High-speed Imaging and Analysis of the Solidification of Undercooled Alloy Melts

by

John W. Lum

S.B., Massachusetts Institute of Technology (1994)

Submitted to the Department of Materials Science and Engineering
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
in Materials Science and Engineering

at the

Massachusetts Institute of Technology
June, 1996

Signature of Author.... ....
nd Engineering
May 10, 1996

Certified by. ....
Merton C. Flemings
[illegible] Supervisor

Accepted by...................................... ....
Michael F. Rubner
TDK Professor of Materials Science and Engineering
Chair, Departmental Committee on Graduate Students

High-speed Imaging and Analysis of the Solidification of Undercooled Alloy Melts

by

John W. Lum

Submitted to the Department of Materials Science and Engineering on May 10, 1996 in partial fulfillment of the requirements for the degree of Master of Science in Materials Science and Engineering.

ABSTRACT

Rapid solidification of undercooled pure nickel and hyperperitectic Fe-(10-30) wt.% Ni alloys has been imaged at sufficiently high spatial resolution (64 x 64 pixels) and temporal resolution (40,500 frames/sec) to observe interfacial shape and solidification velocities exceeding 45 m/s. Imaging was of 8 gram, quartz-fluxed melts at undercoolings between 70 K and 300 K. Dendrite velocities within the melt were calculated from the surface velocities observed employing a simple geometric model of growth. Solidification was found to proceed invariably from a single nucleation point at every composition. In the case of Fe-Ni, primary solidification of both the equilibrium FCC phase and the metastable BCC phase was observed, with phase selection dependent upon the thermal history, composition, presence of heterogeneous nucleants, and degree of undercooling attained. In all cases of primary BCC solidification, a subsequent transition to the FCC phase was observed, resulting in a two-stage "double recalescence" evident in the thermal and video records. A primary solidification map was generated to predict the preferred phase for primary solidification as a function of composition and nucleation temperature for similar experiments with quartz-fluxed Fe-Ni specimens.

The computer-based growth model enabled determination of solidification velocities with much greater precision than had been previously accomplished using pyrometric techniques alone. Growth velocity in pure Ni was found to follow an approximate power-law relationship with respect to undercooling up to some critical value ΔT^*, where $150 \text{ K} < \Delta T^* < 180 \text{ K}$. Growth velocities in BCC Fe-(10-12) wt.% Ni followed a similar power-law relationship, but no velocity transition was observed despite undercoolings of nearly 250 K. However, both pure Ni and Fe-Ni specimens did exhibit a secondary transition in solidification interface morphology. At smaller undercoolings, interface curvature was discontinuous, with tendencies toward growth along preferred crystallographic directions. At larger undercoolings, interfaces were completely smooth, with no discontinuity in curvature caused by irregular growth. Values for the morphological ΔT^* were estimated at 160-170 K for pure Ni and 180-190 K for intermediate Fe-Ni compositions.

Thesis Supervisor: Merton C. Flemings

Title: Toyota Professor of Materials Processing

TABLE OF CONTENTS:

TABLE OF FIGURES:

Acknowledgements

My deepest and most heartfelt gratitude goes once again to my parents, without whose upbringing I would not be nearly so satisfied with the person I am today. You are still the reader I'm writing for, Mom.

I also owe a great deal to Prof. Merton Flemings for supervising my thesis work. He has, during the last two years, provided just enough guidance to keep me moving in the right direction, while simultaneously giving me the opportunity to make the majority of my research-related decisions for myself. I have always felt very trusted during my time under Prof. Flemings' supervision.

On a somewhat more personal note, I would like to thank Lisa for remaining a rock of support for me through the difficulty we've experienced in the past few months. My twin brother David has also been a steady companion during my graduate work, as he has been as far back as I can remember (and probably beyond). We may not have many MIT lunches left before one of us finally strikes out into the Real World! No less important in the past few years were the fun times I spent with Jeanie, Noland, and my other friends around the Institute and elsewhere.

Thanks is also due to my many colleagues here at MIT, both past and present. In particular, Doug Matson has been a constant source of support since I first set foot in room 8-436. I also owe a great deal to Dr. Tom Piccone, both for his crucial help with the experimental apparatus and also for his fine examples of thorough research dissertations which will continue to assist MIT solidification researchers for many years to come. In addition, I'd like to recognize Matthieu, Cars, Qi, Jusuf, Arvind, Chris, Rob, Hua Shen, Honjo, and Elizabeth for offering their help and friendship to me along the way. I'd also like to thank my youngest colleagues, undergraduates Rob and Elissa, for providing their capable assistance during the second year of this work.

Finally, I gratefully acknowledge the sponsorhip of the National Aeronatics and Space Administration during these two successful years of research.

1. Introduction

One of the most interesting solidification phenomena to undergo extensive study in the last 30 years is the rapid solidification behavior of pure elements and alloy melts. Rapidly-solidified materials have proved to offer a wide range of desirable materials properties not exhibited by their conventionally-cast counterparts. For specimens exhibiting at least one thin dimension, numerous rapid solidification techniques have been developed which rely upon ultra-high rates of heat extraction (10^9-10^{12} K/s); examples include melt-spinning, splat-quenching, ion bombardment, laser-pulse heating, and many others.

These methods, however, cannot be used in the production of large-scale ingots in the foundry, because the absence of a thin dimension makes it impossible to obtain the requisite heat-extraction rates. One alternative approach to produce rapid solidification in these cases is to achieve a large undercooling of the liquid melt prior to solidification.

Undercooling techniques enable rapid solidification by lowering the melt temperature well below the liquidus prior to solidification, gradually removing a significant portion of the ingot's sensible heat. The resulting rapid solidification is caused by the increased thermodynamic driving force inherent to the highly-undercooled liquid, and a large portion of the solidification process can be made to occur in an essentially adiabatic manner, a phenomenon known as "recalescence."

The undercooling phenomenon has been under considerable study for decades, and an important parameter of characterization has been the velocity with which the newly-formed solid propagates through the undercooled liquid during the solidification process. Prior film-based and electronic methods of measuring solidification velocities have met with variable success. In general, investigators have achieved either high temporal resolution (strategically-placed photodiodes), or high spatial resolution (conventional film-based cameras)—but not both. As a result, the velocity measurements from these experiments have suffered from a large degree of scatter.

It is only recently that electronic imaging technology has developed to the point where there exists sufficient temporal resolution (up to 40,500 Hz with the system employed in this work) and spatial resolution (a 64 x 64 pixel image) to generate a complete visual record of the rapid solidification of a highly-undercooled specimen. This record, if properly interpreted, can be used to obtain the most accurate velocity measurements yet performed in this type of experiment.

In the current work, an original computer model has been developed to analyze the photographic records of rapid solidification events and measure the propagation velocity of the solid/liquid interface. The method is first applied to melts of high-purity nickel—an extensively-studied material—in order to demonstrate the validity and accuracy of the technique. Analysis is subsequently extended to the solidification of binary Fe-Ni alloys, with special emphasis placed upon the exploration of the metastable δ-phase and related double-recalescence phenomenon.

2. Literature Survey

2.1 Pure Nickel

2.1.1 Methods of Investigation

Numerous investigators have studied dendrite growth velocities in undercooled high-purity elements, particularly nickel, since the undercooling phenomenon was first observed in the Au system by Van Riemsdyk [1] in 1880. Walker [2] made the first thorough investigation of undercooled nickel melts using photodiodes to measure the time required for an interface to propagate along a column of molten material. Colligan and Bayles [3] used photodiodes in conjunction with high-speed cinematography, determining surface solidification morphology in addition to solidification velocities

Subsequent velocity studies have primarily involved refinements in the placement of photodiodes and interpretation of the thermal record. Piccone, et al [4], for example, employed a single photodiode focused on a portion of an undercooled, quartz-encased nickel ingot in conjunction with a digital oscilloscope. Velocities were determined using the relationship

$$V = \frac{D}{\Delta t}, \tag{2.1}$$

where D is the diameter of the pyrometer view field and Δt is the signal "rise time" for the recalescence, as depicted in Figure 2.1. This technique—like any technique that does not directly image the solidification interface—suffers from several inherent uncertainties regarding both D and Δt: first, the D value is an inaccurate measure of the distance traveled by the solidification interface unless it is certain that growth proceeded directly across the field of view of the pyrometer. In addition, a large degree of interpretation is required to derive a Δt value from a given thermal record; the sigmoidal shape of the temperature transition makes it possible to determine Δt consistently, but not precisely.

Other researchers have employed electromagnetic levitation (containerless processing) to study undercooled Ni melts. One such example is the work of Willnecker, et al [5], who used two hemispherical photodiodes and calculated solidification velocities using signal rise time and geometric considerations. Hofmeister, et al [6], used a single, tightly-focused (1 mm) photodiode, supposedly to minimize the effects of potential multiple nucleation events. In the latter investigation, the reduced pyrometer spot size may have mitigated the effect of the unconstrained solidification path. In the former study, measurement accuracy may have been improved by constraining the solidification path via manually-induced nucleation at the south pole of the specimen. However, in both investigations, the results of the work again depend upon the interpretation of sigmoidal thermal profiles.

Eckler and Herlach [7] used a polar photodiode and a unique capacitance trigger in their work, defining more accurately the duration of the solidification event. In this study, the moment of nucleation was marked by a jump in the capacitance of a metallic stimulation needle upon contact with the base of the specimen. As a result, pyrometry was only required to detect the end of the solidification event.

Bassler, et al [8], were among the first to use a linear array of photodiodes—later moving to a two-dimensional array—in an effort to improve the spatial resolution of their studies. An iterative computer routine was employed in conjunction with the photodiode records to produce a calculated velocity associated with minimum deviation from the observed data. The work was also based upon the assumption that the solid/liquid interface remains convex to the liquid at all undercoolings; deviations were assumed to result from multiple nucleation events, and all such trials were thrown out. However, both the current investigation and prior work [3] have shown numerous deviations from the convex-interface restriction which clearly result from a single nucleation point. As a result, the validity of Bassler's "multiple nucleation" assumption is in question.

2.1.2 Experimental Results

With the exception of Hofmeister and Bassler, the results of the above work are qualitatively and quantitatively similar (Figure 2.2). At small undercoolings, the investigators show an approximate power-law relationship between undercooling and solidification velocity,

$$V = k\,(\Delta T)^{\beta}, \tag{2.2}$$

where β varies between 2 and 3. This behavior is relatively well-predicted by the "LKT" theory of Lipton, et al [9], particularly when a kinetic parameter is included in the analysis. At larger undercoolings, the results of prior investigations diverge more sharply, but many conclude that there exists some critical undercooling, ΔT^*, in the range 170-190K, above which a normal LKT analysis is no longer valid.

In addition, several researchers [10,11,12] have observed a corresponding microstructural transition at ΔT^* in a similar range of values, marking the shift from a columnar dendritic grain structure to a finer, equiaxed structure. Notable also is the transition in solidification interface morphology identified by Colligan and Bayles in their high-speed cinematographic work. It was determined that surface interfaces displayed continuous curvature at undercoolings greater than 160 K, but shifted to a discontinuous, angular morphology as undercooling fell below 145 K.

The scatter of undercooling-velocity data at undercoolings above ΔT^* is quite remarkable, bounded at the lower end by the results of Hofmeister and Bassler, who report a constant velocity of about 20 m/s at all undercoolings larger than ΔT^*. They propose that interfacial attachment kinetics is the limiting factor at these large undercoolings, while no satisfactory explanation has yet been proposed to explain the relationships observed by the other investigators.

Scatter at undercoolings less than ΔT^* is much less pronounced than at higher undercoolings and may be explained by variations in convective effects and methods of pyrometry, deviations from assumed specimen geometry, and, as supported by Eckler, et al [7], uncertainty regarding the growth direction of the dendrites.

2.2 Iron-Nickel

2.2.1 Metastable Phase Solidification

One of the first observations of the tendency of hyperperitectic Fe-Ni alloys to solidify into more than one crystalline phase was made by Cech [13] in 1956. It was found that micron-range particles of Fe-29.5 at.% Ni cooled from the liquid state formed not only the expected equilibrium FCC phase upon solidification, but also frequently solidified into an unexpected BCC phase. A simple graphical extension of the liquidus and solidus lines of the (BCC) δ phase on the equilibrium Fe-Ni phase diagram (Figure 2.3) supported the hypothesis that molten droplets of Fe-Ni undercooled below these extensions could solidify into a metastable form of δ Fe-Ni.

2.2.1.1 Theoretical Basis

Kelly, et al [14] presented a strong theoretical argument for the solidification of alternative (or nonequilibrium) crystalline phases from an undercooled melt based on the thermodynamics and kinetics of nucleation and growth. Numerical results for Fe-Ni showed that the metastable phase was indeed preferred for homogeneous nucleation under a wide range of particle sizes, cooling rates, and compositions. In the case of heterogeneous nucleation, wetting angle and other variables entered the analysis, but primary metastable phase solidification remained a theoretical possibility.

