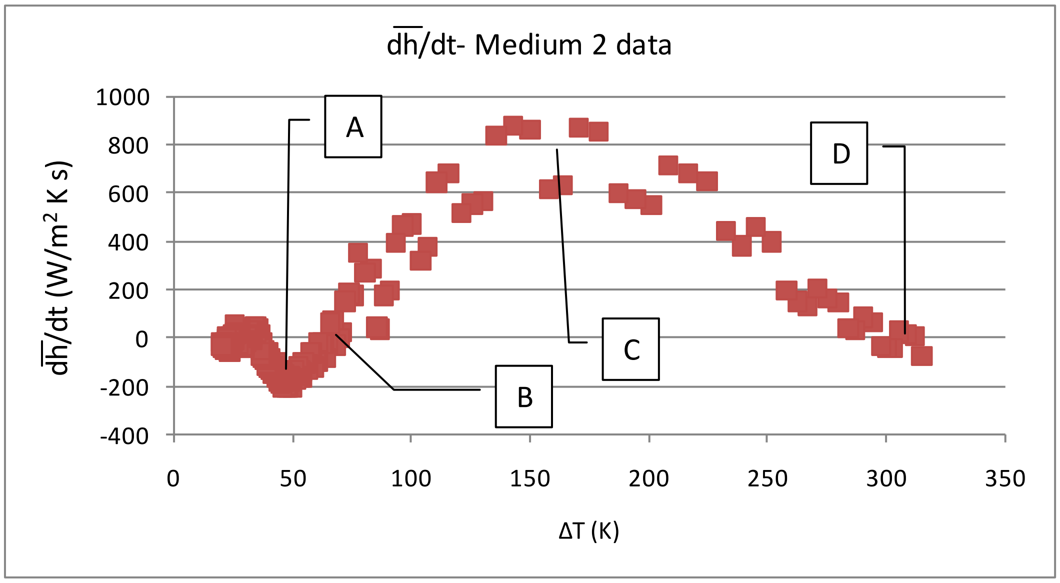


Figure 6.3-2: Plot of the Medium 2 data, with the first derivative with respect to time taken, error bars omitted for clarity.

The zero of first derivative of the function allows points B and D to be determined. Usually the zeros of the second derivative would be calculated to determine the location of points A and C, but the second derivative is too noisy and does not provide a basis to determine points A and C. Instead alternative methods are implemented to determine these points. A cubic regression was fit to the data between the super heat temperatures of 40K and 60K. The minimum of this line was calculated to be 48K.

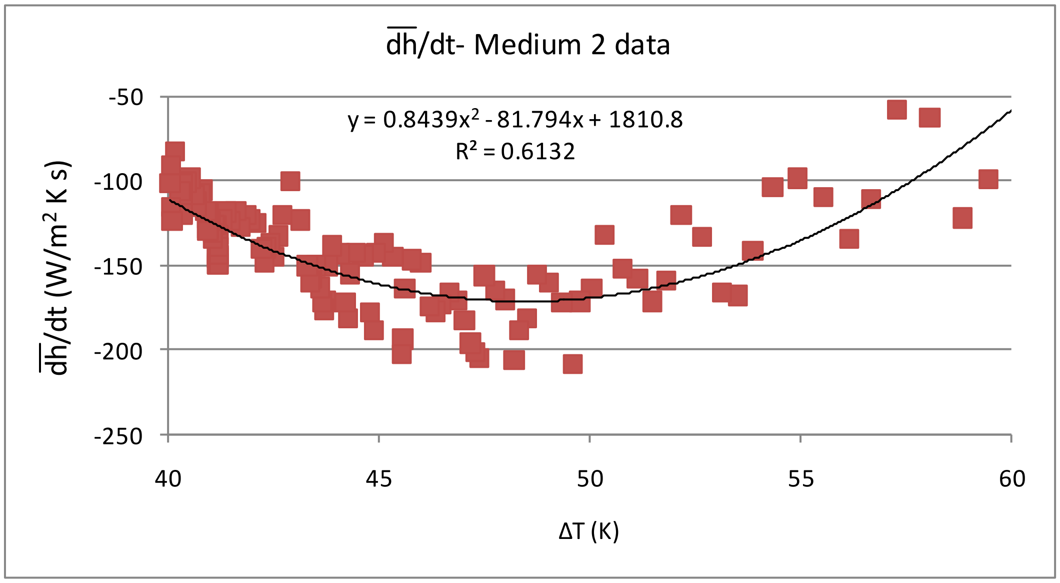


Figure 6.3-3: Plot for the discrete TAHTC determination with a cubic regression fit to the local data, which was used to determine transition point A.

## 6.3.2 Transition B Determination

To determine transition B, as labeled in Figure 6.3-2, a regression line is fit to the calculated data between the superheat temperatures of 50K and 80K. The regression line was solved for the x-axis intercept and a zero at 64K was calculated. A plot of the data and the corresponding regression line can be found in Figure 6.3-4.

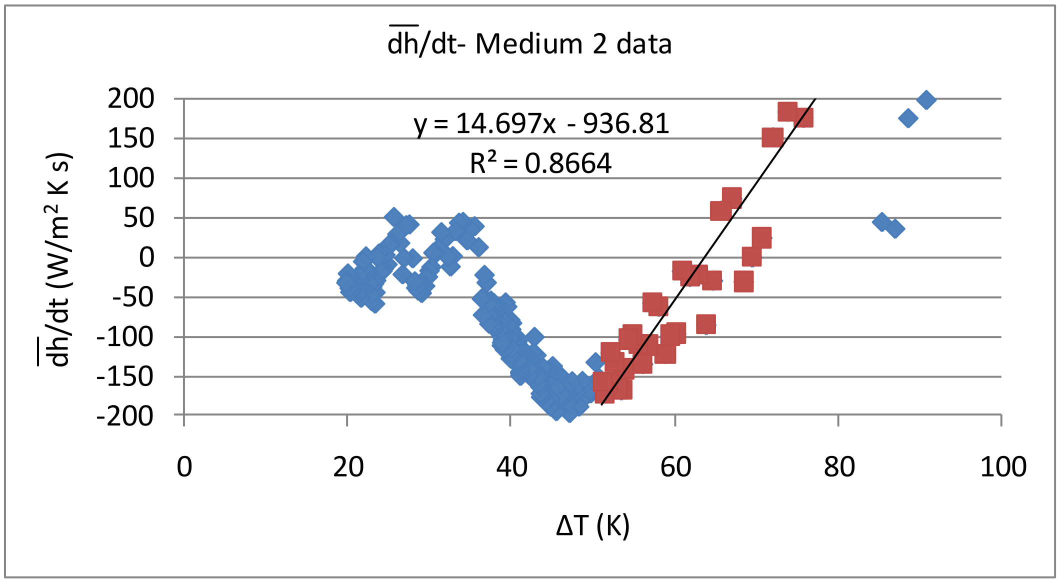


Figure 6.3-4: Plot for the discrete TAHTC determination with a regression line fit to the local data, which was used to determine transition point B.

## 6.3.3 Transition C Determination

To determine transition C, as labeled in Figure 6.3-2, the average temperature of the seven maximum points that were centered on the maximum in the plot were calculated to be a superheat temperature of 163K. The seven points that were used are denoted in Figure 6.3-5.

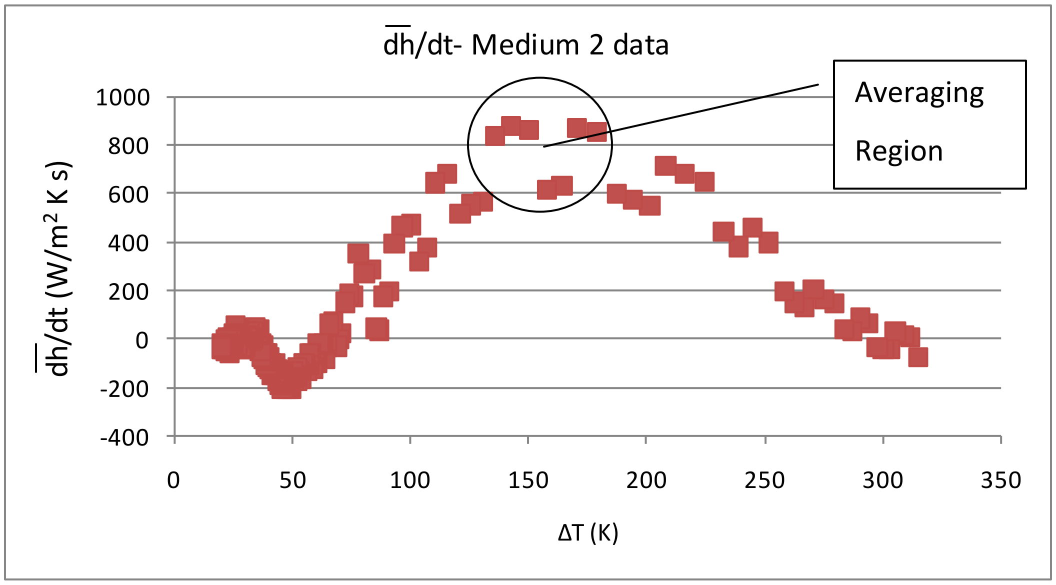


Figure 6.3-5: Plot for the discrete TAHTC determination with the seven maximum points denoted, which are used to determine transition point C.

### 6.3.4 Transition D Determination

To determine transition D, as labeled in Figure 6.3-2, a regression line is fit to the calculated data between the superheat temperatures of 250K and 320K. The regression line was solved for the x-axis intercept and a zero at 304K was calculated. A plot of the data and the corresponding regression line can be found in Figure 6.3-6.

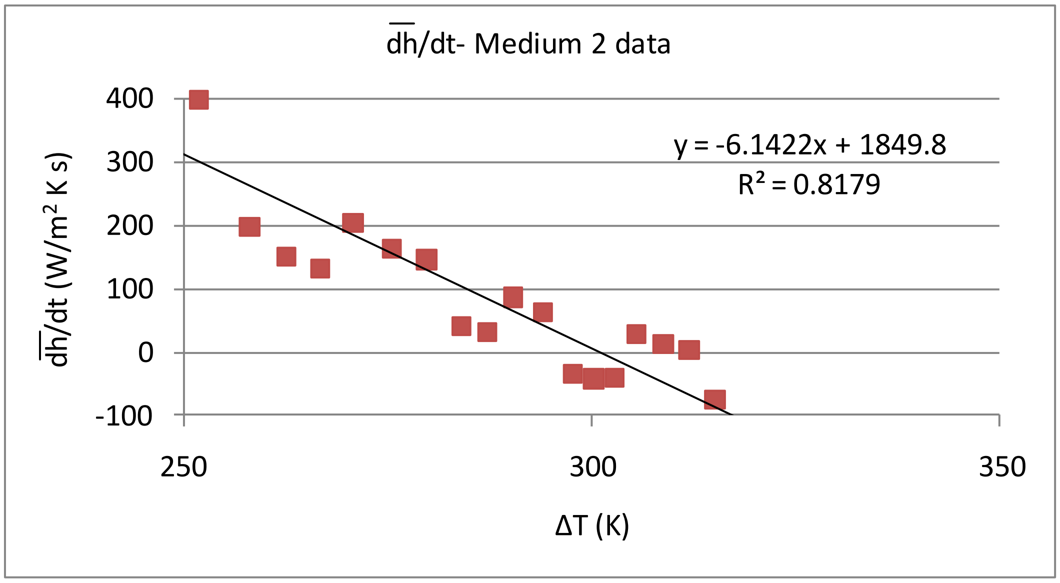


Figure 6.3-6: Plot for the discrete TAHTC determination with a regression line fit to the local data, which was used to determine transition point D.

## 6.4 HTC components

The method of determining the heat transfer used in section 6.2 provides a graphical method of visualizing the trend, but does not provide a basis to derive mathematical relationships. Using the assumption that heat transfer is constant was shown to be invalid for the entire transient, as seen by the non-linear shape of the plot in Figure 6.2-6. The assumption of constant heat transfer will be used for each time step. While it is known that the heat transferred is varying continuously, using a quantized heat transfer will allow the development of more general equations. The use of this assumption because the change between any two time steps is small compared to the overall trend.

A more accurate method to determine the heat transfer is needed. The heat transfer during the cooling process is a function of boiling phenomenon and temperature. Using a method adapted from Beck an equation for the heat transferred can be developed using ordinary least squares (47). This method was chosen because for two reasons. The first reason was that James Beck wrote one of the seminal works on parameter estimation and recommended this approach. The second reason is that by using a single pass method to determine the heat transfer, the numerical instability that is present in scenarios with small Fourier numbers are present, such as this one, will not be a problem because a non-iterative method was chosen.