2.2.1.2 Powder Studies

Subsequent powder studies have continued to reveal metastable phase formation from the melt in Fe-Ni alloys. Kim, et al [15] reported development of the alternative BCC phase in submicron droplets of Fe-30-, and Fe-40 at.% Ni, but not in Fe-50 at.% Ni. Thoma, et al [16] (an excellent review paper on this topic) found BCC solidification in a wide compositional range, proposing a map to define the structural evolution of solidified Fe-Ni as a function of both composition and undercooling.

Although Libera, et al [17] managed to obtain thermal profiles of 44-46 μm Fe-30 at.% Ni particles upon solidification, they were unable to salvage the individual particles and thus did not perform any microstructural evaluation. Also, the temporal resolution of their pyrometry was insufficient to reveal potential two-stage recalescence behavior.

2.2.1.3 Bulk Studies

Investigations of bulk Fe-Ni specimens have further characterized metastable BCC phase solidification. The Thoma study reported no retained metastable BCC structure in larger (1-3 mm) droplets of Fe-(10 to 30) wt.% Ni; however, it was speculated that the cooling rate in these droplets was insufficient to suppress a solid state BCC-->FCC transformation, and that the initial solid phase to form from the undercooled melt may well have been the metastable δ phase.

Stronger evidence for the proposed transition was supplied by Zhao, et al [18] and Herlach, et al [19], who recorded fast thermal profiles of bulk Fe-Ni solidification events. These efforts revealed a double-recalescence behavior for certain hyperperitectic Fe-Ni specimens at sufficient undercoolings: initial growth of BCC material was followed by a plateau period near the metastable BCC solidus temperature lasting from about 1 ms to several seconds, followed ultimately by a transformation to FCC material from either a liquid/solid mixture or the solid state. These thermal profiles, in agreement with theoretical work by Chuang, et al [20] which more accurately defined the metastable liquidus and solidus lines for δ Fe-Ni, provided an exceedingly strong indication that the metastable δ phase was solidifying directly from the undercooled melt.

2.2.2 Experimental Results

In general, the microstructure and solidification velocities observed in undercooled Fe-Ni melts resemble the results of the pure Ni work. The undercooling-velocity data from Barth, et al [21] for Fe-25- and Fe-30 at.% Ni are summarized in Figure 2.4; the behavior is reasonably well modeled by modified LKT analyses for the equilibrium FCC phase. The investigators argued that the data in Figure 2.4 reflects the existence of a critical undercooling (ΔT^*) for both compositions at 175 K—similar to that of pure Ni—although this appears to be a more valid assessment of the lower data set. Above $\Delta T^*=175$ K, it was surmised that dendrite growth theory no longer offered an accurate prediction of the ΔT-V correspondence, indicating some shift in solidification mechanism. It is worth noting that this study assumed primary solidification of the equilibrium FCC phase in Fe-25- and Fe-30 at.% Ni. The Zhao study found FCC solidification in Fe-30 wt.% Ni, but primary BCC solidification in Fe-5- and Fe-10 wt.% Ni specimens.

Other investigations identified a microstructural ΔT^* which marked a transition from a dendritic structure below ΔT^* to a much finer-grained, "spherical" structure at higher undercoolings. Early work by Kattamis, et al [22] indicated a ΔT^* value of 170 K for Fe-25 wt.% Ni. Abbaschian, et al [23] found a similar value of 175 K for that alloy, while the Barth study reported a microstructural ΔT^* between 135 K and 180 K for a range of Fe-Ni compositions.

One final experimental result of importance is the demonstration of phase selection or "phase seeding" in Fe-Ni through the use of a heterogeneous nucleation trigger of high catalytic potency. Herlach, et al [19], were able to induce solidification of the metastable BCC phase in Fe-(20-30) at.% Ni by surface stimulation with a trigger wire composed of BCC $Fe_{92}Mo_8$. Identical specimens displayed primary FCC solidification when allowed to recalesce spontaneously from similar or greater undercoolings. This result reveals the potential for reliable phase selection in this and other alloy systems through the use of carefully-selected heterogeneous nucleants.

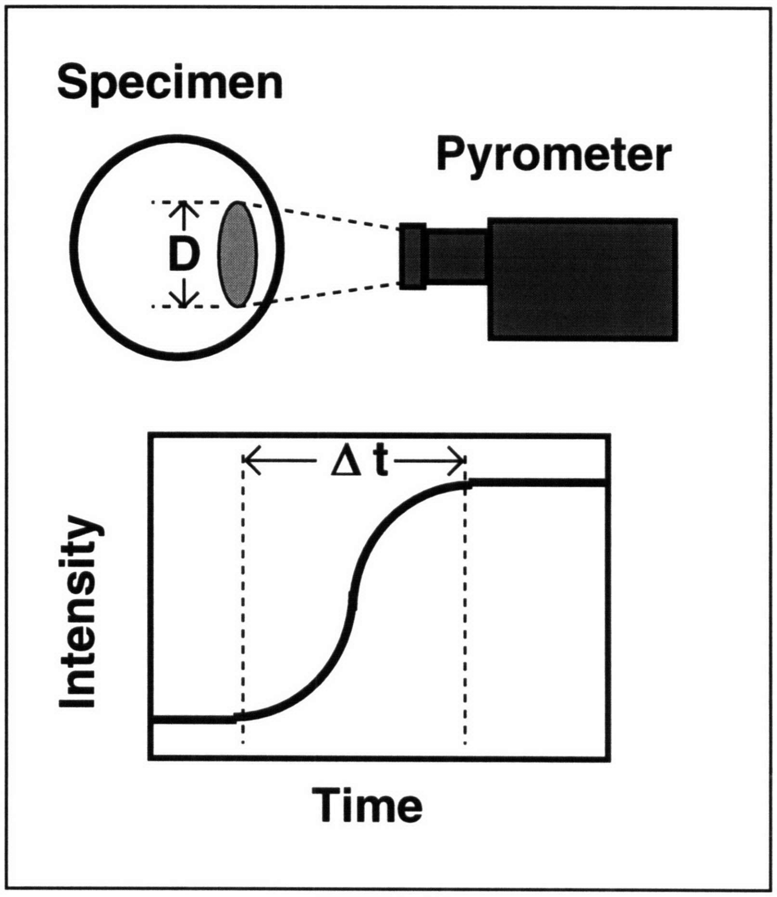

Figure 2.1: Typical pyrometric method to determine solidification velocity, where $V = D / \Delta t$.

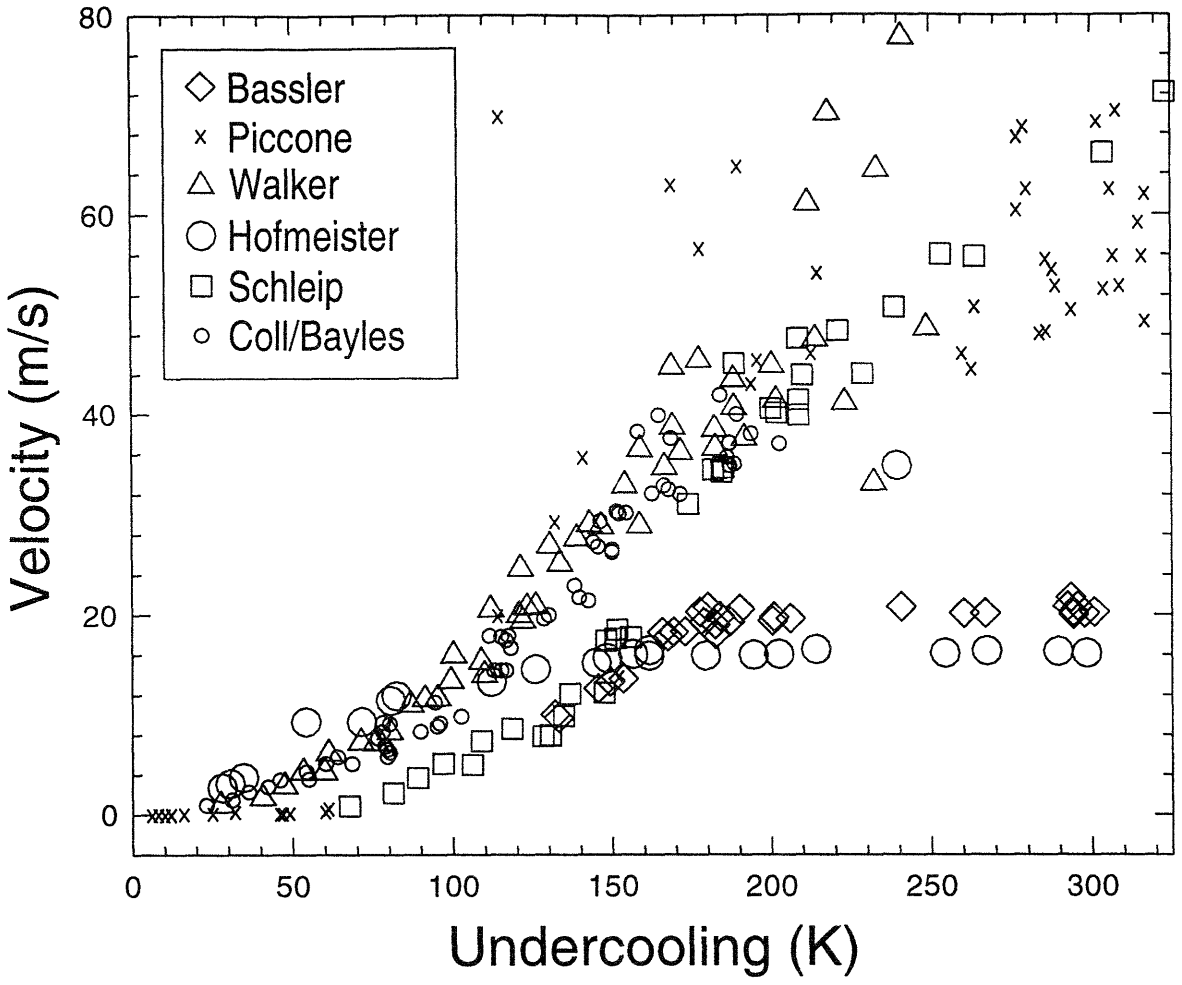

Figure 2.2: Selected prior investigations of pure Ni. References for plotted data: Bassler, et al [8]; Piccone, et al [4]; Walker [2]; Hofmeister, et al [6]; Schleip, et al [12]; Colligan and Bayles [3].

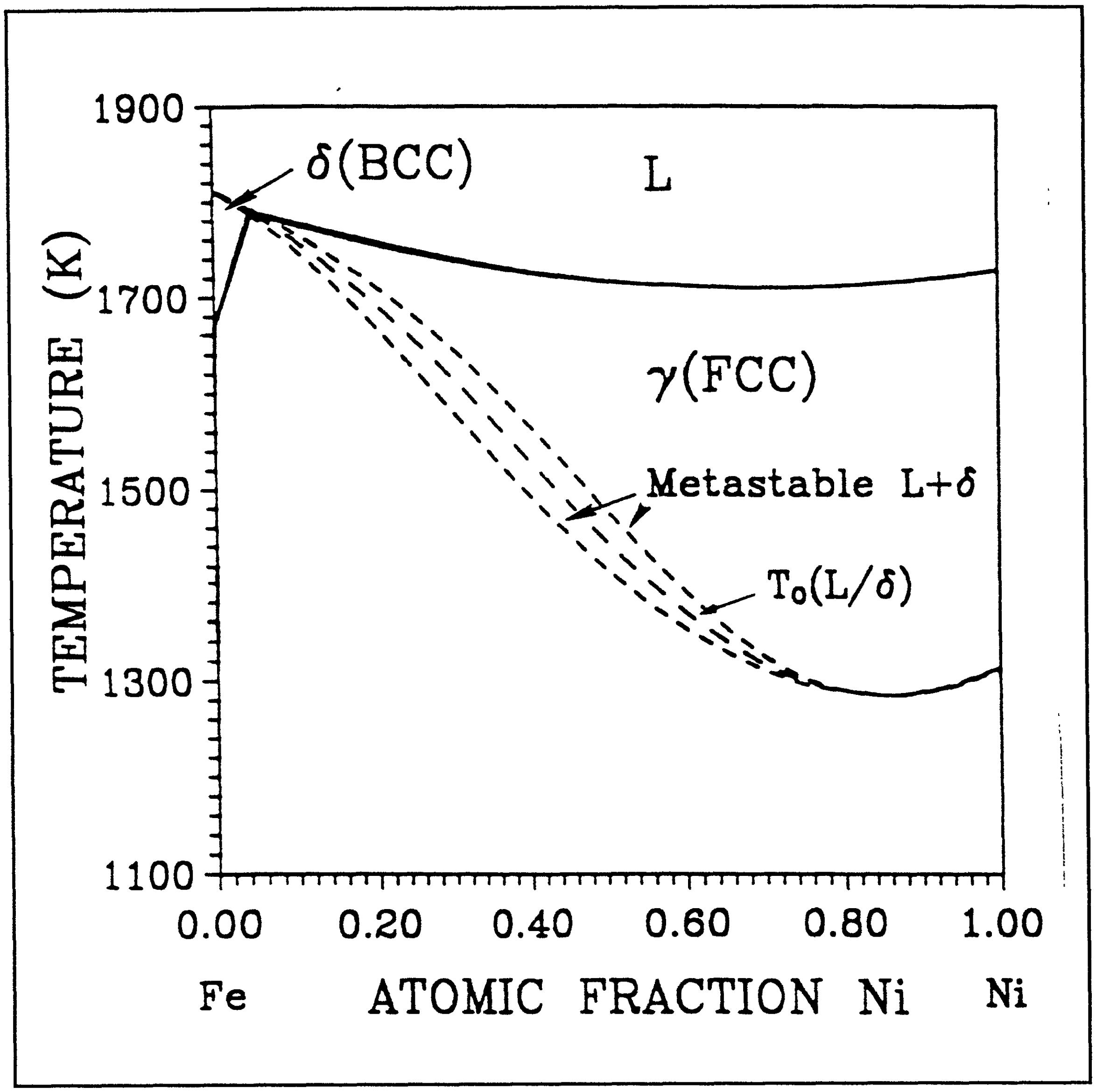

Figure 2.3: Fe-Ni equilibrium phase diagram with calculated metastable extensions (from Chuang, et al [20]).

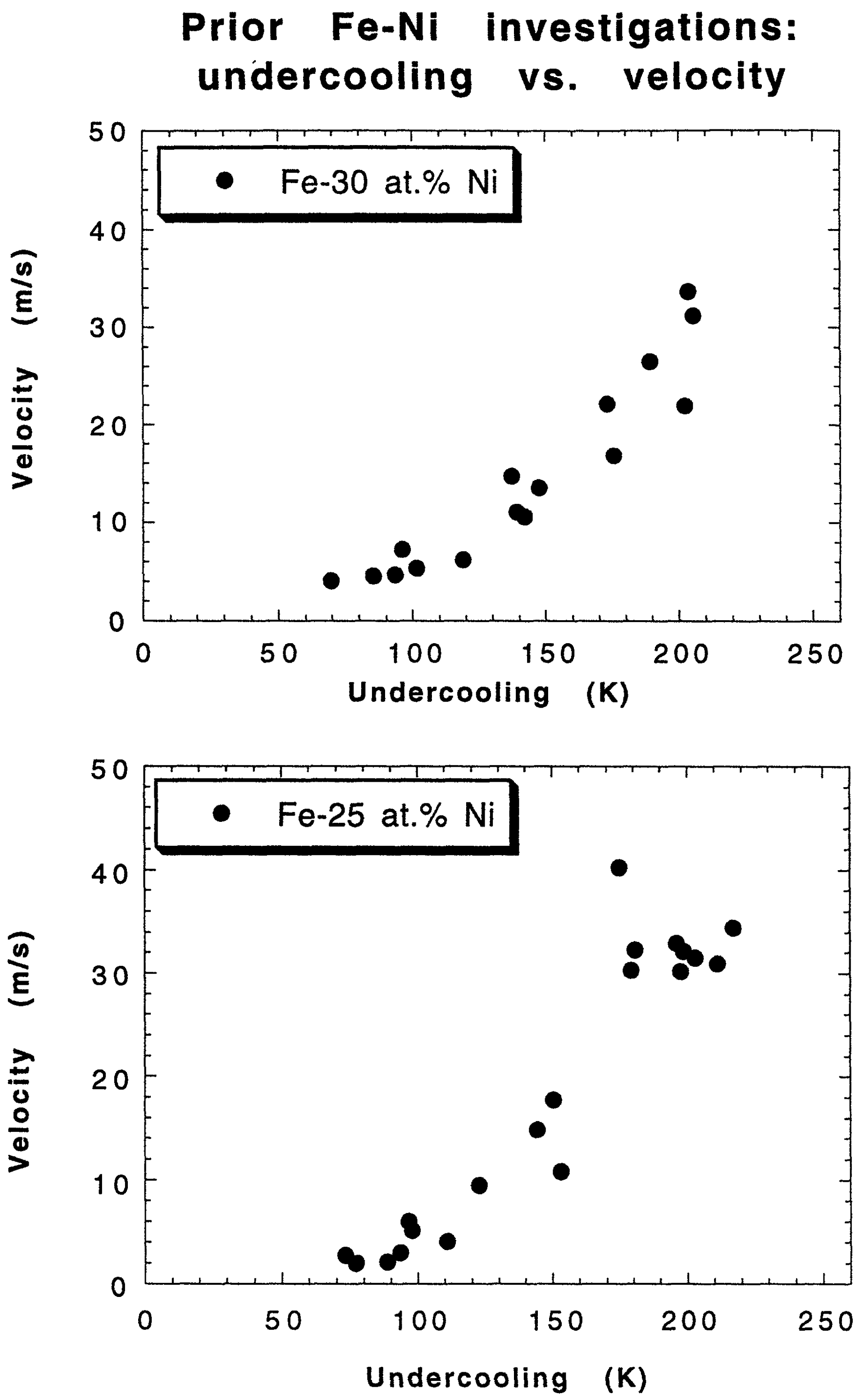

Figure 2.4: Prior velocity calculations for Fe-Ni alloys performed by Barth, et al [21]. A critical undercooling of 175 K was identified for both compositions after comparison with LKT predictions.

3. Experimental Procedure

Figure 3.1 shows a schematic of the apparatus used in the current work to obtain thermal profiles and visual images of undercooled specimens.

3.1 Specimen Preparation

Two lots of high-purity nickel were used for this study, both obtained from Electronic Space Products International. The first lot was 99.9+ pct. (3N) pure wire stock; the second was rod stock of 99.999+ pct. (5N) purity. The iron used in the Fe-Ni specimens was 99.9 pct. pure flake obtained from Johnson Matthey Alfa Aesar.

For each pure Ni specimen, approximately eight grams of either 3N or 5N material was cut, cleaned in ethanol, and then induction-melted inside an open-ended quartz tube on a bed of high-purity quartz frit (see Figure 3.2). The Fe-Ni specimens were prepared in similar manner from portions of 3N Ni wire and Fe flake pre-weighed to a nominal accuracy of ±.001 g. Compositions under study included Fe-10-, 12-, 15-, 20-, and Fe-30 wt.% Ni. For all specimens, prior to melting, the enclosing test tube was evacuated and flushed several times with high-purity Ar gas; subsequently, the gas was allowed to flow continuously through the test tube.

Power for heating was supplied by a Lepel model T-10-3 10 kW radio-frequency current generator operating at a nominal frequency of 400 kHz. Energy was input to the specimen by an 8-turn, 1/8" copper tubing split induction coil. During a typical thermal cycle, the cylindrical ingot was inductively heated through its liquidus temperature to a maximum superheat of 100-150 K. The heating power was then switched off and the specimen was allowed to cool under the flowing Ar until either spontaneous nucleation occurred, or else nucleation was stimulated at the top surface of the ingot using a short length of Fe or Ni wire.

3.2 Pyrometry

3.2.1 Data Acquisition

Temperatures were measured using a silicon photodiode-based Capintec Ratioscope III two-color pyrometer operating at near-infrared wavelength bands centered at 0.81 and 0.95 μm. Pyrometric data was recorded by a Nicolet 4094B digital oscilloscope with two model 4562 plug-ins and an XF-44 twin disk drive. During a typical run, the four differential analog inputs of the Nicolet were used as follows: two inputs sampled the pyrometer channels continuously at a rate of approximately 20-50 Hz, producing a 1-3 minute thermal history of a melt-superheat-solidification cycle. The other two inputs produced a high-speed recalescence profile by triggering upon recalescence and recording at the rate of 0.2-1 MHz for 4-20 ms, retaining an adjustable amount of pre-trigger data.

3.2.2 Data Analysis

The results of each thermal cycle were saved in Nicolet-specific format on 5 1/4" floppy disk for archiving and then downloaded to an IBM PC-AT computer via a GPIB interface using the "Henry" file-transfer package provided by Nicolet. Usually, the 4094LTU transfer program was used, rendering the data into an older "LOTUS 1-2-3" spreadsheet-style format. The data was ultimately transformed into simple tab-delimited column format using a C program (see Appendix A4) in order to facilitate subsequent analysis.

Temperatures were derived from each two-channel set of pyrometer data using the following relation:

$$T_{obs} = T_{ref} + m\,(R_{obs} - R_{ref}), \tag{3.1}$$

where R_{obs} is the observed ratio of the signal intensities of the two pyrometer channels: $R_{obs} = I_{0.95\,\mu m} / I_{0.81\,\mu m}$,

T_{ref} is the temperature of a known reference point on the thermal profile, usually the liquidus temperature for a pure material,

R_{ref} is the ratio value observed at T_{ref}, and

m is the slope of an experimentally-obtained ratio-temperature curve, roughly linear over a range of several hundred degrees

This linear relationship between ratio and temperature is an empirical one only; radiation theory [24] suggests the following dependence:

$$R = k_1 \exp\left(\frac{-k_2}{T}\right), \tag{3.2}$$

with $$k_2 = \left(1.44 \text{ x } 10^4 \ \mu\text{m K}\right)\left(\frac{1}{\lambda_1} - \frac{1}{\lambda_2}\right)$$

This equation predicts a theoretical variation in m, the slope from Eq. 3.1, of about 6% between 1726 K and 1426 K (a 300 K undercooling in pure nickel).

However, the error in temperature measurement is reduced somewhat by defining m as the average slope observed in a wide range of undercoolings during a set of calibration experiments. These experiments require the immersion of a thin, zirconia-sheathed type-B thermocouple in the molten ingot, allowing simultaneous measurement of both pyrometer signal ratio and true specimen temperature. A best-fit linear analysis of the undercooling segment of the resulting ratio-temperature plot (Figure 3.3) yields the average m value for a given run. In the present work, the overall average slope value for nickel was found to be 880 ±20 K^{-1}; this compares well with a value of 850 K^{-1} previously obtained [26] using the same setup. No reliable calibrations were performed for the Fe-Ni material; instead, an approximate value of 900 K^{-1} was assumed. As a result, the absolute accuracy of the temperature measurements is estimated to be ±10 K for nickel and, conservatively, ±25 K for Fe-Ni specimens at the highest undercoolings.

The conversion in Eq. 3.1 was applied interactively to the pyrometer data using a virtual instrument written in LabVIEW[i] running on a 100 MHz Pentium-based IBM-compatible computer. The resulting time-temperature profiles could then be evaluated to determine the extent of undercooling for each thermal cycle.

3.3 Image Acquisition and Analysis

3.3.1 Acquisition

3.3.1.1 Hardware

The camera depicted in Figure 3.1 is a Kodak EktaPro HS Motion Analyzer, Model 4540—a high-speed digital camera. The device supports a maximum acquisition rate of 40,500 frames/sec, at which speed the image consists of a 64 x 64 pixel array with a precision of 8 bits/pixel. A 200 mm lens was employed in this work to capture an approximately 10 mm x 10 mm image from a working distance of about 40 cm.

3.3.1.2 Technique

After each thermal cycle, selected video frames comprising the entire solidification event were recorded onto S-VHS videotape for archiving. In order to perform subsequent computer-assisted image analysis, the images from the videotape were re-digitized and stored on a Macintosh IIci personal computer using a frame-grabber expansion card and an image-processing package called IPLab[ii]. The frame-grabber sampled the videotaped images at twice the pixel resolution of the EktaPro 4540, producing 128 x 128 pixel images with an artificially enhanced resolution. These images were suitable for further analysis and digital image-processing using IPLab.

[i] LabVIEW is a registered trademark of National Instruments, Inc.
ii IPLab is a registered trademark of Signal Analytics, Inc.

3.3.2 Analysis

3.3.2.1 Solidification Interface

The camera observes growth of the solidification front by sensing a change in light intensity (and therefore temperature) near the front comprising the array of dendrite tips. This temperature change takes place over a distance of approximately α/V from the growth front interface, where α is thermal diffusivity for nickel and V is dendrite tip velocity. Taking $\alpha = 1.6 \times 10^{-5}$ m^2/s (refer to Appendix A2 for a summary of physical constants used) and a conservative solidification velocity of 5 m/s, the ratio of 3.2 μm indicates that the thermally measured growth front closely approximates the actual physical interface.

3.3.2.2 Processing

The digitized images of the solidification interface were manipulated as follows: first, the location of the solid/liquid interface was consistently defined in each recorded frame by isolating only those pixels (x_i, y_i) included in a numerically-defined transition zone between the neighboring bright (solid) and dark (undercooled liquid) pixels (Figure 3.4a). The zone typically spanned about 5% of the intensity transition and was centered near 25% maximum brightness.