Rearranging equation (6.1) provides the parameter of interest β:

$$\beta = \frac{hA}{\rho Vc} = \frac{h}{\rho L_C c}, \tag{6.4}$$

and equation (6.1) becomes:

$$\beta(T_\infty - T(t)) = \frac{dT}{dt}\,. \tag{6.5}$$

Since β is a function of temperature, it will be approximated by a function of superheat:

$$\beta = b_1 + b_2(T(t) - T_\infty)^n\,, \tag{6.6}$$

where $b_1$, $b_2$, and n are parameters to be determined using a least squares method. Using equation (6.4) β has units of $s^{-1}$, which means $b_1$ has units of $s^{-1}$ and $b_2$ has units of $(s*K)^{-1}$. Solving the resultant differential equation by separation of variables provides the following general solution:

$$b_1 * t * n = \ln[b_1 + b_2(T(t) - T_\infty)^n] - (n-1)\ln[1 - \frac{T(t)}{T_\infty}] - \ln[T_\infty - T(t)] + c\,. \tag{6.7}$$

To find a specific solution the initial condition,

$$\mathrm{T}\ \ 0\ \ = \mathrm{T}_0 \tag{6.8}$$

Is applied and two substitutions,

$$\theta = T(t) - T_\infty \tag{6.9}$$

$$\theta_0 = T_0 - T_\infty \tag{6.10}$$

are made. The resulting specific solution is:

$$b_1 * t * n = \ln[b_1 + b_2\theta^n] - (n-1)\ln[1 - \frac{T(t)}{T_\infty}] - \ln[-\theta] \\ - \ln[b_1 + b_2{\theta_0}^n] + (n-1)\ln[1 - \frac{T_0}{T_\infty}] + \ln[-\theta_0] \tag{6.11}$$

Applying laws of logarithms and algebraic manipulation provides a simpler form of the equation:

$$\frac{\theta}{\theta_0} = e^{-b_1 t}(\frac{b_1 + b_2\theta^n}{b_1 + b_2{\theta_0}^n})^{\frac{1}{n}} . \tag{6.12}$$

Using the experimental data for t, θ, and $\theta_0$ the parameters $b_1$, $b_2$, and n can be determined by minimizing the sum of the squares in the difference. The ordinary least squares formula is:

$$S = \sum_{i=1}^{J}\left[\left(e^{-b_1 t}(\frac{b_1 + b_2\theta^n}{b_1 + b_2{\theta_0}^n})^{\frac{1}{n}} - \frac{\theta}{\theta_0}\right)^2\right] \tag{6.13}$$

The first derivate of this equation with respect to $b_1$, $b_2$, and n are:

$$\frac{dS}{db_1} = 2\sum_{i=1}^{J}\left(e^{-b_1 t}(\frac{b_1 + b_2\theta^n}{b_1 + b_2{\theta_0}^n})^{\frac{1}{n}} - \frac{\theta}{\theta_0}\right) * \\ \left(\frac{e^{-b_1 t}}{n}\left(\frac{b_1 + b_2\theta^n}{b_1 + b_2{\theta_0}^n}\right)^{\frac{1}{n}-1}\left(\frac{1}{b_1 + b_2{\theta_0}^n} - \frac{b_1 + b_2\theta^n}{{b_1 + b_2{\theta_0}^n}^2}\right) - te^{-b_1 t}(\frac{b_1 + b_2\theta^n}{b_1 + b_2{\theta_0}^n})^{\frac{1}{n}}\right) \tag{6.14}$$

$$\frac{dS}{db_2} = 2\sum_{i=1}^{J}\left(e^{-b_1 t}\left(\frac{b_1+b_2\theta^n}{b_1+b_2{\theta_0}^n}\right)^{\frac{1}{n}} - \frac{\theta}{\theta_0}\right)* \left(\frac{e^{-b_1 t}}{n}\left(\frac{b_1+b_2\theta^n}{b_1+b_2{\theta_0}^n}\right)^{\frac{1}{n}-1}\left(\frac{\theta^n}{b_1+b_2{\theta_0}^n} - \frac{{\theta_0}^n \;\; b_1+b_2\theta^n}{b_1+b_2{\theta_0}^n \;\; ^2}\right)\right) \tag{6.15}$$

$$\frac{dS}{dn} = 2\sum_{i=1}^{J}\left(e^{-b_1 t}\left(\frac{b_1+b_2\theta^n}{b_1+b_2{\theta_0}^n}\right)^{\frac{1}{n}} - \frac{\theta}{\theta_0}\right)\left(e^{-b_1 t}\left(\frac{b_1+b_2\theta^n}{b_1+b_2{\theta_0}^n}\right)^{\frac{1}{n}}\right)* \left(\left(\frac{b_1+b_2{\theta_0}^n}{n \;\; b_1+b_2\theta^n}\left(\frac{b_2\theta^n \ln[\theta]}{b_1+b_2{\theta_0}^n} - \frac{b_2{\theta_0}^n \;\; b_1+b_2\theta^n \;\; \ln[\theta_0]}{b_1+b_2{\theta_0}^n \;\; ^2}\right)\right) - \frac{\ln\left[\frac{b_1+b_2\theta^n}{b_1+b_2{\theta_0}^n}\right]}{n^2}\right) \tag{6.16}$$

Setting each of the derivatives to zero to find the minimum, three equations with three unknowns are present. No simple method of solving these equations exists and thus a numerical method must be applied to solve for $b_1$, $b_2$, and n. As suggested by Beck the n coefficient was set to unity and then the $b_1$ and $b_2$ coefficients were solved for using a script written in Mathematica.

Once the $b_1$ and $b_2$ coefficients are determined, the actual coefficients for the heat transfer must be determined. At the beginning of this section β was defined as a parameter of characteristic length, specific heat, density, and heat transfer and then β was related to a line function of superheat. The heat transfer equation is estimated to be a function of superheat where:

$$h(\theta) = h_1 + h_2\theta + h_3\theta^2, \tag{6.17}$$

and the specific heat as also a function of superheat:

$$c(\theta) = c_1 + c_2\theta \tag{6.18}$$

where for the T∞ used in this study, $c_1$ = 435 and $c_2$ = 0.165. Setting equation (6.4) equal to (6.6) and substituting equations (6.17) and (6.18) for h and c, respectively, provides the following relationship linking β(θ) to h(θ):

$$b_1 + b_2\theta = \frac{h_1 + h_2\theta + h_3\theta^2}{\rho L_C \ \ c_1 + c_2\theta} . \tag{6.19}$$

Thus solving for $h_1$, $h_2$, and $h_3$:

$$h_1 = \rho L_C c_1 b_1 \tag{6.20}$$

$$h_2 = \rho L_C (c_1 b_2 + c_2 b_1) \tag{6.21}$$

$$h_3 = \rho L_C c_2 b_2 \tag{6.22}$$

To incorporate the effects of the boiling regime into the equation, a piecewise equation is implemented to encapsulate the heat transfer and boiling phenomenon relationship. This was be done by breaking the heat transfer range superheat, 48°C ≤ θ ≤ 600°C, into four regions. These regions were determined by examining the experimental data as discussed in section 6.3. The form of the resulting piecewise equation is below:

$$h(\theta) = \begin{cases} h_{1-1} + h_{1-2}\theta + h_{1-3}\theta^2 & 48 \le \theta < 64 \\ h_{2-1} + h_{2-2}\theta + h_{2-3}\theta^2 & 64 \le \theta < 163 \\ h_{3-1} + h_{3-2}\theta + h_{3-3}\theta^2 & 163 \le \theta < 304 \\ h_{4-1} + h_{4-2}\theta + h_{4-3}\theta^2 & 302 \le \theta < 600 \end{cases} \tag{6.23}$$

where coefficients labeled $h_{1-1}$ – $h_{4-1}$ are $h_1$ coefficients for each of the four boiling regions and similarly for the $h_{1-2}$ – $h_{4-2}$ and $h_{1-3}$ – $h_{4-3}$ coefficients. The numerical values for the coefficients can be found in Table 6.4-1.

Table 6.4-1: Table of the coefficients for the piecewise HTC as displayed in equation (6.23).

| | $h_1$ | $h_2$ | $h_3$ | n |
|---|---|---|---|---|
| $48 \leq \theta < 64$ | -1862 | 132.340 | -1.088 | 1.000 |
| $64 \leq \theta < 163$ | 2460 | -4.179 | -0.002 | 1.000 |
| $163 \leq \theta < 304$ | 3015 | -10.473 | 0.016 | 1.000 |
| $304 \leq \theta$ | 1896 | -4.863 | 0.0059 | 1.000 |

A plot of equation (6.23) using the coefficients from Table 6.4-1 can be found in Figure 6.4-1. While the discontinuities between region 1 and 2 and region 2 and 3 are small, there is a sizeable discontinuity between region 3 and 4. This is due to the source of the data. The coefficients derived for regions 1, 2, and 3 uses the Medium 2 data and the coefficients derived for region 4 uses the High 2 data. Currently differences in the data that would cause a discontinuity this large are not understood and a comparison of other data is needed to properly explain or remedy this problem. Another possibility for this discontinuity is that the change in heat transferred over a given time step is large at that transition and may not be completely captured by the equipment used to measure the transient. A comparison of the discrete calculation and the HTC are provided in Figure 6.4-2 through Figure 6.4-5. The only correlation that operates in scenarios with Re>1000, is plotted alongside the proposed HTC in Figure 6.4-6.

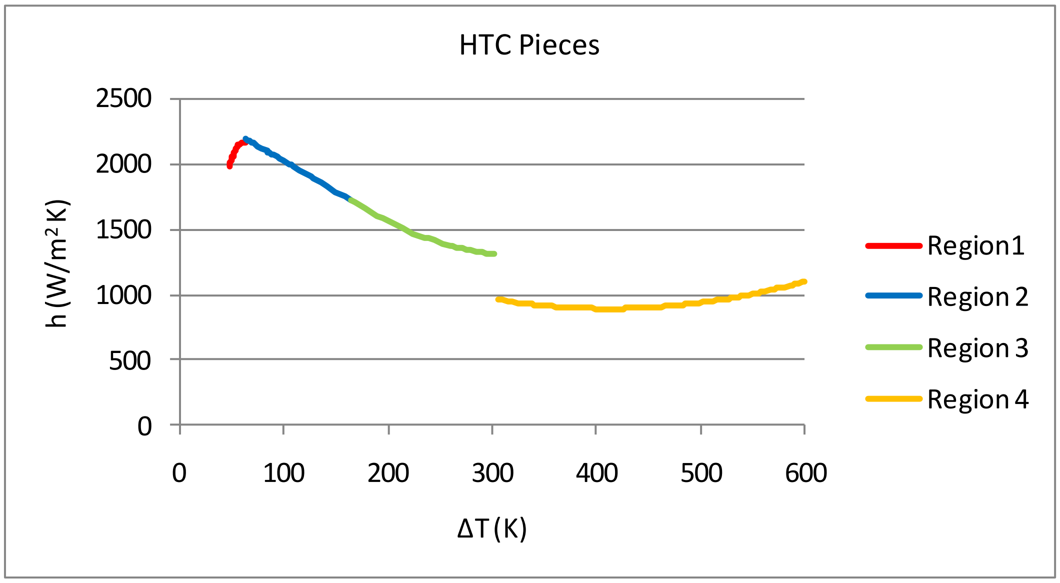


Figure 6.4-1: Plot of the four pieces of the HTC plotted together.

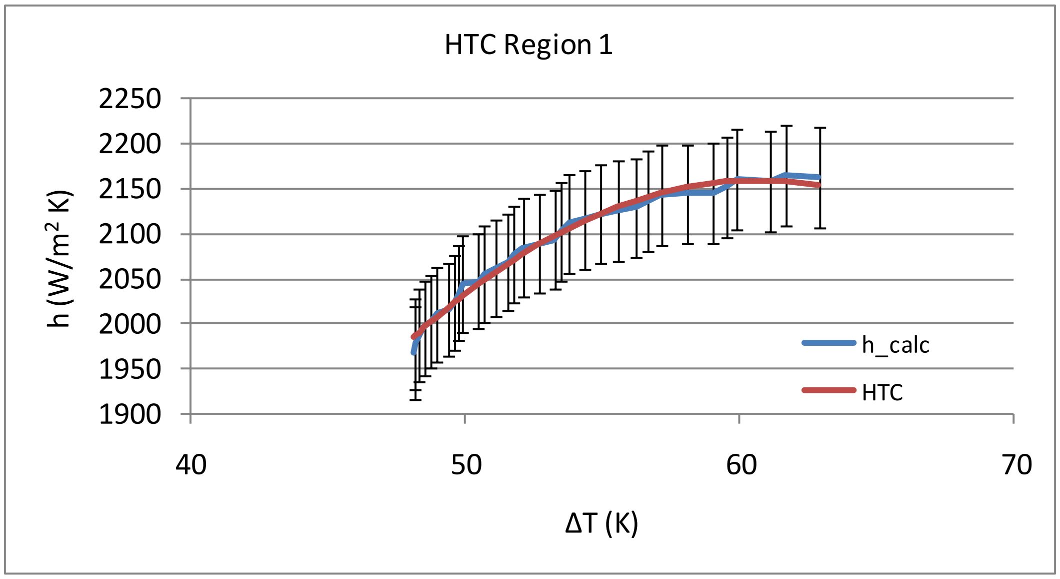


Figure 6.4-2: Comparison using the discrete heat transfer calculation with the HTC over region 1.