As this interfacial band was identified in each individual frame, the interface pixels were incrementally incorporated into a 32-bit composite master image (Figure 3.4b). Since, in the general case, it was possible for a specific composite pixel (x_c, y_c) to have appeared in the interfacial bands of multiple frames (for example, a slow-moving interface), special care had to be taken to ensure that the frame-by-frame pixel record could be accurately reproduced from the composite image. Thus, each interface pixel from an individual frame was made to contribute an increase in intensity of 2^N in the corresponding composite pixel, where $N=\{0,1,2,...\}$ was the frame number from which the pixel originated.

3.3.2.3 Composite Image Interpretation

The resulting composite image was a two-dimensional matrix of intensity values which could be interpreted in the following manner: a zero at a given (x,y) position indicated that no pixels from any frame's solid/liquid interfacial band appeared at that location during solidification. Nonzero values were readily interpreted by their binary representation: a nonzero bit at arbitrary bit position M (counting the lowest-order bit as bit 0) indicated the presence of a pixel from the interfacial band of frame M. For example, a value of 0d192 = 0b11000000 at coordinate (45, 63) indicated that the solid/liquid interface had been present at that location during frames 6 and 7 of the solidification record. In general, however, interface motion was fast and predictable enough that each coordinate contained either a zero or a perfect power of two.

The composite image was finally transferred as text to a UNIX-based workstation for velocity analysis. There, another C routine (see Appendix A4) was used to create a formatted datafile listing the interface points by frame along with various supplementary information and calibration values. This datafile completely characterized the solidification event along the visible surface of the specimen and was suitable for input to the iterative solidification model.

3.3.2.4 Pixel Intensity Profiling

One additional method of analysis which helped characterize the double-recalescence phenomenon in Fe-Ni was pixel intensity profiling. In this technique, an IPLab script stepped through a series of solidification images and sequentially recorded the average pixel intensity observed in a very small (2x2 pixel) portion of each image. The result was a time-intensity profile similar to the output of a 40.5 kHz wide-band pyrometer tightly-focused on a ~0.16 mm square portion of the specimen. No attempt was made to calibrate the observed intensities to true temperature in this work, since experimental conditions varied greatly from specimen to specimen; however, the profiles were very useful in establishing relative temperatures and defining successive stages in solidification.

3.4 Solidification Model

3.4.1 Assumptions

An original computer-based solidification model (see Appendix A4) was used to calculate dendrite growth velocity within the melt from the observed surface propagation. The primary assumption underlying the model was that the solid phase grew spherically outward at a constant rate from a single nucleation point in an infinite undercooled melt; observed surface interfaces reflected the intersection of the expanding spheres with the physical boundaries of the ingot (Figure 3.5).

It was expected that this assumption might begin to break down either at very small undercoolings or else during the latter portion of the solidification event, when remnant undercooled liquid may become significantly reheated by the neighboring solid. Shrinkage effects might also distort the assumed geometry of the specimen.

It was assumed in addition that solidification proceeded quasi-adiabatically in the undercooled melt, and that the effects of the quartz enclosure on solidification kinetics were negligible. In this manner, the material at the quartz/metal interface could be assumed to be representative of the material in the bulk of the melt. This is another set of assumptions which will admittedly break down at very small undercoolings, when time scales increase and heat flow begins to become an issue.

3.4.2 Operation

Operating under the constant-velocity assumption, the computer model first input all the (x,y) points from a given interface datafile and assigned a relative time value (relative to the acquisition time of the first video frame) to each point. The relative time consisted of an integral multiple of T_f plus a fractional value of T_f that varied with the screen location of the interface pixel, where T_f was the acquisition time for a single video frame. The fractional portion is required because the pixel elements are not scanned simultaneously, but instead sampled by four horizontal passes of a 1x16 pixel vertical raster block (Figure 3.6; see also effect on appearance of image in Figure 4.3). As a result, the relative time value for a given pixel (x_i, y_i) in frame N is expressed as

$$t_{rel} = T_f \left\{ N + \frac{1}{4}\left[floor\left(\frac{y_i}{16}\right) + \frac{x_i}{64} \right] \right\} \tag{3.3}$$

Given the dimensions of the cylindrical ingot and a calibration value (mm/pixel), the model was also able to assign a z value to each (x,y) pair, thus defining each interfacial point completely in space and time.

At this stage, there remained two unknowns: the exact coordinate of the nucleation site and the period of time which had elapsed between nucleation and the first video frame of the solidification event. Given any possible nucleation point (x_n, y_n, z_n) and time delay (t_n) the program was able to calculate a consequent average velocity for each frame based upon the location of its interface points (x_i, y_i, z_i):

$$\overline{V_f} = \frac{\sum_{1}^{P} V_i}{P} \quad \text{for } V_i = \frac{D_i}{t_i}, \tag{3.4}$$

where $D_i = \sqrt{(x_i - x_n)^2 + (y_i - y_n)^2 + (z_i - z_n)^2}$,

$t_i = t_{rel} + t_n$,

and P is the total number of interface points evaluated.

The standard deviation of a group of average frame velocities, normalized over the average overall velocity, represented the degree of validity associated with the assumed nucleation position and time. The model proceeded to iterate over both position and time until it reached a local minimum in standard deviation, and the resulting solution was assumed to reflect the actual nucleation point of the solidification event.

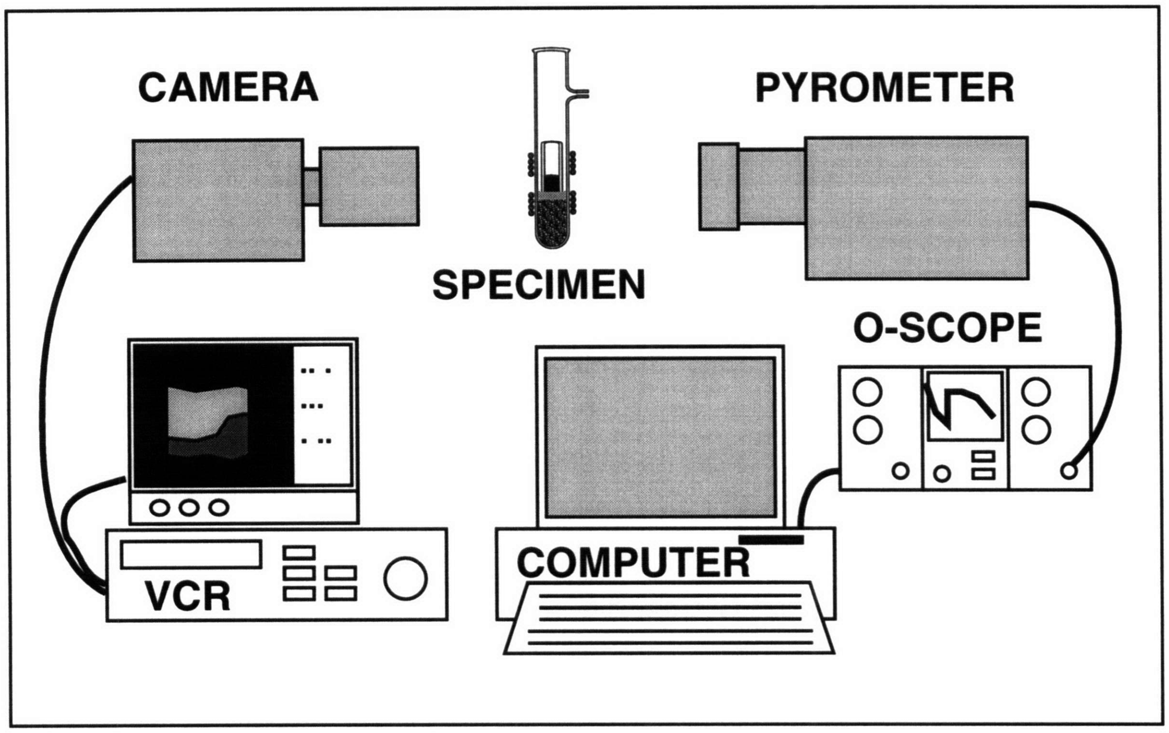

Figure 3.1: Schematic of apparatus used for pyrometric data and image acquisition.

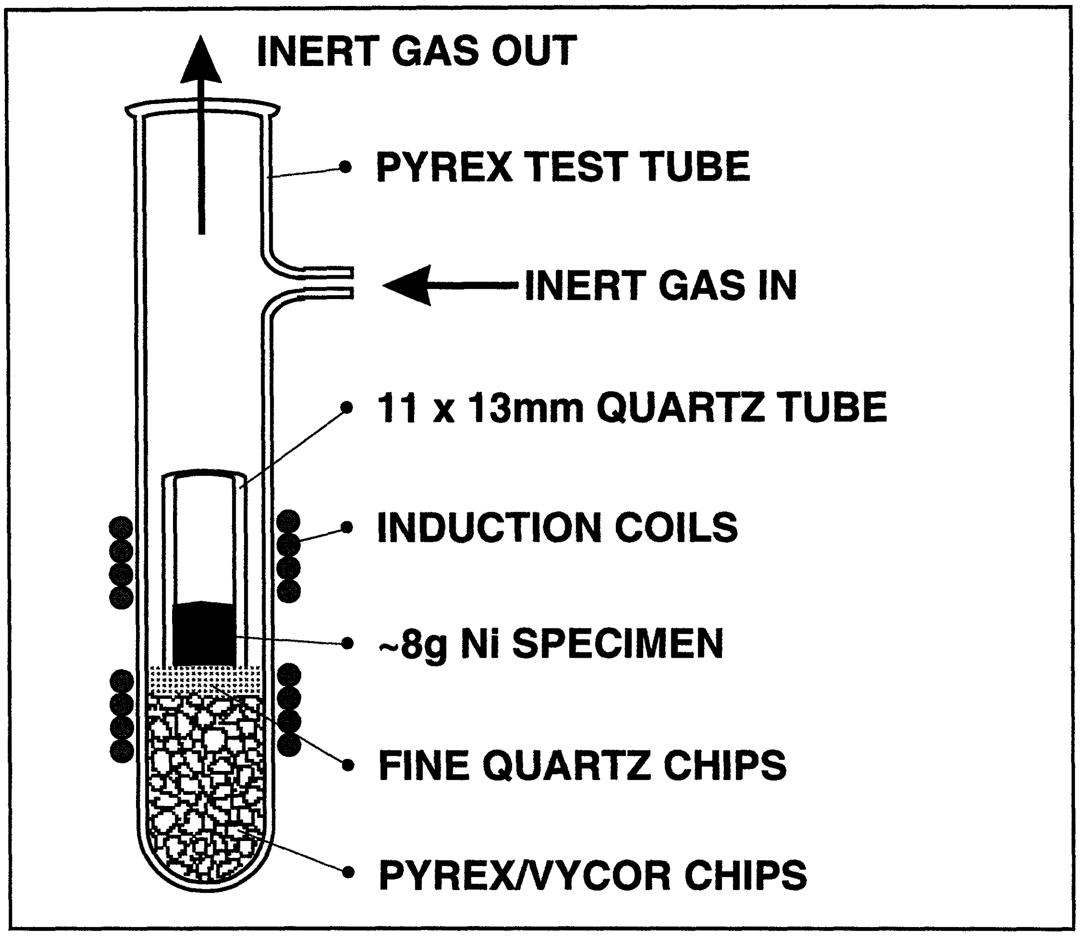

Figure 3.2: Detail of experimental setup, showing specimen position and geometry during processing.

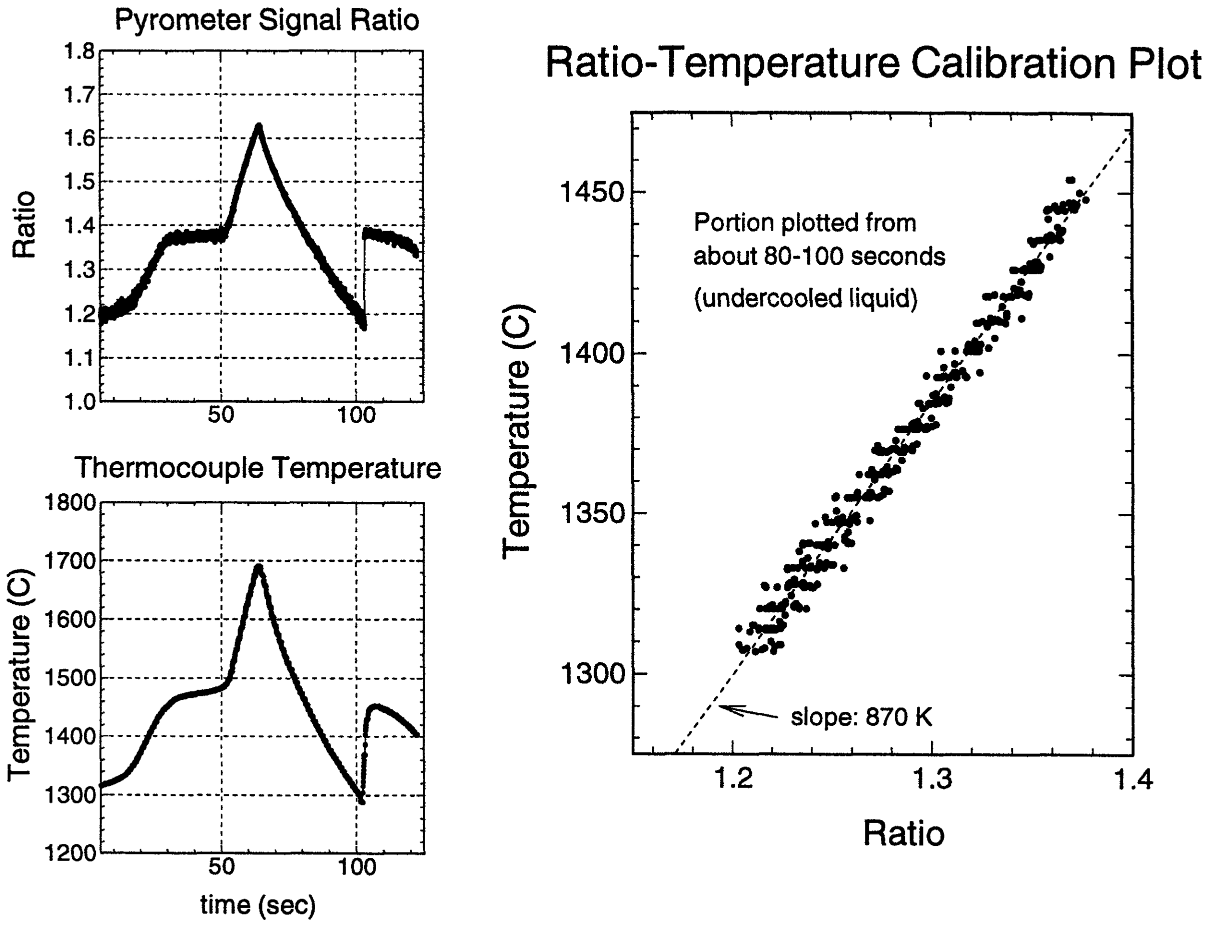

Figure 3.3: Typical ratio-temperature curve obtained during slope-calibration testing for pure Ni. Included at left are the time-ratio and time-temperature data for the specimen which yielded the ratio-temperature plot. In this case, a best-fit linear analysis of the data gives a ratio-temperature slope of 880 K.

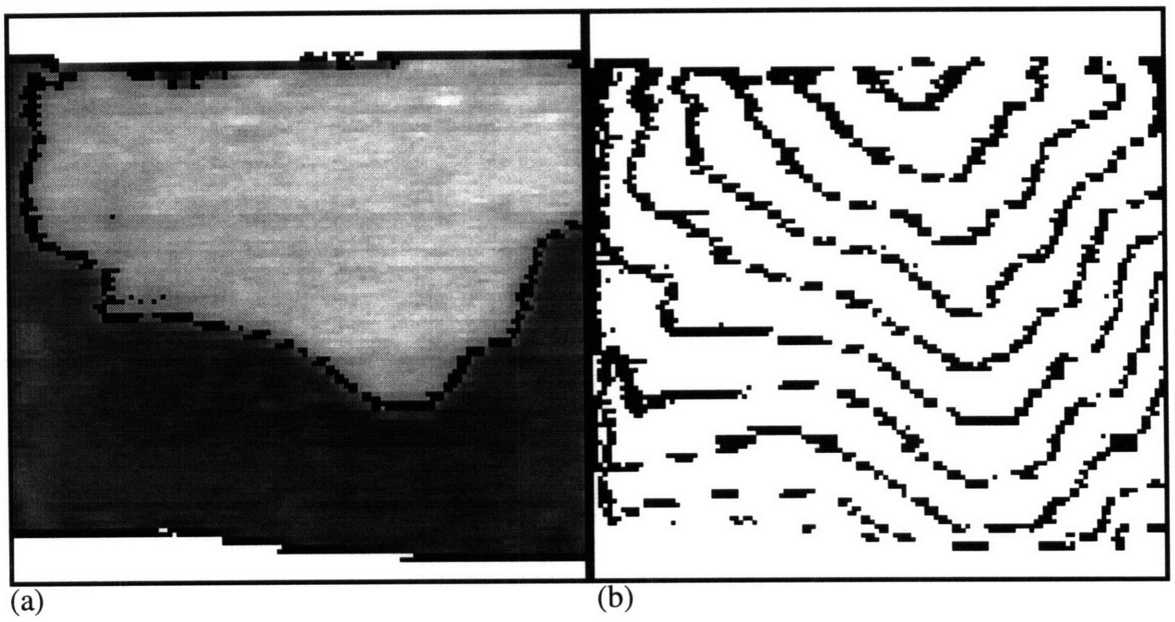
(a) (b)

Figure 3.4: Determination of solidification interface. At left is single image from recalescence event with computer-defined interface shown in black and newly-formed solid bright at top. At right is complete sequence of interfaces from same recalescence event.

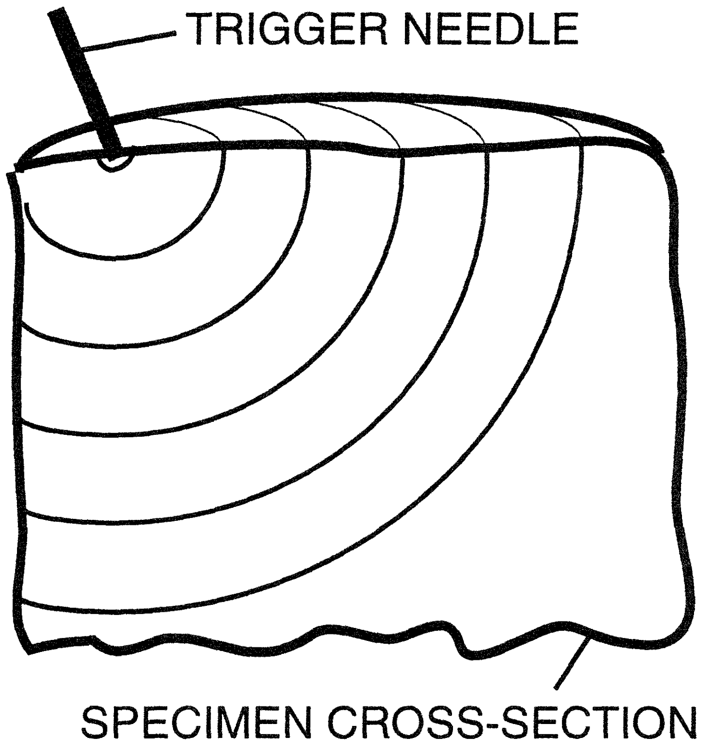

Figure 3.5: Cross-section of solidifying specimen, with successive model interfaces shown at constant time intervals following nucleation.

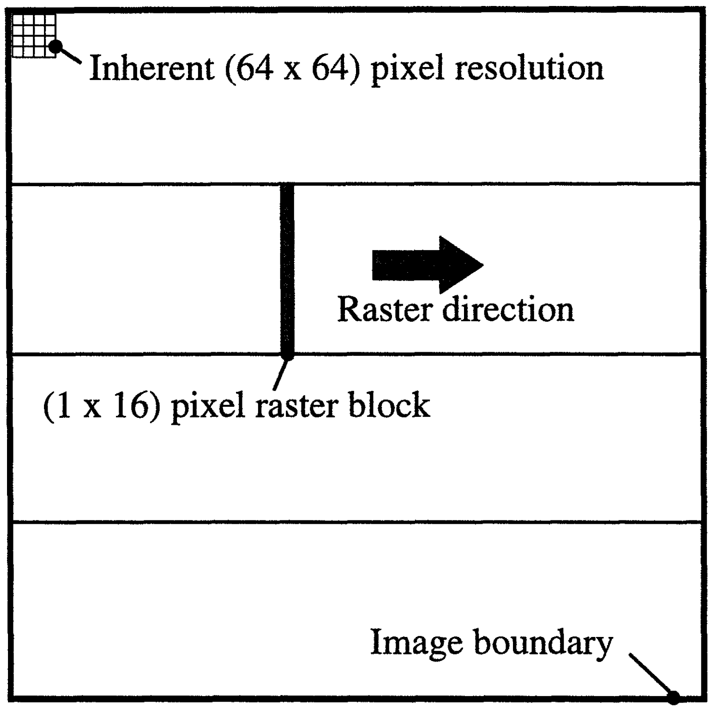
Inherent (64 x 64) pixel resolution
Raster direction
(1 x 16) pixel raster block
Image boundary

Figure 3.6: Raster motion during image acquisition at 40,500 frames/sec. Raster block begins at top left and scans from left to right, top to bottom in four passes, reaching bottom right after slightly less than 1/40,500 sec.

4. Results and Analysis

4.1 General Observations

The high-speed optical investigation produced thousands of images, with most recalescence events comprising between 5 and 100 video frames. A comparison of the pure Ni and Fe-Ni images revealed several trends which held for all compositions.

4.1.1 Multiple Nucleation

Multiple simultaneous nucleation was never conclusively observed at any composition, despite detailed post-experimental review of both spontaneously-nucleated and needle-stimulated specimens. One or two apparent multiple nucleation events could instead be attributed to propagation from a single event on the back side of the specimen, during which the jagged leading edge of the dendrite array intersected with various unconnected points on the front (visible) surface (Figure 4.1). A particularly striking example of this phenomenon occurred during related investigation of Ni-25 wt.% Sn alloys, and is shown in Figure 4.2, in which the "apparent" new grains all have the same crystallographic orientation as the parent grain.

These results do not imply that multiple nucleation can never occur. Instead, the video evidence is a strong indication that growth kinetics dominate nucleation kinetics in this sort of experiment. Moreover, the growing solid does not appear to have any influence on nucleation in the undercooled material ahead of the solid/liquid interface.

4.1.2 Interfacial Morphology

The solidification interface was also observed at every composition to exhibit two distinct interfacial morphologies. The envelope of solid dendrites appeared macroscopically jagged at lower undercoolings, but smooth and convex to the liquid at higher undercoolings, a phenomenon previously reported by Colligan and Bayles [3] in pure Ni. This behavior will be further discussed in later sections devoted to the individual specimen compositions.

4.2 Lens and Wetting Effects

A combination of two factors caused the diameter of the molten specimen to appear larger than the 11 mm inside diameter of the quartz tubing that housed it. First, a lens effect arose near the left- and right-hand walls of the tube when its contents were viewed from the side, due to the increased thickness of curved quartz between the viewer and the specimen. A secondary and more dominant effect developed when a hot specimen became molten and began to wet the inside wall of the tube; there was an immediate and significant apparent increase in sample diameter.

After some experimentation, it was found that the best way to quantify the above phenomena was to introduce an immersion oil-soaked section of millimeter-rule graph paper onto the inner surface of the quartz tube in place of a specimen, and photograph the setup with backlighting (Figure 4.4). The immersion oil enabled the graph paper to wet the quartz in much the same way as the specimen, and the wetting and lens effects could be calculated by examining the apparent spacing of the vertical rules of the paper across the width of the image.

The unaffected, 1 mm-spaced horizontal rules in Figure 4.4 provided a calibration value in terms of pixels/mm which could later be used to relate pixel distances to true distances when calculating velocities. The value was also fed into a small optimization routine, along with the positions of the vertical rules in the calibration image. The routine calculated that the data was best modeled by assuming the inside radius of the quartz tube was approximately 14% larger than its physical value. Figure 4.5 shows black model rule lines ($R_{apparent}$ = 6.25 mm) superimposed over the gray observed rule lines (R_{actual} = 5.5 mm) traced from Figure 4.4, demonstrating excellent agreement over almost the entire width of the tube. In practice, data was never taken from less than about 1 mm away from the sides of the tube, in order to avoid the large error inherent to such measurements.

4.3 Solidification Model Behavior

4.3.1 Iterative Convergence

As previously discussed, operation of the solidification model consisted of an iterative search for an optimum nucleation point, assuming constant-velocity growth of solid. The variables of iteration were, in order, t_n (time delay), z_n, x_n, and y_n, where the positive z axis extended toward the camera. The model's ability to converge upon an exact solution depended greatly upon the true location of the nucleation point; complete convergence was far more likely if the nucleation point lay on the front surface than if it lay on the top, bottom, back, or interior of the specimen.

4.3.1.1 Convergence for All Variables

Often, a spontaneous nucleation site appeared to be located on the front surface (metal/quartz interface) of the ingot. In some cases, the nucleant was clear upon review to have been an oxide particle or a scratch in the quartz tube; in other cases the only evidence was an initial solidification interface with a very small radius of curvature (Figure 4.3). For almost all of these "front-surface" nucleation events, the computer model verified the apparent nucleation site by iterating to convergence in all variables. The predicted nucleation point typically matched the observed point within approximately 0.5 mm, and the delay time between nucleation and first frame acquisition was between 0 and 1 frames—a required condition for a visible nucleation site.