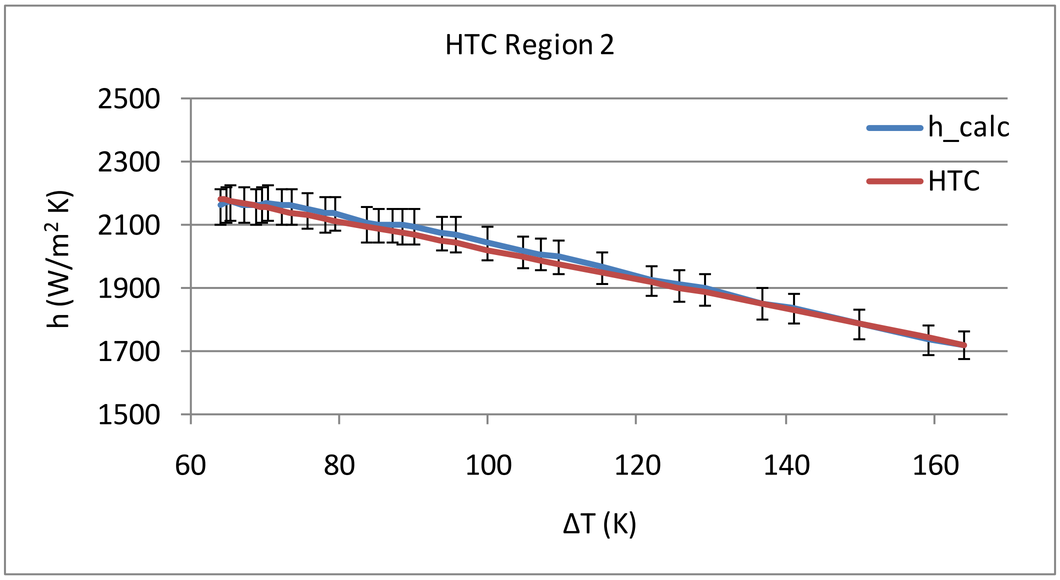


Figure 6.4-3: Comparison using the discrete heat transfer calculation with the HTC over region 2.

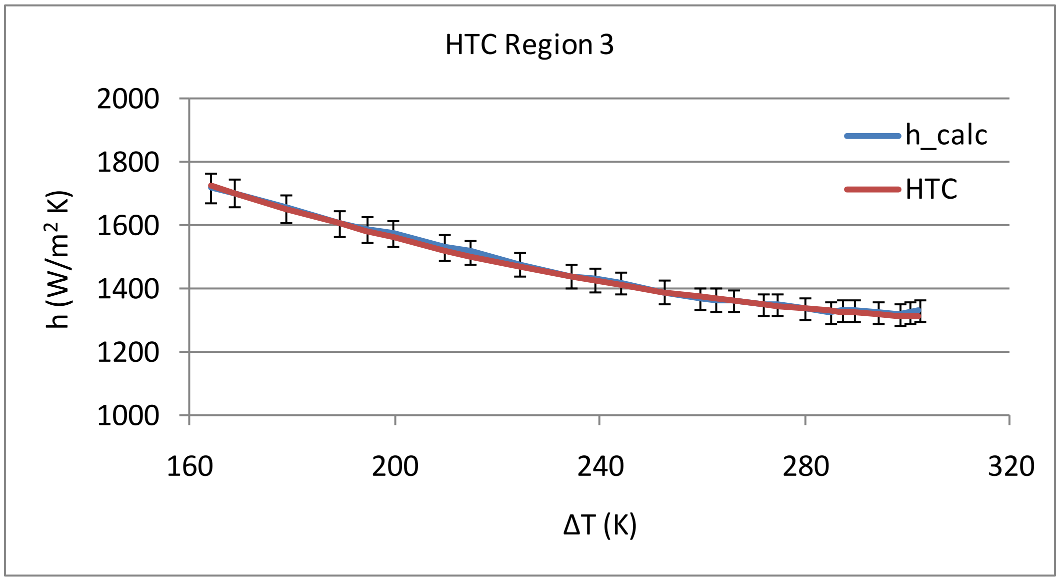


Figure 6.4-4: Comparison using the discrete heat transfer calculation with the HTC over region 3.

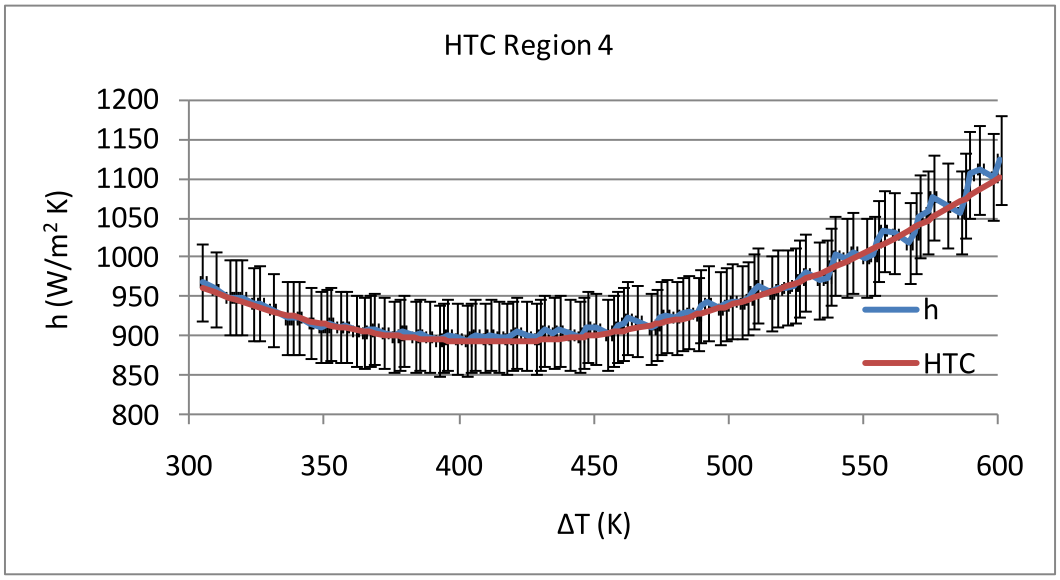


Figure 6.4-5: Comparison using the discrete heat transfer calculation with the HTC over region 4.

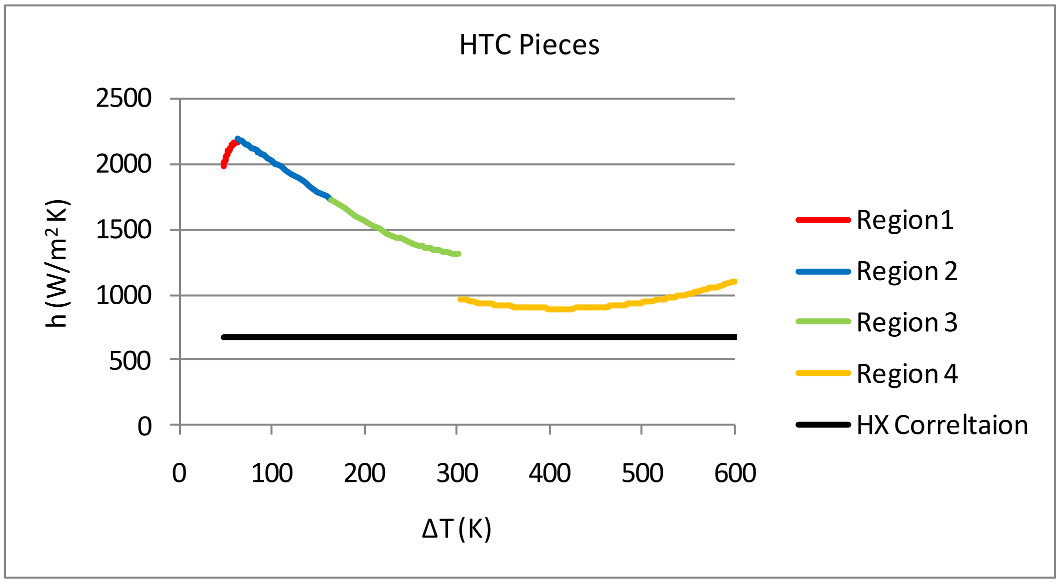


Figure 6.4-6: Plot of the developed HTC with the only correlation that was found to be applicable, a steam-water heat exchanger.

## 6.5 Non-dimensionalization of the HTC

In the HTC equation presented in previous chapter, the relationship was presented for a given flow rate, nozzle geometry, nozzle to plate relationship and fluid properties. Converting the given relationship into a non-dimensional form would allow the proposed HTC to be applicable to different fluids, characteristic lengths, hydraulic diameters, and material properties. The most common representation of a heat transfer coefficient is writing it in terms of the Nusselt number, which is a dimensionless term that describes the temperature gradient at a surface:

$$Nu = \frac{h * L_C}{k_f}, \tag{6.24}$$

where h is the heat transfer coefficient, $L_C$ is the characteristic length, and $k_f$ is the fluid's thermal conductivity. Using the definition of β as presented in equation(6.4), h can be represented as:

$$h = \beta * \rho * c_p * L_C \tag{6.25}$$

and substituted into (6.24) to produce a new relationship for Nu:

$$Nu = \frac{\beta * \rho * L_C^{\,2} * c_p}{k_f}, \tag{6.26}$$

This relationship can be more generalized by incorporating the Prandlt number which is the ratio of the momentum and thermal diffusivities:

$$\Pr = \frac{c_p \mu}{k_f} \tag{6.27}$$

and the Nusselt relationship becomes:

(6.28)

$$Nu = \frac{\beta * \rho * L_C{}^2 * \Pr}{\mu}, \tag{6.29}$$

where μ is the viscosity of the fluid. To further generalize the Nusselt relationship, the Reynolds number can be incorporated. The Reynolds number is the ratio of the inertia and viscous properties of the impinging fluid and is written as:

$$\mathrm{Re} = \frac{\rho * V_l * D_h}{\mu}, \tag{6.30}$$

where $V_l$ is the liquid velocity and $D_h$ is the hydraulic diameter of the spray nozzle. Incorporating equation (6.30) into equation(6.29) produces the following relationship:

$$Nu(\Pr, \mathrm{Re}, L_C, \beta, V_l, D_h) = \frac{\beta * L_C * \mathrm{Re} * \Pr}{V_l * D_h}. \tag{6.31}$$

The above equation provides a Nusselt number that is a function of β, characteristic length, Reynolds number, Prandlt number, fluid velocity, and nozzle hydraulic diameter and a plot of the relationship can be found in Figure 6.5-1. Values for β in each boiling regime can be found in Table 6.5-1. This relationship will only be valid for similar geometries and material that are thermally thin. For all the parameters, it is the user's responsibility to use the correct material properties in each section of this correlation. For example, the thermal conductivity of water is different than the thermal conductivity of steam and thus the Prandlt number will be different in each boiling regime.