It should be noted that the model was also able to converge for all variables in some cases where the nucleation point did not lie on the front surface. For instance, there were a number of examples of needle-stimulated top-surface nucleation that were correctly predicted by the solidification model. More generally, in all cases of complete convergence, the model's optimum (x_n, y_n, z_n) coordinate was found to lie within a distance (.05*R) of the ingot's surface, where R represents either the radius (cylindrical surface nucleation) or the half-height (top/bottom surface nucleation) of the ingot. This behavior was interpreted as strong evidence that nucleation occurred at a metal/quartz or metal/gas interface in all cases in which it was not manually stimulated at the top surface by a needle.

4.3.1.2 Convergence on Cylindrical Surface

In many other cases, the model was unable to reach an exact solution for t_n, z_n, x_n, and y_n. This often occurred when the nucleation point lay unseen on a hidden surface and occasionally even when the point appeared to be located on the front surface of the specimen. In many of these cases, a solution was possible if the nucleation site was constrained to lie on the cylindrical surface of the specimen and the model iterated over only t_n, x_n, and y_n, with z_n determined by the other two spatial coordinates and the geometry of the ingot.

4.3.1.3 Time-bounded Convergence

A final general case occurred when no convergence was possible for any set of initial conditions because the optimum t_n value increased without bound. As expected, the model often behaved in this manner when the true nucleation point was far removed from the front surface of the specimen and the camera had not recorded the initial portion of the solidification event.

Fortunately, it was often possible to impose constant upper- and lower-bounds on t_n in these cases and solve for the optimum (x_n, y_n, z_n) coordinates and solidification velocities given the bounding t_n values. This was a particularly useful technique in the case of non-convergent needle-stimulated events, because the y_n coordinate was known to lie within a small distance of the (visible) top edge of the specimen. The strategy in these cases consisted of iterating with various fixed values of t_n until two arguments were found which resulted in sensible bounding values of y_n—typically, two pixels above and two pixels below the observed top edge of the specimen. The corresponding velocity values were also taken to be upper- and lower-bounds, and they did not often differ by more than five percent.

In all cases where bounded solution sets showed a variation of more than 10% of the average solution, the solutions were removed from further consideration. This result typically followed when the radius of curvature of the first observed solidification interface was unusually large, supporting the notion that the camera had only observed the final portion of the solidification event and the model has broken down.

4.4 Pure Ni Results

4.4.1 Solidification Velocity

Presented in Figure 4.6 are the solidification velocity values, plotted against undercooling, obtained from the computer model used in this work. Comparison with dendrite growth theory will be made using the so-called "LKT" model of Lipton, et al. [9], who adapted the Ivantsov dendrite growth model [25] using a marginal stability analysis (see Appendix A1 for LKT development). Shown in Figure 4.6 are both the unmodified LKT prediction for pure nickel and the LKT prediction assuming a linear kinetic undercooling parameter μ = 0.40 m/sK. Clearly, there is fair agreement between experiment and theory up to some critical undercooling ΔT^*, where 150 K < ΔT^* < 180 K in this case. Above ΔT^*, it is difficult to characterize exactly the relationship between undercooling and velocity, other than to observe that solidification velocity continues to increase. These results are in general accord with all of the prior work referenced, with the exception of Hofmeister and Bassler.

4.4.2 Interfacial Morphology

A morphological analysis of the solidification interface at various undercoolings was carried out in the following manner: any specimen whose images exhibited discontinuous interface curvature not attributable to the camera raster pattern was classified as "jagged;" specimens showing only continuous interface curvature were called "smooth." This analysis revealed a transition undercooling range for Ni of 160-170 K, above which all solidification envelopes were smooth and below which all envelopes were jagged (Figure 4.7). This range is slightly above the critical range for morphological transition previously proposed by Colligan and Bayles. The temperature range does correspond well to the "critical undercooling" values observed on undercooling-velocity plots both in the current work (150-180°C) and by prior investigators. Furthermore, a morphological transition ΔT^* of 160-170°C is also approximately equal to the microstructural ΔT^* observed by Walker, Colligan, Schleip, and others in pure Ni.

The present work strongly suggests that the morphological transition in solid/liquid interfacial shape and the observed transitions in velocity and microstructure are coupled phenomena resulting from the same intrinsic shift in solidification mechanism. However, the physical mechanism responsible for the transitions has yet to be satisfactorily explained. Various hypotheses point toward either a kinetic attachment limitation [6], the onset of dendrite fragmentation by remelting [26], dynamic nucleation via shrinkage-induced cavitation in the undercooled melt [27], or other fluid-flow phenomena.

4.5 Fe-Ni Alloy Results

4.5.1 Pyrometric Thermal Profiles

Two typical high-speed thermal profiles from Fe-10 wt.% Ni specimens are shown in Figure 4.8. The upper profile represents a specimen which was not observed in the video record to have undergone double recalescence; instead, the equilibrium FCC phase solidified directly from the undercooled melt. This profile exhibits a clear thermal peak at the tail end of the rapid transition, approximately 10 K in amplitude and spanning about 0.1 ms. A similar phenomenon was observed by Piccone [28] during his studies of hyper-peritectic Fe-Ni, Ni-Sn, and Fe-Co alloys using the same apparatus. He attributed the peak to superheating and subsequent remelting of a portion of the newly-formed dendrite array.

The profile in the lower half of Figure 4.8 shows a specimen which did exhibit double-recalescence behavior in the video record. This behavior is clearly reflected in the thermal profile by a preliminary plateau at approximately 1495 K (near the metastable liquidus temperature) lasting for about 1 ms, followed by a secondary rise in temperature to near the equilibrium FCC liquidus with no characteristic thermal peak.

Unfortunately, not every specimen which underwent double recalescence visible by the camera displayed such an unambiguous pyrometric profile. Many such specimens instead showed little more than a change in inflection, and sometimes no hint at all of double recalescence in the pyrometric record (Figure 4.9). Failure to detect multiple thermal transitions was probably a result of the pyrometer's relatively low spatial resolution; the view field of the instrument was between 6 and 10 mm in diameter for the duration of the work, and the output voltage reflected the average light intensity of the entire field. It is not surprising that transitions narrowly separated in distance or time were not crisply detected by this instrument.

4.5.2 Pixel Intensity Profiles

The pixel intensity profiling outlined in section 3.3.2 was a much more reliable method of detecting and characterizing double recalescence events, despite its limited temporal resolution (40,500 Hz). In general, double recalescences were easily identified by a secondary intensity transition either on videotape or in an animation of the digitized images from a single event. Occasionally, the growth of FCC material took the shape of a nearly continuous interface which lagged the BCC solidification interface by a short distance. More often, however, the FCC transition was more diffuse, appearing to nucleate and grow from multiple points behind the BCC interface. In either case, the transition was predictable enough so that, at any given point on a single specimen, there appeared to be a roughly constant delay time between the passage of the BCC interface and the subsequent transformation to brighter FCC material.

Due to the consistent behavior of the transformation, the choice of the target area for pixel intensity profiling was relatively unimportant; an arbitrary 2x2 pixel area free of oxidation or quartz discoloration was chosen for each event. The resulting time-intensity profile was generally a much more sensitive record than the corresponding pyrometer profile (Figure 4.10), giving a clearer indication of recalescence behavior and delay times between transitions, if not accurate temperature values.

4.5.3 Recalescence Behavior

4.5.3.1 Characterization

Recalescence characterization was accomplished through simultaneous examination of the pixel intensity profile, the high- and low-speed thermal profiles, and the video record. Double recalescences which were clear on the video record were almost always verified by a double-plateau in the pixel intensity profile and were considered clear instances of initial BCC phase formation. Apparent single recalescences on the video record were typically supported by single-plateau pixel intensity profiles; in addition, the high-speed thermal profiles in these cases generally exhibited a superheat peak on recalescence of about 5-15 K which was absent in the case of double recalescence.

The video record was no help in the identification of secondary recalescence events which lagged the initial solidification by more than about 0.2 seconds, because video data was not recorded beyond this time. However, the low-speed thermal record was assumed to be sensitive enough to identify these delayed transitions. In cases where the recalescence behavior was truly unclear, the data points were omitted from plots and calculations. Figure 4.11 reveals the recalescence behavior of the various Fe-Ni specimens as a function of composition and nucleation temperature. Filled symbols indicate that primary BCC solidification and double recalescence occurred; outlined symbols indicate primary FCC solidification. Briefly ignoring the significance of symbol shape, one can interpret Figure 4.11 as a primary solidification map valid for all quartz-fluxed Fe-Ni specimens of similar aspect ratio weighing approximately 8 g and exhibiting the same cooling rate (phase selection depends upon the thermal history of the undercooled liquid as well as the presence of heterogeneous nucleants [17]).

This work suggests that there exists a window of primary BCC solidification in hyperperitectic Fe-Ni bounded at smaller undercoolings and very large undercoolings by a transition to primary FCC solidification. The window appears to span about 150 K at 10 wt.% Ni, tapers to less than 120 K at 15 wt.% Ni, and has disappeared entirely at compositions of 20 wt.% Ni and higher.

4.5.3.2 Phase Selection via Induced Nucleation

As noted earlier, both Fe and Ni trigger needles were used at various times to induce heterogeneous nucleation at lesser undercoolings, with the idea that the Fe (BCC from room temperature to 1185 K and from 1667-1811 K) might induce primary BCC solidification and the Ni (FCC through the entire range) might produce instead the FCC phase. As Figure 4.11 clearly reveals, these efforts met with no success at all, with the exception of three points near 1350 K in the Fe-12 wt.% Ni material. In those three instances, FCC material formed from the Ni trigger on a section of the map which otherwise featured solely BCC solidification.

At all other compositions and temperatures, trigger needle structure seemed to have absolutely no effect on the determination of the primary solidification phase. Apparently, FCC Ni is simply not a sufficiently potent heterogeneous nucleant to have much of an effect on the kinetics of primary phase solidification in the compositions that were extensively studied. Perhaps a trend would have emerged if more data had been collected from the Fe-15 wt.% Ni alloy.

4.5.3.3 Delay Times in Double Recalescence

The delay time between recalescences was defined on the pixel intensity profile as the time between points of maximum curvature at the beginning of each transition. The resulting correspondence between undercooling and delay time for double recalescence is shown in Figure 4.12 for the compositions which most consistently exhibited that behavior.

Figure 4.12 reveals that average delay time drops with increasing undercooling in a roughly exponential manner. The data scatter may be an indication that there are several different mechanisms by which the FCC phase nucleates and grows into the mixture of undercooled liquid and BCC solid material. However, the data is in qualitative agreement with a theory that FCC formation requires the attainment of a critical local fraction solid, since the fraction solid immediately following recalescence can be expressed as:

$$f_s = \frac{\Delta T}{\Delta T_{hyp}}, \quad \text{where } \Delta T_{hyp} = \frac{\Delta H_f}{c_p} \tag{4.1}$$

FCC nucleation may result from the impingement of growing BCC dendrite arms during the equilibrium solidification period following initial recalescence.

4.5.4 Interfacial Morphology

A morphological analysis identical to that described in section 4.3.2 enabled the characterization of solidification interfaces as "smooth" or "jagged" for most compositions of Fe-Ni studied in this work. Figure 4.13 summarizes the results of the analysis.

Once again, there is a rather sharply-defined transition from jagged to smooth morphology at a critical undercooling; this ΔT^* is approximately 190 K for both Fe-10- and Fe-12 wt.% Ni. The exact value at higher Ni compositions cannot be precisely determined from the data, but it is evidently no more than 190 K, and not a great deal less. The range of 160-190 K previously identified for morphological transition in pure Ni would suggest that ΔT^* decreases only slightly, if at all, as the specimen grows richer in nickel content.

4.5.5 Solidification Velocity

Solidification velocities obtained from the computer model are given in Figure 4.14 for Fe-10- and Fe-12 wt.% Ni specimens which exhibited double recalescence (undercoolings are measured from the *metastable* liquidus temperature). The two sets of data are plotted on the same graph because LKT theory predicts a virtually identical undercooling-velocity curve for both compositions. Indeed, it is clear from the plot that the two data sets essentially overlap and are well-predicted by an LKT analysis for Fe-Ni using a kinetic parameter of 0.28 m/sK.

The most notable aspect of Figure 4.14 is the clear lack of any transition in functional dependence in the range $170\ K < \Delta T < 190\ K$. Instead of the downward shift observed in this work for Ni (Figure 4.6) and in the work of Barth, et al [21] for Fe-25- and Fe-30 wt.% Ni, the data continues to be well-described by LKT theory right up to undercoolings of nearly 250 K.

In light of the observed transition in interface morphology, the lack of any corresponding shift in solidification velocity is somewhat difficult to explain. One difference between these Fe-(10-12) wt.% Ni results and the contrasting pure Ni and prior Fe-Ni work is that the primary solidification phase in the current work is a BCC material. Perhaps crystallographic structure plays an important role in determining the interface kinetics of these rapidly-solidified materials. Unfortunately, there is insufficient data in the present work at the higher Ni concentrations to draw any conclusions regarding the solidification velocities observed there for FCC Fe-Ni.