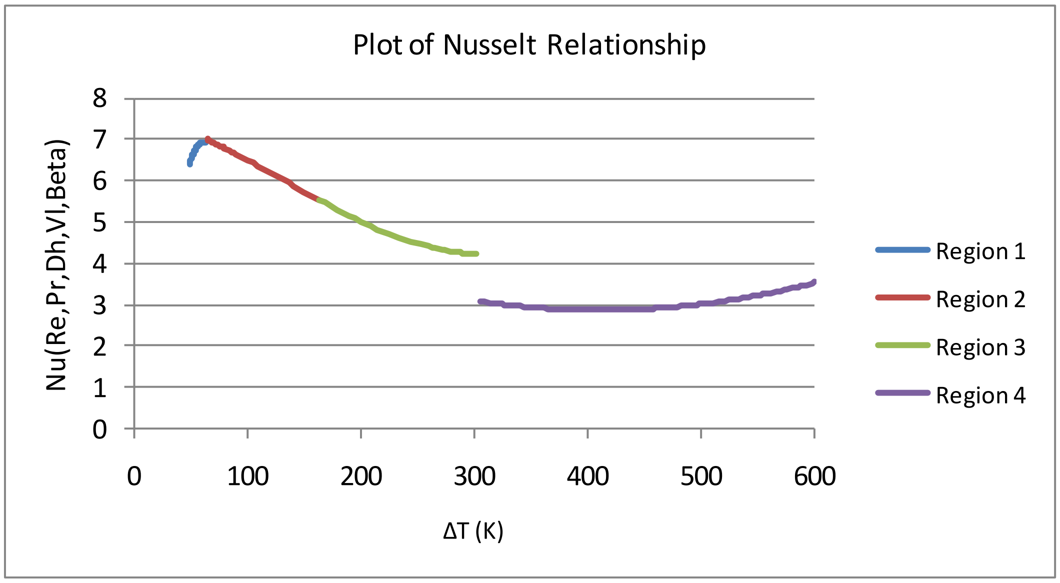


Figure 6.5-1: Plot of the equation (6.31) using the developed terms for β.

Table 6.5-1: Table of the coefficients for Beta.

| $B= b_1+ b_1*\theta$ | $b_1$ | $b_2$ |
|---|---|---|
| $48 \le \theta < 64$ | -0.291 | -0.449 |
| $64 \le \theta < 163$ | 0.385 | -0.001 |
| $163 \le \theta < 304$ | 0.4718 | 0.007 |
| $304 \le \theta$ | 0.297 | 0.002 |

# 7 Conclusions and Future Work

An experiment was designed, built, and operated at OSU to measure the temperature of a stainless steel plate, alloy 304, as it was cooled from 700°C to ambient temperature. A jet of water droplets was strayed onto the plate at a given flow rate and known distance from the plate. The plate temperature was measured by an infrared camera at a rate of 20Hz. The apparatus was fitted with thermocouples, flow meters, and pressure transducers that were used to determine the boundary conditions to the cooling transient. All instruments were calibrated and operated in a known and traceable manner allowing the generation of quality data. The purpose for collecting quality thermal transient data for a stainless steel plate was to develop a heat transfer correlation at high Reynolds numbers.Equation Chapter 7 Section 7

Using the data gathered from the experimental tests, the heat transferred from the test plate to the impinging water droplets was calculated. The nozzle was a known distance from the plate and a known fluid flow rate was used, which provided a Reynolds number of ~33,000. During the cooling process the fluid at the plate's surface passes through all four boiling regimes. Since the heat transferred from the plate to the fluid is a function of the boiling phenomenon, the resultant HTC is broken into four pieces in an effort to capture the physical changes in the system as the temperature changes. The current HTC was built of one test case, where the geometric and fluid constraints were set at the nozzle manufacturer's prototypical conditions.

While many HTCs exist, none have been found to describe the cooling of a plate with Reynolds numbers of greater than 10,000. The HTC developed in this work was built using a Reynolds number this outside of the typical correlation's bounds. This new HTC provides the ability to predict the heat transferred during a turbulent impingement conditions. The HTC was then non-dimensionalized to provide the following relationship:

$$Nu(\mathrm{Pr},\mathrm{Re},L_C,\beta,V_l,D_h)=\frac{\beta * L_C * \mathrm{Re} * \mathrm{Pr}}{V_l * D_h} \tag{7.1}$$

where β is a function of superheat and a parameter determined using the least squared fit to the empirical data. The components for β can be found in Table 6.5-1. This correlation is limited to similar geometries using materials that are thermally thin.

This work provides the foundation to extend the proposed HTC to a wider set of impingement cases. While the parameters used to develop the HTC represent the prototypical application for the given nozzle, incorporating the effects of varying nozzle position and orientation with respect to the test plate would increase HTC's window of applicability. The bounds between each piece of the HTC will vary with the roughness of the impingement target and work to incorporate variable boundaries into the HTC based on the surface conditions would provide more representative predictions as the boiling regime changes during the cooling transient when the surface finish is non-prototypical.

## *Bibliography*

1. Incropera, Frank P., DeWitt, David P. *Heat and Mass Transfer Fourth Edition.* New York : John Wiley & Sons, 2002. 0-471-38649-9.

2. Robb, Lesli. *Effect of Spray Height, Lead Angle and Offset Angle on Impact.* Wheaton, IL : Spraying Systems Co., 2004.

3. Hamed M. Al-Ahamdi, S.C. Yao. *Experimental Study on the Spray Cooling of High Temperature Metal Using Full COne Industrial Sprays.* Pittsburg, Pennsylvania : Carnegie Mellon University, 2006.

4. Tanner, Kristy. *Comparison of Impact, Velocity, Drop Size and Heat Flux to Redefine Nozzle Performance in the Caster.* Wheaton, IL : Spraying Systems Company, 2004.

5. Kasperski, Ken. *Spray Cooling Results of Air/Mist Spray Nozzles with Reduced Air Volumes.* Wheaton, IL : Spraying Systems Company, 2008.

6. Ferruh Erdogdu, Maria Ferrua, Smrendra K. Singh, Paul Singh. Air-impingement cooling of boiled eggs: Analysis of flow visualization and heat transfer. *Journal of Food Engineering.* 79, 2007, 920-928.

7. Sreekant Narumanchi, Andrey Troshko, Desikan Bharathan, Vahab Hassani. Numerical simulations of nucleate boiling in impinging jets: Applications in power electronics cooling. *International Journal of Heat and Mass Transfer.* 51, 2008, 1-12.

8. S.V.J Narumanchi, V. Hassani, D. Bharathan. *Modeling Single-Phase and Boiling Liquid Jet Impingement Cooling in Power Electronics.* s.l. : National Renewable Energy Laboratory, 2005. NREL/MP-540-38787.

9. C.H. Shen, C. Gau. Heat exchanger fabrication with arrays of sensors and heaters with its micro-scale impingement cooling process analysis and measurements. *Sensors and Actuators.* 114, 2004, 154-162.

10. Anja Royne, Christopher J. Dey. Design of a jet impingement cooling device for densly packed PV cells under high concentration. *Solar Energy.* 81, 2007, 1014-1024.

11. Martin, Holger. Heat and Mass Transfer between Impinging Gas Jets and Solid Surfaces. *Advances in Heat Transfer.* 13, 1977, 1-60.

12. V. Narayanan, J. Seyed-Yagoobi, R.H. Page. An experimental study of fluid mechanics and heat transfer in an impinging slot jet flow. *International Journal of Heat and Mass Transfer.* 47, 2004, 1872-1845.

13. Simona C. Arjocu, James A. Liburdy. Near Surface Characterization of an Impinging Elliptic Jet Array. *Transactions of ASME.* 1999, Vol. 121, June.

14. Vadiraj Katti, S.V. Prabhu. Experimental Study and theoretical analysis of local heat transfer distribution between smooth flat surface and impinging air jet from a circular straight pipe nozzle. *International Journal of Heat and Mass Transfer.* xxx, 2008, Vol. Article in Press as of 1/2008, xxx.

15. *Oscillatory thermal structures induced by unconfined slot jet impingement.* Vinod Narayanan, Vishal A Patil. 32, 2007, Experimental Thermal and Fluid Science, pp. 682-695.

16. *Identification of Dominant Heat Transfer Modes Associated with the Impingement of an Elliptical Jet Array.* S. C. Arjocu, J. A. Liburdy. May 2000, Journal of Heat Transfer, Vol. 122, pp. 240-247.

17. Matt Goodro, Jongmyung Park, Phil, Ligrani, Mike Fox, Hee-Koo Moon. Effects of Mach number and Reynolds number on jet array impingement heat transfer. *International Journal of Heat and Mass Transfer.* 50, 2007, 367-380.

18. Jung-Yang San, Chih-Hoa, Ming-Hong Shu. Impingement cooling of a confined circular air jet. *Int. J. Heat Mass Transfer.* 40, 1997, Vol. 6, 1355-1364.

19. Haydar Eren, Nevin Celik. Cooling of a heated flat plate by an obliquely impinging slot jet. *International Communications in Heat and Mass Transfer.* 33, 2006, 372-380.

20. Islam, Md. Ashraful. Jet impingement quenching phenomena for hot surfaces well above the limiting temperature for solid-liquid contact. *International Journal of Heat and Mass Tranfser.* In Press, as of 1/2008, 2008, xxx.

21. Vishal A. Patil, Vinod Narayanan. Spatially Resolved Heat Transfer rates in an Impinging circular microscale jet. *Microscale Thermophysical Engineering.* 9, 2005, 183-197.

22. Tsutsui, Yasunobu Fujita and Masayuki. Experimental Investigation of Falling Film Evaporation on Horizontal tues. *Heat Transfer.* 27, 1998, Vol. 8, 609-618.

23. Monde, Toshiaki Inoue and Masanori. Prediction of Pool Boiling heat Transfer Coefficient in Ammonia/Water Mixtures. *Heat Transfer.* 38, 2009, Vol. 2, 65-72.

24. Han-Taw Chen, Hung-Chin Wang. Estimation of heat-transfer characteristics on a fin under wet conditions. *International Journal of Heat and Mass Transfer.* xxx, 2008, xxx (Article in Press as of 1/2008).

25. John D. Bernardin, Clinton J. Stebbins and Issam Mudawar. Effects of surface roughness on water droplet impact history and heat transfer regimes. *International Journal of Heat and Mass Transfer.* 1997, Vol. 40, 1.

26. Engineers Toolbox. Overall Heat Transfer Coefficients for some common fluids and heat exchange surfaces. *Engineers Toolbox.* [Online] 2005. [Cited: 8 15, 2009.] http://www.engineeringtoolbox.com/overall-heat-transfer-coefficients-d_284.html.

27. Kenneth J. Baumeister, Robert C. Hendricks, Thomas D. Hamill. *Metastable Leidenfrost States.* Washington, D.C. : NASA, 1966. NASA TN D-3226.

28. Mudawar, John D. Bernardin and Issam. A Leidenfrost Point Model for Impinging Droplets and Sprays. *Journal of Heat Transfer.* 2004, Vol. 126, 272-278.

29. Myers, J.E. Sgheiza and J.E. Behavior of Nucleation Sites in Pool Boiling. *AIChE Journal.* 1985, Vol. 31, 10.

30. John D. Bernardin, Clinton J. Stebbins and Issam Mudawar. Mapping of impact and heat transfer regimes of water drops impinging on a polished surface. *International Journal of Heat and Mass Transfer.* 1997, Vol. 40, 2.

31. ASM International. *Thermal Properties of Metals.* Materials Park : ASM International, 2002. 0-87170-768-3.

32. Y.S. Touloukain, C.Y. Ho, Ed. *Thermophysical Properties of Selected Aerospace Materials, Part 2, Thermophysical Properties of Seven Materials.* s.l. : Perdue Research Foundation, 1977.

33. Box, G.E.P., Jenkins, G.M. *Time Series Analysis Forecasting and Control.* San Francisco : Holden-Day, Inc., 1970.

34. Omega Engineering. Transactions in Measurement and Control. *Technical Reference Section.* [Online] 2002. http://www.omega.com/.

35. Flir Systems. Flir A325 infrared Camera. *Flir Infrared Cameras.* [Online] Flir Thermography, 2009. [Cited: 3 11, 2009.] http://www.goinfrared.com /cameras/Automation/thermovision-a325-infraredcamera.

36. Ikeuchi USA. Standard Full Cone Spray Nozzles. *"The Mist Engineers", IKEUCHI USA, INC.* [Online] Ikeuchi USA, 2004. [Cited: 08 05, 2009.] http://www.ikeuchiusa.com/1_2_01.html.

37. Agilent Technologies. *Pratical Thermocouple Measurements.* s.l. : Agilent Technologies, 2008. Application Note 290.

38. Total Temperature Instrumentation, Inc. Watlow Adjustable Spring Thermocouple. *Instrumart.* [Online] 2009. [Cited: 7 11, 2009.] http://www.instrumart.com/Product.aspx?ProductID=22562.

39. OMEGA. General Purpose Pressure Transducer Solid State Pressure Transducer. *omega.com.* [Online] Omega, 2009. [Cited: 3 11, 2009.] http://www.omega.com/ppt/pptsc.asp?ref=PX209_PX219.

40. Emerson Process Management. R-Series Coroilis Meter. *Emerson Process Management.* [Online] 2009. [Cited: 3 11, 2009.] http://www2.emersonprocess.com/en-US/brands/micromotion/coriolis-flow-density-meters/R-series/Pages/index.aspx.