A more likely explanation is that the velocity transition is not actually governed by a critical undercooling, but instead by a critical velocity which is comparable across materials. The transitions in the Barth work appear to begin somewhere above 30 m/s. The data in Figure 4.6 for pure Ni suggests that the critical velocity range corresponding to $150°C < \Delta T^* < 180°C$ is approximately $26\ m/s < V^* < 35\ m/s$; velocities of nearly 50 m/s were observed at the largest undercoolings. The maximum velocities observed for Fe-(10-12) wt.% Ni in this work were only 30-35 m/s. Thus, if the transition velocity for Fe-(10-12) wt.% Ni alloys is near the top of the range observed for pure Ni, it is not surprising that no transition was experimentally observed. However, regardless the explanation, it would seem that solidification velocity behavior and interface morphology are not strongly coupled in the metastable phase solidification of these Fe-Ni alloys, as seemed to be the case for pure Ni.

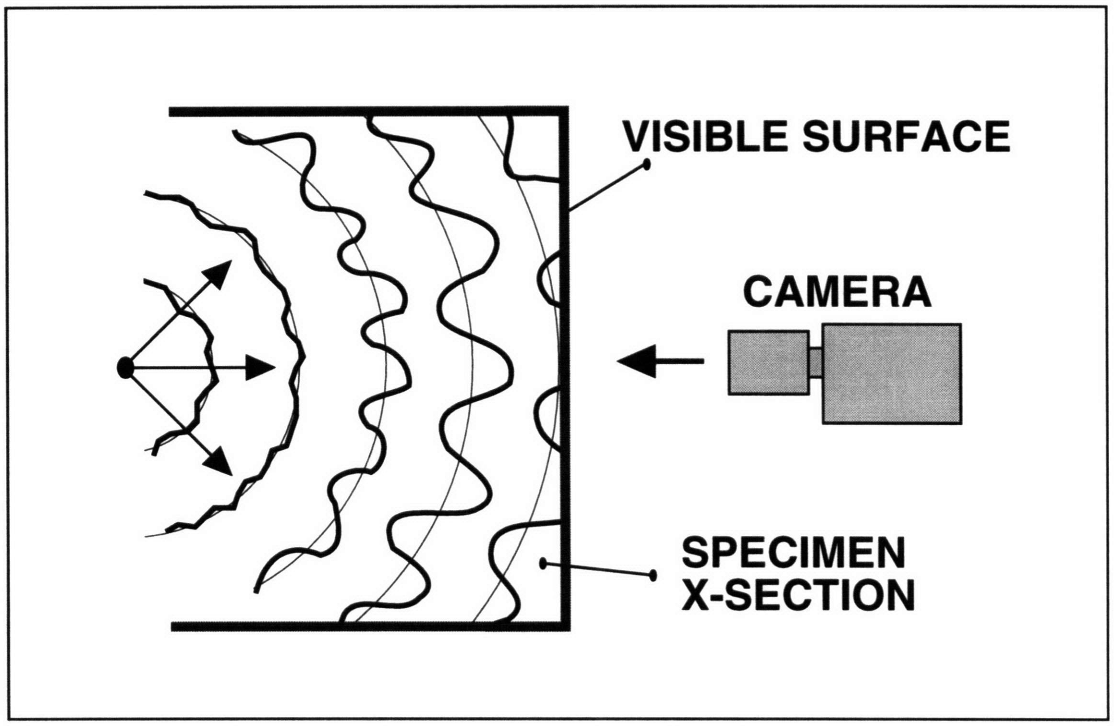

Figure 4.1: Schematic of apparent multiple nucleation scenario, observed experimentally in Figure 4.2.

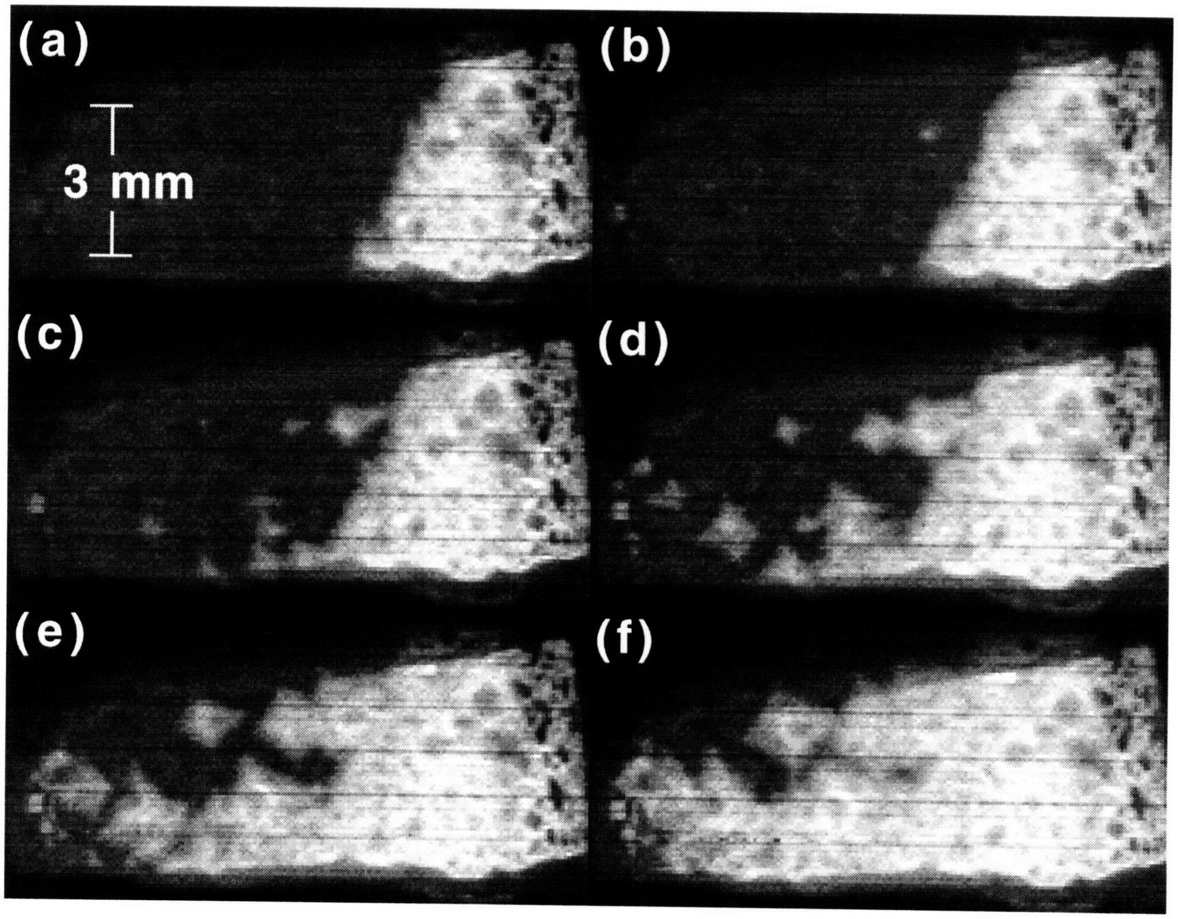

Figure 4.2: Example of solidification interface propagating from back to front, almost directly toward the camera lens. System is Ni-25%Sn, ~25 ms between images. Dendrite tips intersecting front surface appear as numerous expanding diamonds with the same axial orientation.

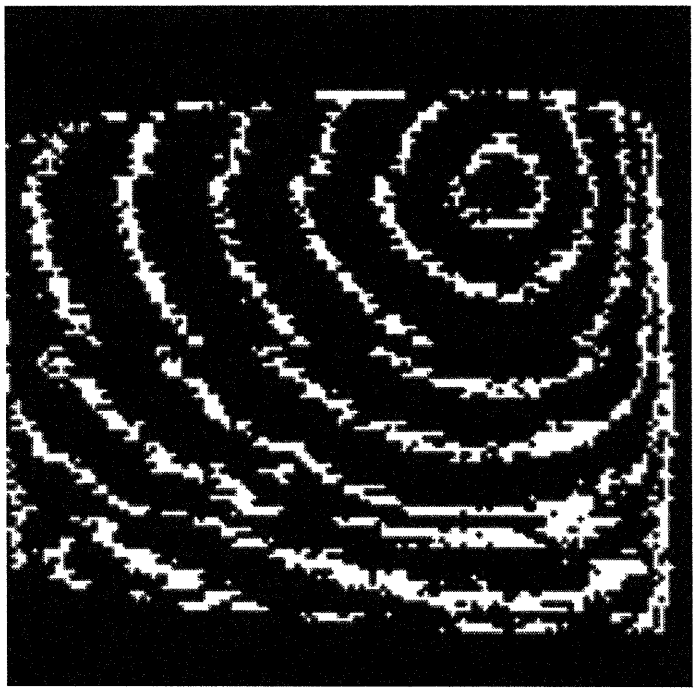

Figure 4.3: Composite interface image showing front-surface nucleation; initial interface at upper-right exhibits very small radius of curvature. Note also raster pattern effect at 1/4, 1/2, and 3/4 of the image height causing apparent discontinuities in interface curvature.

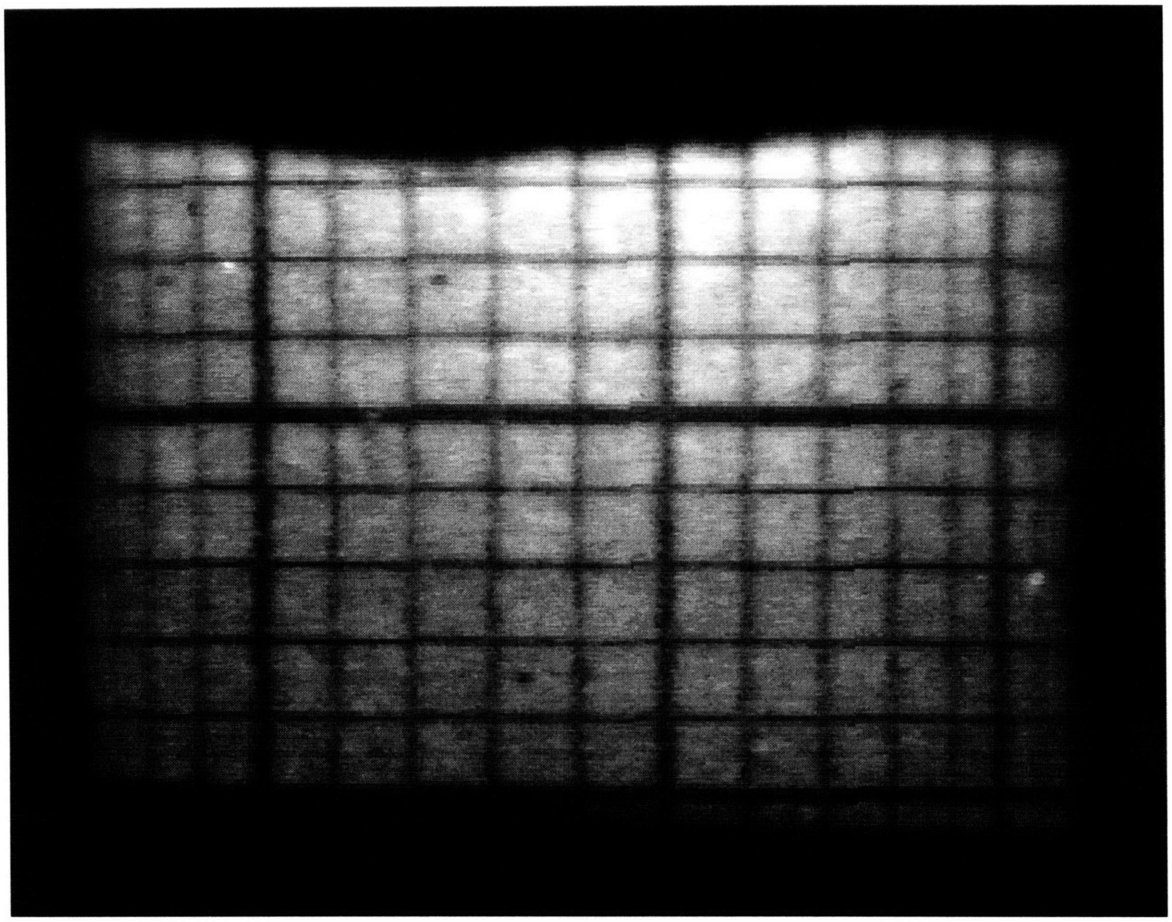

Figure 4.4: Calibration image of oil-soaked millimeter graph paper rolled inside 11 x 13 mm quartz tubing.