41. Edmund Optics. Commercial Quality 1/4 Wave First Surface Mirror. *Edmund Optics.* [Online] Edmund Optics, 2009. [Cited: 3 11, 2009.] http://www.edmundoptics.com/onlinecatalog/displayproduct.cfm?productID=1754&PageNum=4&StartRow=61.

42. Agilent. Agilent | 34970A Data Acquisition Switch Unit. *Agilent Technologies.* [Online] Agilent, 2009. [Cited: 3 11, 2009.] http://www.home.agilent.com/agilent/product.jspx?pn=34970a.

43. Hugh W. Coleman, W. Glenn Steele. *Experimentation and Uncertainty Analysis for Engineers.* New York : John Wiley & Sons, 1999. 0-471-12146-0.

44. Knoll, Glenn F. *Radiation and Detection Measurement.* Hoboken, NJ : John Wiley and Sons, 2000. 978-0-471-07338-3.

45. Agilent Technologies. *Agilent 34970A Data Acquisition/Switch Unit.* U.S.A. : Agilent Corp., 1999. 34970-90002.

46. Answers.com. Boiling: Definition, Synonyms. *Answers.com.* [Online] [Cited: 8 20, 2009.] http://content.answers.com/main/content/img/McGrawHill/Encyclopedia/images/CE757526FG0010.gif.

47. James V. Beck, Kenneth J. Arnold. *Parameter Estimation in Engineering Science.* New York : John Wiley & Sons, 1977. 0-471-06118-2.

*Appendix*

## Thermal View Source Code

```
//=================================================================
// Thermal Transient Viewer
// Version: 3.0
// Written by: Seth Cadell
// Written for the MHI project conducted at the OSU Nuclear
// Engineering Department.
// Purpose:       This program will load data files created by
// ThermoVision ExaminIR, a program used to support the
//    Flir A325 Infrared camera.  Once the data
//    is loaded, each time step's data will be
//    displayed in a dimensional grid.  The user will be provided
//    with controls that will allow them to specify the
//    files to load, the point in time that is being viewed and
//    the camera angle wrt the 3D plot.
// Input:   The user will fill in various boxes to load the data.
// Output:  The user inputs will be displayed on the canvas.
// Exit:    0-    Native Exit
//*****************************************************************
#include <stdlib.h> /* Must come first to avoid redef error */
#include <GL/glui.h>
#include <vector>
#include <iostream>
#include <iomanip>      // for setw and setprecision
#include <fstream>      // for file input
#include <sstream>      // for string manipulation
#include <string>
#include <math.h>

using namespace std;

/** Constants and enums **/
enum buttonTypes {LOAD_BUTTON = 0, EXIT_BUTTON, PLAY_BUTTON,
EXPORT_BUTTON, EXPORT_POSI, RESET_BUTTON};
enum frameTypes {OBJ_TEXTFIELD = 0, FF, LF, TS, DI, UC, LC, xmin,
xmax, ymin, ymax, CD, cTRACK};
enum cameraOps {CAMROTATE = 0, TRACK, DOLLY};
enum LeftButton {ROTATE, SCALE};
enum StyleTypes {PointStyle , LINES};

int WIN_WIDTH = 1000;
int WIN_HEIGHT = 800;

/** These are the live variables modified by the GUI ***/
const float ANGFACT = {1.};
const float SCLFACT = {0.005f};
const float MINSCALE = {0.005f};
const int LEFT = 4;
const int MIDDLE = 2;
const int RIGHT = 1;
int ActiveButton;
int LeftButton;
int Xmouse, Ymouse;
float Scale;
float Xrot, Yrot;

int main_window;                // window to display the isometric
                                 view of the data
int iWinW = WIN_WIDTH;          // updated window width
int iWinH = WIN_HEIGHT;         // updated window height
float fRb, fLb, fBb, fTb;       // global variables to hold
                                 the viewvolume information
int styleType = PointStyle;
int iFirstFrame = 1;
int iLastFrame = 100;
int iXmin = 0;                  // X minimum cropping variable
int iXmax = 239;                // X maximum cropping variable
int iYmin = 0;                  // Y minimum cropping variable
int iYmax = 319;                // Y maximum cropping variable
int iStartTime = 0;
int iTimeStep = 0;              // displayed time step variable
int iFrameRate = 100;           // variable to hold the timer delay
in milliseconds
int iFrameNumber = 0;           // currently loaded frame number
int iLoop = 0;                  // Loop variable for the play
function
int iPlay = 0;                  // variable to hold play info--
0=still, 1==play
int iLowerColor = 0;            // lower shading bound
int iUpperColor = 800;          // upper shading bound
float camRotMat[16];            // matrix to hold camera rotation
data
float camTrack[2] = {0.0, 0.0};     // camera track variables
float camDolly = 0.0;               // camera dolly variable

// export variables
float centerTrack[2]= {120.0, 180.0};     // center of the
                                            viewing area
float fHoriMult = 5.0;  // multiplier for the horizontal offset
float fVertMult = 5.0;  // multiplier for the vertical offset
int     iExportPosi = 0;        // holder for the export
                                 position of the data
float fCamDist = 0.8;               // variable to hold the
distance between the camera and the measured surface
```

```
int ExportPoints = 0;   // != 0 when the points should be drawn

float aiTemps[1000][240][320];
string sFile = "Null";
const int NUMS = 256;   // number of S texels
const int NUMT = 256;   // number of T texels
const int NUMP = 256;   // number of P texels

struct viewport{ int x, y;};
viewport VP[6];
float VPScale[3] = {1.0, 1.0, 1.0};

/** Globals **/
GLUI *glui;
GLUI_EditText *objFileNameTextField;
GLuint AxesList;  // will be assigned when list is compiled
GLuint DataList;
GLuint ExportList;
GLuint ScaleList;
int AxesRegen;    // used as a redraw flag when list items change
int DataRegen;
int ExportRegen;
int ScaleRegen;

// Function Definitions
void buttonCB( int );
void CameraManipCB( int );
void drawAxes( void );
void drawData( int );
void drawExports( int );
void drawScale( void );
int    ExportData( int );
void GetColor( float, float[] );
void initGLUI( void );
void InitLists( int );
void initScene( void );
void loadFile( void );
void myDisplay(   void );
void myMouse( int, int, int, int );
void myMotion( int, int );
void myGlutReshape(     int, int );
void playFunc( void );
void Reset( void );
void styleCB( int );
void textCB( int );

int main(int argc, char **argv)
//=================================================================
//    main() function
//    Purpose:    Handle the GUI creation and pass control the
//                GlutMain Loop
//    Input:
//    Output:
//*****************************************************************
{
      // setup glut
      glutInit(&argc, argv);
      glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

      glutInitWindowPosition( 0, 0 );
      glutInitWindowSize( WIN_WIDTH, WIN_HEIGHT);
      main_window = glutCreateWindow( "Thermal Transient Viewer
Main Window" );
      glutDisplayFunc( myDisplay );

      glutReshapeFunc(myGlutReshape);
      glutMouseFunc( myMouse );
      glutMotionFunc( myMotion );

      initScene();      // initialize the scene
      initGLUI();       // initialize the interface
      InitLists( 0 );   // initialize all display lists
      Reset();          // set all defaults

      glutMainLoop();
      return EXIT_SUCCESS;
}// end main

void buttonCB(int control)
//=================================================================
//    buttonCB() function
//    Purpose:    Handle the button actions
//    Input:      Function is called when any button is clicked
//    Output:     Exit_Button: Exit the program
//                Load_Button: Load the files to be displayed
//                Play_Button: Set variables to allow playback
//*****************************************************************
{
      float fTemp = 0.0;
      int iPower = 1;
      int dummy = 0;

      switch(control)
      {
      case EXIT_BUTTON: exit(0);          // exit program
            printf("Exit Button clicked \n");
            break;
      case EXPORT_BUTTON:     dummy = ExportData( 1 );
            if( dummy == 0 )  cout << "Data Successfully Exported
" << endl;
            else                    cout << "Error in Data
Export" << endl;
            break;

      case EXPORT_POSI:
            cout << " Export position change. " << endl;
            if(iExportPosi == 0){   // set pixel spacing for
                                     region #0
                  fHoriMult =  -5.0;
                  fVertMult = -5.0;
            }
            if(iExportPosi == 1){   // set pixel spacing for
                                     region #1
                  fHoriMult =   5.0;
                  fVertMult = -5.0;
            }
            if(iExportPosi == 2){   // set pixel spacing for
                                    region #2
                  fHoriMult =  -5.0;
                  fVertMult =  5.0;
            }
            if(iExportPosi == 3){   // set pixel spacing for
                                    region #3
                  fHoriMult =  5.0;
                  fVertMult = 5.0;
            }
            ExportRegen = 1;  // update the export display list
            break;

      case LOAD_BUTTON: loadFile();
            cout << "Load Button clicked.  Base file name: " <<
sFile << endl;
            DataRegen = 1;    // update the data display list
            break;

      case RESET_BUTTON:      Reset();
            break;

      case PLAY_BUTTON:
            if(iPlay == 0)
            {
                  iStartTime = glutGet( GLUT_ELAPSED_TIME );
                                    // set the time of playback
                  iPlay = 1;
                  cout << "Enabling Play Function\n";
            }
            else
            {
                  iPlay = 0;
                  cout << "Disabling Play Function\n";
            }
            break;
      default:          break;
      }
      glutSetWindow( main_window );
      glutPostRedisplay();
}// end buttonCB

void CameraManipCB(int id)
//================================================================
//    CameraManipCB() function
//    Purpose:    Alert the user of campera manipulation
//                parameter modifiaction
//    Input:      Function called when rotation, dolly, or
//                 camera translate used
//    Output:     prompt sent to terminal alerting user of change
//****************************************************************
{

      switch(id)
      {
      case CAMROTATE:   printf("Rotating the camera\n");
            break;
      case TRACK:       printf("Tracking the camera\n");
            break;
      case DOLLY:       printf("Dollying the camera\n");
            break;
      default:          break;
      }
      glutSetWindow( main_window );
      glutPostRedisplay();
}

void drawAxes( void )
//================================================================
//    drawAxes() function
//    Purpose: Display the Axes of the plot
//    Input:      Function is called everytime the canvas needs
//                 to be redrawn
//    Output:     The canvas is redrawn with a set of axes->
//                 X,Y,Z - R,G,B
//****************************************************************
{
      glColor3f(1.0,0.0,0.0);       // red line
      glLineWidth(4.0);             // increase line width
      glBegin(GL_LINES);
      glVertex3f(0.0, 0.0, 0.0);    // origin
      glVertex3f(0.0, 0.0, 340.0);  // end of x axis
      glEnd();
      glRasterPos3f(0.0, 0.0, 340.0);
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 88);

      glColor3f(0.0,1.0,0.0);       // green line
      glBegin(GL_LINES);
      glVertex3f(0.0, 0.0, 0.0);    // origin
      glVertex3f(260.0, 0.0, 0.0);  // end of y axis
      glEnd();
      glRasterPos3f(260.0, 0.0, 0.0);
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 89);

      glColor3f(0.0,0.0,1.0);       // blue line
      glBegin(GL_LINES);
      glVertex3f(0.0, 0.0, 0.0);    // origin
      glVertex3f(0.0, 800.0, 0.0);  // end of z axis
      glEnd();
      glRasterPos3f(0.0, 800.0, 0.0);
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 84);
}// end drawAxes()

void drawData( int style )
//=================================================================
//    drawData() function
//    Purpose:    Draw the temperature map, based on styleType
//    Input:      Function called from the display function.
//    Output:     The associated shapes are rendered
//*****************************************************************
{
      float rgb[3];                           // store current color
      float xpos, ypos, temp;                 // dummy variables
      glPushAttrib(GL_ALL_ATTRIB_BITS);
      if(style == LINES){
            glBegin( GL_LINES);
            for(int i=iXmin; i<iXmax; i++){
                  for(int j=iYmin; j<iYmax; j++){
                        xpos = float(i);
                        ypos = float(j);
                        temp = aiTemps[iFrameNumber][i][j];
                        GetColor( temp, rgb );
                        // set scale
                        glColor3f(rgb[0],rgb[1],rgb[2]);
                        glVertex3f(xpos,  0.0, ypos);
                        glVertex3f(xpos, temp, ypos);
                  }// end j loop
            }// end i loop
            glEnd();
      }// end line style
      else{
            glBegin( GL_POINTS);
            for(int i=iXmin; i<iXmax; i++){
                  for(int j=iYmin; j<iYmax; j++){
                        xpos = float(i);
                        ypos = float(j);
                        temp = aiTemps[iFrameNumber][i][j];
                        GetColor( temp, rgb );
                        glColor3f(rgb[0],rgb[1],rgb[2]);
      // set scale
                        glVertex3f(xpos, temp, ypos);
                  }// end j loop
            }// end i loop
            glEnd();
      }// end point style
      glPopAttrib();
}// end drawData

void drawExports( int style )
//=================================================================
//    drawExports() function
//    Purpose:    Provide the user a visual reference for the
//                 user to choose the export points
//    Input:      Function uses the globals- fHoriMult and
//                 fVertMult to determine the point spacing
//    Output:     The corresponding lines will be drawn in
//                 the three windows
//*****************************************************************
{
      float index;
      glColor3f(0.0, 1.0, 0.0);

      if(style == 1){
            glLineWidth(1.0);
            glBegin(GL_LINES);
            glVertex3f(centerTrack[1], 0.0, centerTrack[0]);
      // center point
            glVertex3f(centerTrack[1], 800.0, centerTrack[0]);
            for( int j=0; j<10; j++){
                  index = float( j );
                  glVertex3f(centerTrack[1]+index*fHoriMult*.707,
0.0, centerTrack[0]+index*fVertMult*.707);// for diagonals
                  glVertex3f(centerTrack[1]+index*fHoriMult*.707,
900.0, centerTrack[0]+index*fVertMult*.707);

// for points along the short side- vary centerTrack[1]
glVertex3f(centerTrack[1], 0.0, centerTrack[0]+index*fVertMult);
glVertex3f(centerTrack[1], 900.0,centerTrack[0]+index*fVertMult);

// for point along the long side- vary centerTrack[0]
glVertex3f(centerTrack[1]+index*fHoriMult, 0.0, centerTrack[0]);
glVertex3f(centerTrack[1]+index*fHoriMult, 900.0, enterTrack[0]);
            }
            glEnd();
      } // end if line style
      else{
            glPointSize(3.0);
            glBegin(GL_POINTS);
            glVertex3f(centerTrack[1], 800.0, centerTrack[0]);

            for( int j=0; j<10; j++){
                  index = float( j );
                  glVertex3f(centerTrack[1]+index*fHoriMult*.707,
800.0, centerTrack[0]+index*fVertMult*.707);// for diagonals
                  glVertex3f(centerTrack[1], 800.0,
centerTrack[0]+index*fVertMult);
                  glVertex3f(centerTrack[1]+index*fHoriMult,
800.0, centerTrack[0]);
            }
            glEnd();
      }// end point style

}// end drawExports

void drawScale( void )
//=================================================================
//    drawScale() function
//    Purpose:    will display a filled polygon with a black
//          boarder and hashes which will represent the color
//          scale for the cuboids
//    Input:            None
//    Output:           will display color scale on the canvas
//*****************************************************************
{
      float lowerbound, red1, green1, upperbound, range;
      lowerbound = float(iLowerColor);
      upperbound = float(iUpperColor);
      range = upperbound - lowerbound;
      red1 = range*0.33 + lowerbound;
      green1 = range * 0.66 + lowerbound;

      glPushAttrib(GL_ALL_ATTRIB_BITS);
      glPolygonMode(GL_FRONT, GL_FILL);
      // Scale for lower region
      glBegin(GL_QUADS);                              // draw
filled center- color varies according to scale
```

```
      glColor3f(0.0, 0.0, 0.0);
      glVertex3f( 0.0, lowerbound, 0.0);
      glVertex3f(20.0, lowerbound, 0.0);
      glVertex3f(20.0, 0.0, 0.0);
      glVertex3f( 0.0, 0.0, 0.0);
      glEnd();

      // Scale for red region
      glBegin(GL_QUADS);                             // draw
filled center- color varies according to scale
      glColor3f(  1.0, 0.0, 0.0);
      glVertex3f( 0.0, red1, 0.0);
      glVertex3f(20.0, red1, 0.0);
      glColor3f(  0.0, 0.0, 0.0);
      glVertex3f(20.0, lowerbound, 0.0);
      glVertex3f( 0.0, lowerbound, 0.0);
      glEnd();

      // Scale for green region
      glBegin(GL_QUADS);                             // draw
filled center- color varies according to scale
      glColor3f(  1.0, 1.0, 0.0);
      glVertex3f( 0.0, green1, 0.0);
      glVertex3f(20.0, green1, 0.0);
      glColor3f(  1.0, 0.0, 0.0);
      glVertex3f(20.0, red1, 0.0);
      glVertex3f( 0.0, red1, 0.0);
      glEnd();

      // Scale for blue region
      glBegin(GL_QUADS);                             // draw
filled center- color varies according to scale
      glColor3f(  1.0, 1.0, 1.0);
      glVertex3f( 0.0, upperbound, 0.0);
      glVertex3f(20.0, upperbound, 0.0);
      glColor3f(  1.0, 1.0, 0.0);
      glVertex3f(20.0, green1, 0.0);
      glVertex3f( 0.0, green1, 0.0);
      glEnd();

      // Scale for upper region
      glBegin(GL_QUADS);                             // draw
filled center- color varies according to scale
      glColor3f(  1.0, 1.0, 1.0);
      glVertex3f( 0.0, 800.0, 0.0);
      glVertex3f(20.0, 800.0, 0.0);
      glVertex3f(20.0, upperbound, 0.0);
      glVertex3f( 0.0, upperbound, 0.0);
      glEnd();
      glLineWidth(2.0);                          // set thin line
      glColor3f(0.0, 0.0, 0.0);
      // draw hashmarks and label
      for(int i=0; i<9; i++)
      {
            glBegin(GL_LINES);
            // draw hash mark
            glVertex3f(17.5, i*100.0, 0.0);
            glVertex3f(22.5, i*100.0, 0.0);
            glEnd();
            glRasterPos3f(25.0, i*100.0, 0.0);
      // set text position
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 48+i);
      // draw number
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 48);
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 48);
      }// end label and hash loop
      glPopAttrib();

}// end drawScale()

int ExportData( int x )
//=================================================================
//    ExportData Function
//    Purpose:
//    Input:
//    Output:
//*****************************************************************
{
      int hStep = int(fHoriMult);
      int vStep = int(fVertMult);
      int NumPoints = 11;
      int iFrames = iLastFrame - iFirstFrame;
      string sExtension = "export.csv";
      string sFinal = "Null";
      sFinal = sFile + sExtension;


      ofstream outData;       // declare file output class
      outData.open(sFinal.c_str());
      outData << fixed << showpoint;
      outData << setprecision(2);

      if(!outData) return 1;
      // Create Header for file
      outData << "Export Data from the MHI test apparatus at OSU"
<< endl;
      outData << "File name " << sFinal << endl;
      outData << "Each entry below comes from the raw data output
by the Flir ExaminIR software." << endl << endl;
      outData << "FrameNum   Time    ";
      for( int j=0; j<NumPoints; j++)     outData << "Diag"<<
setw(2)<< j << "   ";
      for( int j=0; j<NumPoints; j++)     outData << "Vert"<<
setw(2)<< j << "   ";
      for( int j=0; j<NumPoints; j++)     outData << "Hori"<<
setw(2)<< j << "   ";
      outData << endl;
      outData << "Pixel     " << int(centerTrack[1]) << ',' <<
int(centerTrack[0]) << "  ";
      for( int j=0; j<NumPoints; j++)
            outData << setw(3)<< int(centerTrack[1]+j*hStep*.707)
<<','<< setw(3) <<int(centerTrack[0]+j*vStep*.707)<<"  ";
      for( int j=0; j<NumPoints; j++)
            outData << setw(3) << int(centerTrack[1]) << ',' <<
setw(3) << int(centerTrack[0])+j*vStep << "  ";
      for( int j=0; j<NumPoints; j++)
            outData << setw(3) << int(centerTrack[1])+j*hStep
<<','<< setw(3) << int(centerTrack[0])<< "  ";
      outData << endl << endl << endl;


      outData << "FrameNum   Time    ";
      for( int j=0; j<NumPoints; j++)     outData << "Diag"<<
setw(2)<< j << "   ";
      for( int j=0; j<NumPoints; j++)     outData << "Vert"<<
setw(2)<< j << "   ";
      for( int j=0; j<NumPoints; j++)     outData << "Hori"<<
setw(2)<< j << "   ";
      outData << endl;

      int d1 = 0;
      int d2 = 0;
      // Output Actual Data
      for( int i=0; i<iFrames; i++){
            // for frame number
            outData << setw(8) << i + iFirstFrame << ' ';
            // for points along diagonal
            outData << setw(8) <<
aiTemps[i][int(centerTrack[1])][int(centerTrack[0])] << ' ';
            for( int j=0; j<NumPoints; j++){
                  outData << setw(8) <<
aiTemps[i][int(centerTrack[1]+j*hStep*.707)][int(centerTrack[0]+j
*vStep*.707)] << ' ';
            }
      // for points along the short side- vary centerTrack[1]
            for( int j=0; j<NumPoints; j++){
                  outData << setw(8) <<
aiTemps[i][int(centerTrack[1])][int(centerTrack[0])+j*vStep] << '
';
            }
            // for point along the long side- vary centerTrack[0]
            for( int j=0; j<NumPoints; j++){
                  outData << setw(8) <<
aiTemps[i][int(centerTrack[1])+j*hStep][int(centerTrack[0])] << '
';
            }
            outData << endl;
      }

      outData.close();
      return 0;
} // end

void GetColor( float temp, float RGB[] )
//=================================================================
//    GetColor Function
//    Purpose:    Determine the color of the associated point
//    Input:      Function is passed a floating point that
//                 represents a temp
//    Output:     The corresponding RGB value of the color
//                 is returned
//*****************************************************************
{
      float LowTemp = float(iLowerColor);
      float HighTemp = float(iUpperColor);
      float low, mid, range;
      range = HighTemp - LowTemp;
      low = 0.33*range + LowTemp;
      mid = 0.66*range + LowTemp;

      if(temp < LowTemp){
            RGB[0] = 0.0; RGB[1] = 0.0; RGB[2] = 0.0;
      }// end low temp test
      else{
            if(temp < low){
                  RGB[0] = temp/low; RGB[1] = 0.0; RGB[2] = 0.0;
            }// end low range temp test
            else{
                  if(temp < mid){
                        RGB[0] = 1.0; RGB[1] = (temp - low)/(mid
- low); RGB[2] = 0.0;
                  }// end mid range temp test
                  else{
                        if( temp < HighTemp ){
                              RGB[0] = 1.0; RGB[1] = 1.0; RGB[2] 
= (temp - mid)/(HighTemp - mid);
                        }// end high range temp test
                        else{
                              RGB[0] = 1.0; RGB[1] = 1.0; RGB[2] 
= 1.0;
                        }// end over temp test
                  }// end mid range tests
            }// end low range else
      }// end outside else
} // end GetColor

void initGLUI()
//=================================================================
//    initGLUI Function
//    Purpose:    Create the Gui controls and indicators
//    Input:      Control is passed from main
//    Output:     Control is returned to Main after the GUI is
//                 drawn
//*****************************************************************
{// create and place control window
      glui = GLUI_Master.create_glui( "Viewer Controls", 0, 
WIN_WIDTH +50, 0 );

      GLUI_Panel *FilePanel = glui->add_panel("Files");
      objFileNameTextField = 
            glui->add_edittext_to_panel(FilePanel, "Base 
Filename:",GLUI_EDITTEXT_TEXT,0,OBJ_TEXTFIELD,textCB);
      objFileNameTextField->set_text("d:\\116_exports\\MHI-TEST-
116");
      objFileNameTextField->set_w(300);
      objFileNameTextField->set_alignment(GLUI_ALIGN_RIGHT);

      GLUI_Spinner *FirstFrame;
      FirstFrame= glui->add_spinner_to_panel(FilePanel, "First 
Frame Number", 2, &iFirstFrame, FF, textCB);
      FirstFrame->set_int_limits(000, 20000);
      FirstFrame->set_alignment(GLUI_ALIGN_RIGHT);
      FirstFrame->set_w(100);
      GLUI_Spinner *LastFrame = glui-
>add_spinner_to_panel(FilePanel, "Last Frame Number", 2, 
&iLastFrame, LF, textCB);
      LastFrame->set_int_limits(000, 20000);
      LastFrame->set_alignment(GLUI_ALIGN_RIGHT);
      LastFrame->set_w(100);
      GLUI_Panel *Buttons = glui->add_panel_to_panel(FilePanel, "
", 0);
      glui->add_button_to_panel(Buttons, "LOAD", LOAD_BUTTON, 
buttonCB);
```

```
      glui->add_column_to_panel(Buttons, 0);
      glui->add_button_to_panel(Buttons, "Play", PLAY_BUTTON,
buttonCB);
      glui->add_separator();

      // Cropping Controls
      GLUI_Panel *CropPanel = glui->add_panel("Image Cropping");
      GLUI_Spinner *XMAX = glui->add_spinner_to_panel(CropPanel,
"Y max", 2, &iXmax, xmax, textCB);
      XMAX->set_int_limits(1, 239);
      XMAX->set_alignment(GLUI_ALIGN_RIGHT);
      GLUI_Spinner *XMIN = glui->add_spinner_to_panel(CropPanel,
"Y min", 2, &iXmin, xmin, textCB);
      XMIN->set_int_limits(0, 238);
      XMIN->set_alignment(GLUI_ALIGN_RIGHT);
      GLUI_Spinner *YMAX = glui->add_spinner_to_panel(CropPanel,
"X max", 2, &iYmax, ymax, textCB);
      YMAX->set_int_limits(1, 319);
      YMAX->set_alignment(GLUI_ALIGN_RIGHT);
      GLUI_Spinner *YMIN = glui->add_spinner_to_panel(CropPanel,
"X min", 2, &iYmin, ymin, textCB);
      YMIN->set_int_limits(0, 318);
      YMIN->set_alignment(GLUI_ALIGN_RIGHT);

      // Drawing Method
      GLUI_Panel *StylePanel = glui->add_panel("Drawing Style");
      GLUI_RadioGroup *StyleGroup = glui-
>add_radiogroup_to_panel(StylePanel, &styleType, -1,styleCB);
      glui->add_radiobutton_to_group(StyleGroup, "Points");
      glui->add_radiobutton_to_group(StyleGroup, "Lines");

      // Time Step
      GLUI_Panel *TimePanel = glui->add_panel("Time Adjustment");
      GLUI_Spinner *TimeStep = glui-
>add_spinner_to_panel(TimePanel, "Frame Number", 2, &iTimeStep,
TS, textCB);
      TimeStep->set_int_limits(0,1000);
      GLUI_Spinner *Delay = glui->add_spinner_to_panel(TimePanel,
"Display Rate- Hz ", 2, &iFrameRate, DI, textCB);
      Delay->set_int_limits(0,100);
      glui->add_checkbox_to_panel(TimePanel, "Loop Animation",
&iLoop);

      // Color Modification
      GLUI_Panel *ColorPanel = glui->add_panel("Color
Adjustment");
      GLUI_Spinner *MaxColor = glui-
>add_spinner_to_panel(ColorPanel, "Upper Color Bound", 2,
&iUpperColor, UC, textCB);
```

```
      MaxColor->set_int_limits(0,800);
      GLUI_Spinner *MinColor = glui-
>add_spinner_to_panel(ColorPanel, "Lower Color Bound", 2,
&iLowerColor, LC, textCB);
      MinColor->set_int_limits(0,800);

      // Camera Goodness
      GLUI_Panel *cameraPanel = glui->add_panel("Camera
Manipulation Mode");
      GLUI_Rotation *camRotationManip = glui-
>add_rotation_to_panel(cameraPanel, "Camera Rotation", camRotMat,
            CAMROTATE,CameraManipCB);
      camRotationManip->reset();
      glui->add_column_to_panel(cameraPanel, true);
      GLUI_Translation *trackXYManip = glui-
>add_translation_to_panel(cameraPanel, "Track XY",
GLUI_TRANSLATION_XY,
            camTrack, TRACK, CameraManipCB);
      glui->add_column_to_panel(cameraPanel, true);
      GLUI_Translation *dollyManip = glui-
>add_translation_to_panel(cameraPanel, "Dolly",
GLUI_TRANSLATION_Z,
            &camDolly, DOLLY, CameraManipCB);

      // File Exporting
      GLUI_Panel *exportPanel = glui->add_panel("Data
Processing");
      glui->add_checkbox_to_panel(exportPanel, "Show Export
Points", &ExportPoints);
      //GLUI_Spinner *CamDist = glui-
>add_spinner_to_panel(exportPanel, "Camera Distance (m)",
GLUI_SPINNER_FLOAT,
      //    &fCamDist, CD, textCB);
      //CamDist->set_float_limits(0.0,2.0);
            // List box to handle texture type selection
      GLUI_Listbox *ExportPosition;
      ExportPosition = glui->add_listbox_to_panel(exportPanel,
"Export", &iExportPosi, EXPORT_POSI, buttonCB);
      ExportPosition -> add_item(0 , "Position #0");
      ExportPosition -> add_item(1 , "Position #1");
      ExportPosition -> add_item(2 , "Position #2");
      ExportPosition -> add_item(3 , "Position #3");
            // Export center position
      GLUI_Translation *trackXYCenter = glui-
>add_translation_to_panel(exportPanel, "Center XY",
GLUI_TRANSLATION_XY,
            centerTrack, cTRACK, textCB);
      glui->add_button_to_panel(exportPanel, "EXPORT",
EXPORT_BUTTON, buttonCB);
```

```
      GLUI_Panel *end = glui->add_panel(" ",0);
      glui->add_button_to_panel(end,"RESET", RESET_BUTTON, 
buttonCB);
      glui->add_column_to_panel(end,false);
      glui->add_button_to_panel(end,"EXIT", EXIT_BUTTON, 
buttonCB);
      glui->set_main_gfx_window( main_window );
      GLUI_Master.set_glutIdleFunc( playFunc );
      glui->sync_live();
}// end initGLUI()

void InitLists( int x )
//=================================================================
// InitLists Function
// This function creates the display lists
// Should be called in Main
//*****************************************************************
{
      switch( x )
      {
      case 0:
            AxesList = glGenLists( 1 );
            glNewList( AxesList, GL_COMPILE );
            drawAxes();
            glEndList();

            DataList = glGenLists( 1 );
            glNewList( DataList, GL_COMPILE );
            drawData( styleType );
            glEndList();

            ExportList = glGenLists( 1 );
            glNewList( ExportList, GL_COMPILE );
            drawExports( 0 );
            glEndList();

            ScaleList = glGenLists( 1 );
            glNewList( ScaleList, GL_COMPILE );
            drawScale();
            glEndList();
            break;

      case 1:                 // regenerate axes display list
            AxesList = glGenLists( 1 );
            glNewList( AxesList, GL_COMPILE );
            drawAxes();
            glEndList();
            cout << "AxesRegen" << endl;
            break;

      case 2:           // regenerate data display list
            DataList = glGenLists( 1 );
            glNewList( DataList, GL_COMPILE );
            drawData( styleType );
            glEndList();
            cout << "DataRegen" << endl;
            break;

      case 3:           // regenerate export display list
            ExportList = glGenLists( 1 );
            glNewList( ExportList, GL_COMPILE );
            drawExports( 0 );
            glEndList();
            cout << "ExportRegen" << endl;
            break;

      case 4:           // regenerate scale display list
            ScaleList = glGenLists( 1 );
            glNewList( ScaleList, GL_COMPILE );
            drawScale();
            glEndList();
            cout << "ScaleRegen" << endl;
            break;

      default: break;
      }// end switch

}// end InitLists

void initScene()
//=================================================================
//    initScene Function
//    Purpose:    Provides the initialization infromation used
//                 by OpenGl to display a scene.
//    Input:      None, function is loaded once during startup of
//                 the program
//    Output:           OpenGl parameters are set.
//                      Lighting- Position, Diffusion,
//                       Specularity, and Ambient set
//                      Cleared window to have a grey background
//                      Enabled depth testing
//                      Repeat for other windows- Side and Top
//                       views
//*****************************************************************
{
      printf("Init...\n");
      int sx = glutGet( GLUT_SCREEN_WIDTH );    // get native
screen width
      int sy = glutGet( GLUT_SCREEN_HEIGHT );   // get native
screen width
      printf("Monitor Resolution %i x %i\n", sx, sy );

      glutSetWindow( main_window );
      float light0_pos[] = {1.0, 0.0, 1.0, 0.0};
      float diffuse0[] = {1.0, 1.0, 1.0, 0.5};
      float ambient0[] = {0.1, 0.1, 0.1, 1.0};
      float specular0[] = {1.0, 1.0, 1.0, 0.5};
      glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT0);
      glShadeModel(GL_SMOOTH);
      glEnable(GL_COLOR_MATERIAL);
      glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
      glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
      glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
      glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
      glClearColor(0.5, 0.5, 0.5, 1.0);                     //
set window to grey background
      glEnable(GL_DEPTH_TEST);
      // enable depth test, to avoid depth fighting

      // Load Basic View
      ifstream inData;
      string sBase = "Initial_View.csv";
      inData.open(sBase.c_str());
      if(inData){
            for(int i = 0; i<240; i++){
                  for(int j=0; j<320; j++){
                        inData >> aiTemps[0][i][j];
                        inData.ignore(1);
                  }
            }
      }
      else cout << "Base file did not open." << endl;
      inData.close();
} // end initScene
```

```
void loadFile()
//=================================================================
//    loadFile Function
//    Purpose:    Load the file data into memory.
//    Input:      Control is passed from the buttonCB function.
//    Output:     Data is loaded and read into memory.
//                If a file input error is encountered the user
//     is alerted by a message in the terminal.
//*****************************************************************
{
      ifstream inData;
      string sBase = "c:\\MHI-TEST-114\\MHI-TEST-114";
      string sExtension = ".csv";
      string sNumber = "Null";
      string sFinal = "Null";
      int iFrameTotal = iLastFrame-iFirstFrame;
      for(int frame = 0; frame<iFrameTotal; frame++)
      {
            stringstream out;
            out << frame+iFirstFrame;
            sNumber = out.str();
            sFinal = sFile + sNumber + sExtension;
            cout << "Loading: " << sFinal << endl;

            inData.open(sFinal.c_str());
            if(inData)
            {
                  for(int i = 0; i<240; i++)
                  {
                        for(int j=0; j<320; j++)
                        {
                              inData >> aiTemps[frame][i][j];
                              inData.ignore(1);
                        }
                  }
            }
            else cout << "File did not open." << endl;
            inData.close();

      }
      FILE *fp = fopen( "Texture.tex","wb" );
      if(fp==NULL)
      {
            cout << "Cannot create texture file" << endl;
      }
      else{
            fwrite( &NUMS, 4, 1, fp );
            fwrite( &NUMT, 4, 1, fp );
            fwrite( &NUMP, 4, 1, fp );

            float zero = 0.;
            float value;

            for(int p=0; p<2*NUMP; p=p+2){
                  for(int t=0; t<NUMT; t++){
                        for(int s=31; s<31+NUMS; s++){
                              if(t < 239 && s < 319)
                        // write true value when bounds in range
                                    value = aiTemps[p][t][s];
                              else value = zero;
      // write temp of 0 when out of range for the actual data
                              fwrite( &value, 4, 1, fp );
                              fwrite( &zero, 4, 1, fp );
                              fwrite( &zero, 4, 1, fp );
                              fwrite( &zero, 4, 1, fp );
                        }// end s
                  }// end t
            }// end p
            fclose(fp);       // close the file
            cout << "Texture.tex created " << endl;
      }//else
}// end loadFile

void myDisplay()
//=================================================================
//    myDisplay Function
//    Purpose: Control the display window
//    Input:      Function is called everytime the canvas needs
//                 to be redrawn
//    Output:     The canvas is redrawn.
//*****************************************************************
{
      // Test Display List Regeneration flags
      if(AxesRegen == 1){
            InitLists(1);
            AxesRegen = 0;
      }
      if(DataRegen == 1){
            InitLists(2);
            DataRegen = 0;
      }
      if(ExportRegen == 1){
            InitLists(3);
            ExportRegen = 0;
      }
      if(ScaleRegen == 1){
            InitLists(4);
            ScaleRegen = 0;
      }
      glutSetWindow( main_window );
      glColorMaterial(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE);
      glClear( GL_COLOR_BUFFER_BIT  | GL_DEPTH_BUFFER_BIT);
      // Draw images for viewport #1
      glViewport(VP[0].x, VP[0].y, VP[1].x, VP[1].y);
            // draw viewport 1
      gluLookAt( 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
      // move camera away from object
      glLoadIdentity();
      glPushMatrix();               // Draw Scale
      glTranslatef(-150., 00.0, -2.0);
      //glCallList( ScaleList );
      glPopMatrix();
      glPushMatrix();               // Draw Axes
      // handle camera translation
      glTranslatef(camTrack[0]+200.0,
            camTrack[1], -camDolly-400.0);
      glRotatef((GLfloat)Xrot, 1.0, 0.0, 0.0);
      glRotatef((GLfloat)Yrot, 0.0, 1.0, 0.0);
      glRotatef(45.0,  0.0,  1.0,  0.0 );
      glMultMatrixf(camRotMat);     // handle camera rotation
      glScalef(Scale, Scale, Scale);// scale the scene
      glCallList( AxesList );
      glCallList( DataList );
      // draw lines at export points
      if(ExportPoints != 0)   glCallList( ExportList );
      glPopMatrix();

      // Draw images for viewport #2
      glViewport(VP[3].x, VP[3].y, VP[5].x, VP[5].y);
      // draw viewport 1
      gluLookAt( 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
      // move camera away from object
      glLoadIdentity();
      glPushMatrix();
      // handle view translation
      glTranslatef(-150.0, -150.0, -2.0);
      // rotate scene to proper view
      glRotatef(90.0,  0.0,  1.0,  0.0 );
      glScalef(VPScale[1], VPScale[1], VPScale[1]);
      // scale the scene
      glCallList( AxesList );
      glCallList( DataList );
      if(ExportPoints != 0)   glCallList( ExportList );
      // draw lines at export points
```

```
      glTranslatef(400.0, 0.0, 400.0);
                  // handle view translation
      glRotatef(-90.0,  0.0,  1.0,  0.0 );
                  // rotate scene to proper view
      glCallList( AxesList );
      glCallList( DataList );
      if(ExportPoints != 0)   glCallList( ExportList );
      // draw lines at export points
      glPopMatrix();

      // Draw images for viewport #3
      glViewport(VP[2].x, VP[2].y, VP[4].x, VP[4].y);
      // draw viewport 3
      gluLookAt( 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
      // move camera away from object
      glLoadIdentity();
      glPushMatrix();
      glTranslatef(-150.0, -150.0, -200.0);
            // handle translate object
      glRotatef( 90.0, 1.0, 0.0, 0.0 );
            // rotate image into proper view
      glRotatef( 90.0, 0.0, 1.0, 0.0 );
      glScalef( 2.5*VPScale[2], 0.001, VPScale[2]);
      // scale the scene
      glCallList( AxesList );
      glCallList( DataList );
      if(ExportPoints != 0)   glCallList( ExportList );
      // draw lines at export points
      glPopMatrix();

      glFlush();
      glutSwapBuffers();
      glui->sync_live();
}// end myDisplay()

void myGlutReshape(int x, int y)
//=================================================================
//    myGlutReshape Function
//    Purpose:    This function will change the viewvolume and
//          view port based on the users changes to the viewing
//          window.
//    Input:      The function is passed the new window
//                 size as integers
//    Output:     None, the viewvolume and viewport are
//                 reshaped by reference. The variables fRb,
//                 fLb, fTb, fBb, and fov are updated
//*****************************************************************
{
      printf("Reshaping\n");
      glutSetWindow( main_window );
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-200., 1000.0, -200.0, 1000.0, 0.1, 1000.0);
      glMatrixMode(GL_MODELVIEW);

      VP[0].x = 0;
      VP[0].y = 0;
      VP[1].x = x - y/2;
      VP[1].y = y;
      VP[2].x = x - y/2;
      VP[2].y = 0;
      VP[3].x = x - y/2;
      VP[3].y = y/2;
      VP[4].x = x;
      VP[4].y = y/2;
      VP[5].x = x;
      VP[5].y = y;

      VPScale[0] = 1.0; // set the scale for viewport #1
      VPScale[1] = 0.5; // set the scale for viewport #2
      VPScale[2] = 1.0; // set the scale for viewport #3

}// end myGlutReshape

void myMouse(int button, int state, int x, int y)
//=================================================================
//    myMouse Function
//    Purpose:    This function will handle the mouse clicking
//                Function from the "Joe Graphics" code
//                 provided by Dr. Bailey
//    Input:      Function is called when the mouse button
//                 is depressed
//    Output:     None, the Rx and Ry parameters will be updated
//*****************************************************************
{
      printf("Mouse clicked ...\n");
      int b;      // button identifier
      switch( button )
      {
      case GLUT_LEFT_BUTTON:        b = LEFT;
                                                break;
      case GLUT_MIDDLE_BUTTON:      b = MIDDLE;
                                                break;
      case GLUT_RIGHT_BUTTON:       b = RIGHT;
                                                break;
      default:                            b = 0;
                                                cout << "Unknown
Mouse Button " << endl;
```

```
                                          break;
      }
      // button down sets the bit, up clears the bit
      if( state == GLUT_DOWN ){
            Xmouse = x;
            Ymouse = y;
            ActiveButton |= b;      // set the bit
      }
      else{
            ActiveButton &=~b;      // clear the bit
      }
} // end myMouse()

void myMotion(int x, int y)
//=================================================================
//    myMotion Function
//    Purpose:    This function will handle the mouse moving
//                 the 3D render of the point clound in viewport #1
//                Function from the "Joe Graphics" code provided
//                by Dr. Bailey.
//    Input:      Function is called when the mouse is moved
//                 and the button down
//    Output:     None, the Rx and Ry parameters will be updated
//*****************************************************************
{
      int dx, dy;
      cout << "Mouse Motion" << endl;

      dx = x - Xmouse;
      dy = y - Ymouse;
      if((ActiveButton & LEFT)!=0){
            switch( LeftButton ){
            case ROTATE:      Xrot +=( ANGFACT*dy*0.1 );
                        Yrot +=( ANGFACT*dx*0.1 );
                        break;
            case SCALE:       Scale+= SCLFACT * float(dx-dy);
                        if(Scale < MINSCALE) Scale = MINSCALE;
                        break;
            }
      }
      if((ActiveButton & RIGHT) != 0)
      {
            Scale += SCLFACT * float(dx-dy)*0.1;
            // keep object from disappearing
            if(Scale < MINSCALE) Scale = MINSCALE;
      }
      glutSetWindow( main_window );
      glutPostRedisplay();
}// end myMotion
```

```
void playFunc( void )
//=================================================================
//     playFunc Function
//     Purpose:    Process the animation variables
//     Input:            Control is passed from the idle function
//     Output:           Frame Number is updated based on the
//*****************************************************************
{
      if(iPlay == 1){
            int NumFrames = iLastFrame - iFirstFrame;
            int duration, place, frame;
            duration =
int(1000.*float(NumFrames)/float(iFrameRate));        // duration
of animation in ms
            place = (glutGet(GLUT_ELAPSED_TIME) - iStartTime ) %
duration;   // place in time of animation
            frame =
int(float(place)*float(NumFrames)/float(duration));         //
determine current frame number

            iFrameNumber = frame;
            iTimeStep++;
            DataRegen = 1;
                                    // update data display list
            cout << iFrameNumber << endl;
            if((iLoop != 1) && (iFrameNumber == NumFrames - 3))
iPlay = 0;
            if(iFrameNumber > NumFrames - 3){
                  iFrameNumber = 0;
                  iTimeStep = 0;
            }

      }                                                       //
end if-end of animation test

      glutSetWindow( main_window );       // update windows
      glutPostRedisplay();
}// end playFunc
```

```
void Reset( void )
//=================================================================
//    Reset Function
//    Purpose:    Reset the variables to the default conditions
//    Input:      None
//    Output:     All global variables are returned to
//                 their default
//*****************************************************************
{
      ActiveButton = 0;
      Scale = 1.0;
      Xrot = Yrot = 0.0;
      iFirstFrame = 1;
      iLastFrame = 100;
      iXmin = 0;        // X minimum cropping variable
      iXmax = 239;      // X maximum cropping variable
      iYmin = 0;        // Y minimum cropping variable
      iYmax = 319;      // Y maximum cropping variable
      iTimeStep = 0;    // displayed time step variable
      iFrameRate = 20;  // variable to hold the timer
                              delay in milliseconds
      iFrameNumber = 0; // currently loaded frame number
      iLoop = 0;
      iPlay = 0;        // variable to hold play info—
                        0=still, 1==play
      iLowerColor = 0;        // lower shading bound
      iUpperColor = 800;            // upper shading bound
      camRotMat[1] = camRotMat[2] = camRotMat[3] = 0.;
      camRotMat[4] = camRotMat[6] = camRotMat[7] = 0.;
      camRotMat[8] = camRotMat[9] = camRotMat[11] = 0.;
      camRotMat[12] = camRotMat[13] = camRotMat[14]= 0.;
      camRotMat[0] = 1.0;
      camRotMat[5] = 1.0;
      camRotMat[10] = 1.0;
      camRotMat[15] = 1.;

      camTrack[0] = 0.0;      // camera track variables
      camTrack[1] = 0.0;      // camera track variables
      camDolly = 0.0;         // camera dolly variable

      // export variables
      centerTrack[0]= 180.0;        // center of the viewing area
      centerTrack[1]= 120.0;        // center of the viewing area
      iExportPosi = 0;              // holder for the export
                                     position of the data
      fCamDist = 0.8;               // variable to hold the
             distance between the camera and the measured surface
      ExportPoints = 0; // != 0 when the points should be drawn

      AxesRegen = 0;
      DataRegen = 0;
      ExportRegen = 0;
      ScaleRegen = 0;
}// end Reset

void styleCB(int type)
//=================================================================
//    styleCB() function
//    Purpose:    Handle the style change in the temperature
//                 display method
//    Input:      function called when the style radio buttons
//                 are change
//    Output:     prompt sent to terminal alerting the user of
//                 change and a redraw is called.
//*****************************************************************
{
      cout << "Changing display style.\n";
      glutSetWindow( main_window );
      glutPostRedisplay();
}// end styleCB

void textCB(int id)
//=================================================================
//    textCB() function
//    Purpose:    Handle the user inputs from the text field
//                 and spinners
//    Input:      The file name, frame first/last and time
//                 step spinners all call this function when they
//                 are modified
//    Output:           state is updated and redraw is called
//*****************************************************************
{

      switch(id)
      {
      case OBJ_TEXTFIELD:     printf("Text field edited \n");
                  // alert user
      sFile = objFileNameTextField->GLUI_EditText::get_text();
      // store entered string
            break;
      case FF:
            printf("First Frame changed \n");// alert user
            break;
      case LF:    printf("Last Frame Changed\n");// alert user
            break;
      case TS:
```

```
printf("Time Step Changed\n");// alert user
iFrameNumber = iTimeStep;// set the current frame number
DataRegen = 1;          // regerate data DisplayList
      break;
case DI:
      printf("Delay Interval Changed\n"); // alert user
      break;
case LC:
      printf("Lower Color Bound Modified\n");// alert user
      ScaleRegen = 1;   // regenerate Scale DisplayList
      break;
case UC:
      printf("Upper Color Bound Modified\n");// alert user
      ScaleRegen = 1;   // regenerate Scale DisplayList
      break;
case xmin:
      printf("X minimum crop adjusted\n");// alert user
      ;// test iXmax to ensure atleast on line is displayed
      if(iXmin >= iXmax) iXmax = iXmin + 1
            DataRegen = 1;// regenerate data DisplayList
      break;
case xmax:
      printf("X maximum crop adjusted\n");// alert user
      ;// test iXmin to ensure atleast on line is displayed
      if(iXmin >= iXmax) iXmin = iXmax - 1
      DataRegen = 1;// regenerate data DisplayList
            break;
case ymin:
      printf("Y minimum crop adjusted\n");// alert user
      // test iYmax to ensure atleast on line is displayed
      if(iYmin >= iYmax) iYmax = iYmin + 1;
            DataRegen = 1;// regenerate data DisplayList
      break;
case ymax:
      printf("Y maximum crop adjusted\n");// alert user
      // test iYmin to ensure atleast on line is displayed
      if(iYmin >= iYmax) iYmin = iYmax - 1;
      DataRegen = 1;    // regenerate data DisplayList
      break;

case CD:
      cout << "Camera distance updated." << endl;
      break;

case cTRACK:
      cout << "Window center being moved.\n";
      if(centerTrack[0] < 0.0)
            centerTrack[0] = 0.0;
      if(centerTrack[0] > 320.0) centerTrack[0] = 320.0;
```

```
            cout << "X pos: " << centerTrack[0]<<endl;

            if(centerTrack[1] < 0.0)      centerTrack[1] = 0.0;
            if(centerTrack[1] > 240.0)    centerTrack[1] = 240.0;
            cout << "Y pos: " << centerTrack[1]<<endl;
            ExportRegen = 1;  // regenerate export DisplayList
            break;

      default:    break;
      }
glutSetWindow( main_window );
glutPostRedisplay();
}// end textCB()
```