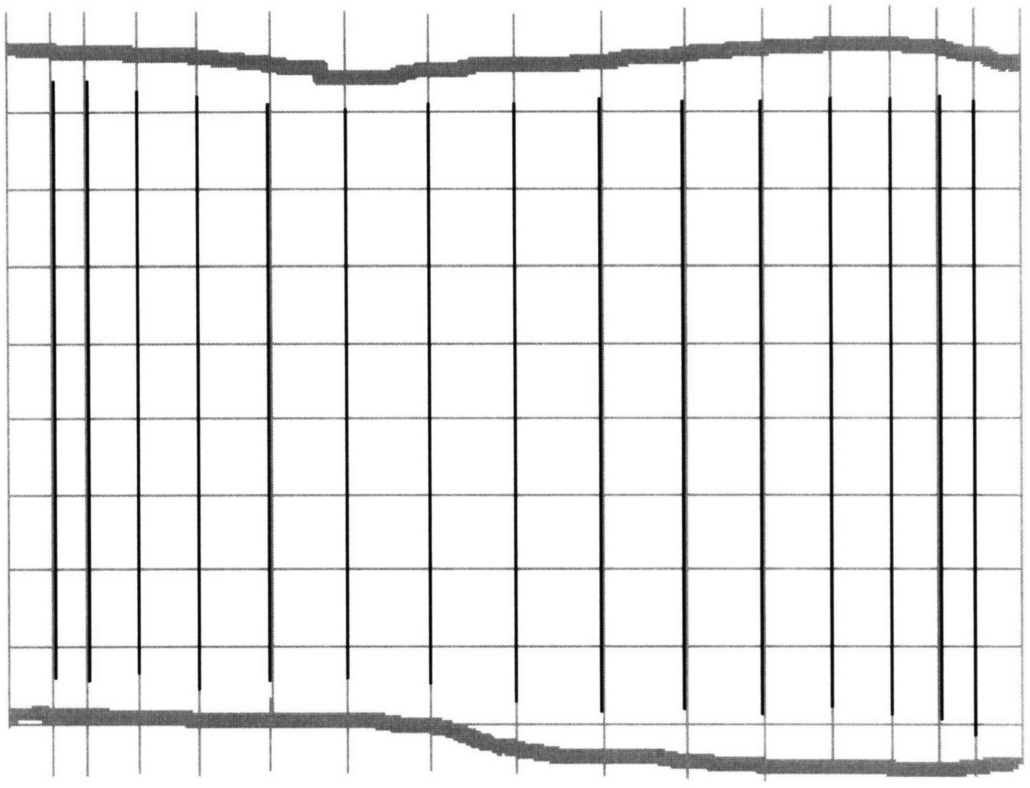

Figure 4.5: Model vertical rules (black) overlaying observed rules (gray, traced from Figure 4.4), verifying that the lens and wetting effects are well-modeled by assuming a 14% increase in specimen radius.

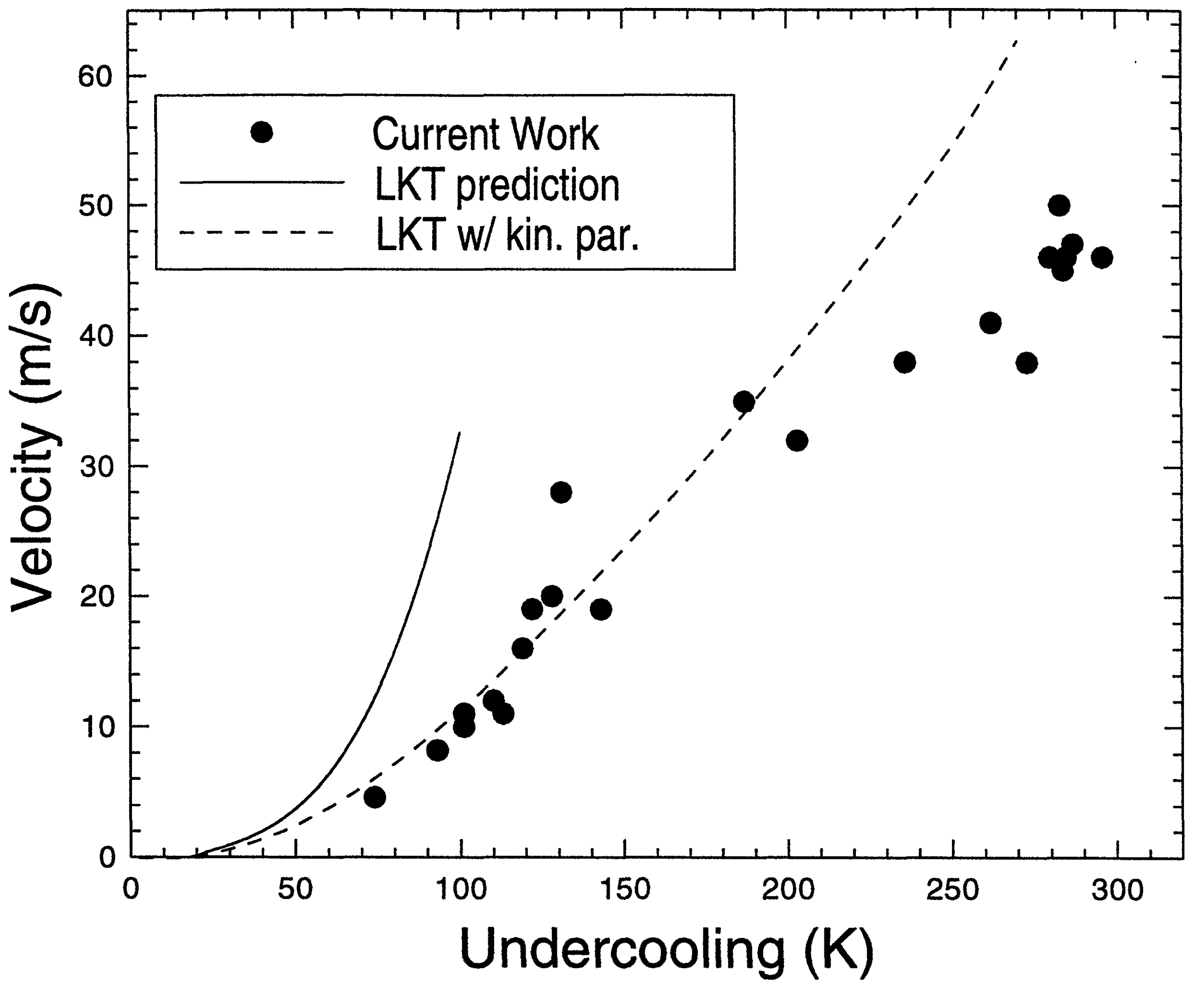

Figure 4.6: Solidification velocity plotted versus undercooling for pure Ni, as determined by this work. Also shown is LKT prediction with and without modification by a kinetic undercooling parameter.

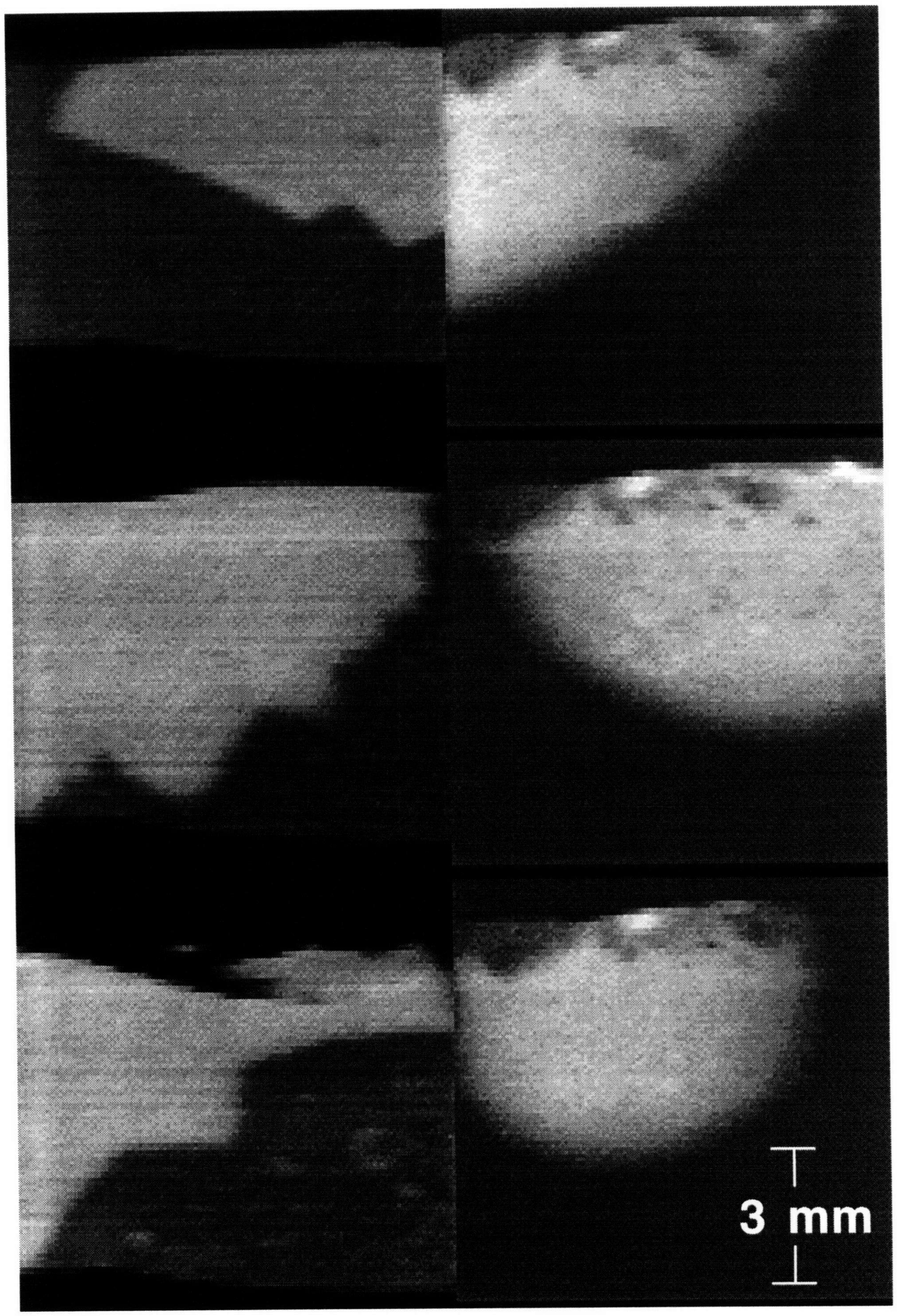

Figure 4.7: Interface morphology transition for pure Ni. Shown are typical interface morphologies observed at undercoolings less than the critical range of 160-170 K (left column) and greater than the critical range (right).

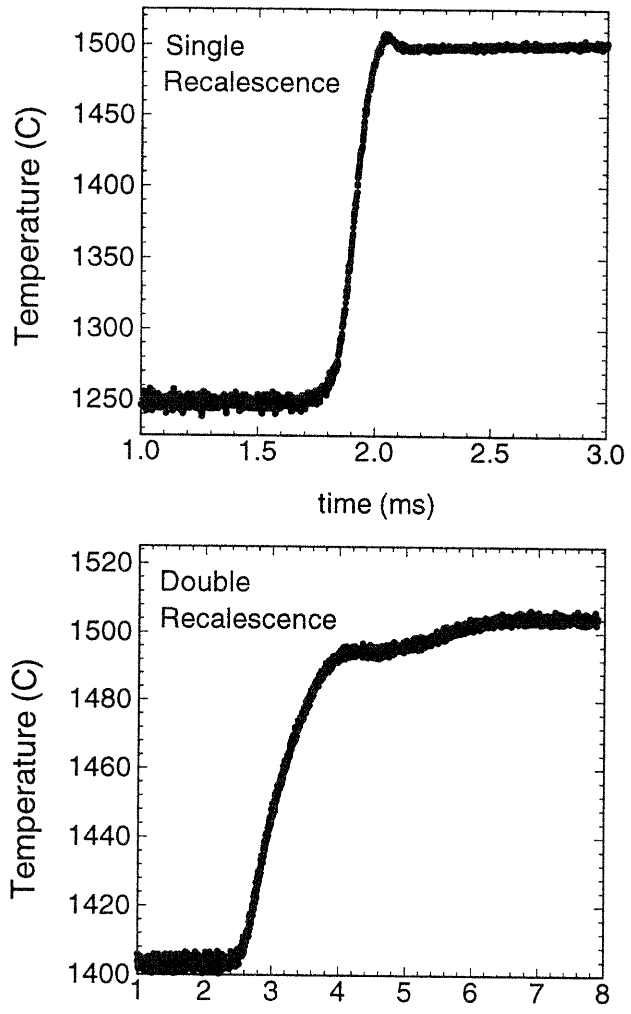

Figure 4.8: Typical thermal profiles for Fe-10 wt.% Ni specimens undergoing single recalescence only (top) and double recalescence (bottom). Note superheat peak at top and double-plateau behavior at bottom.

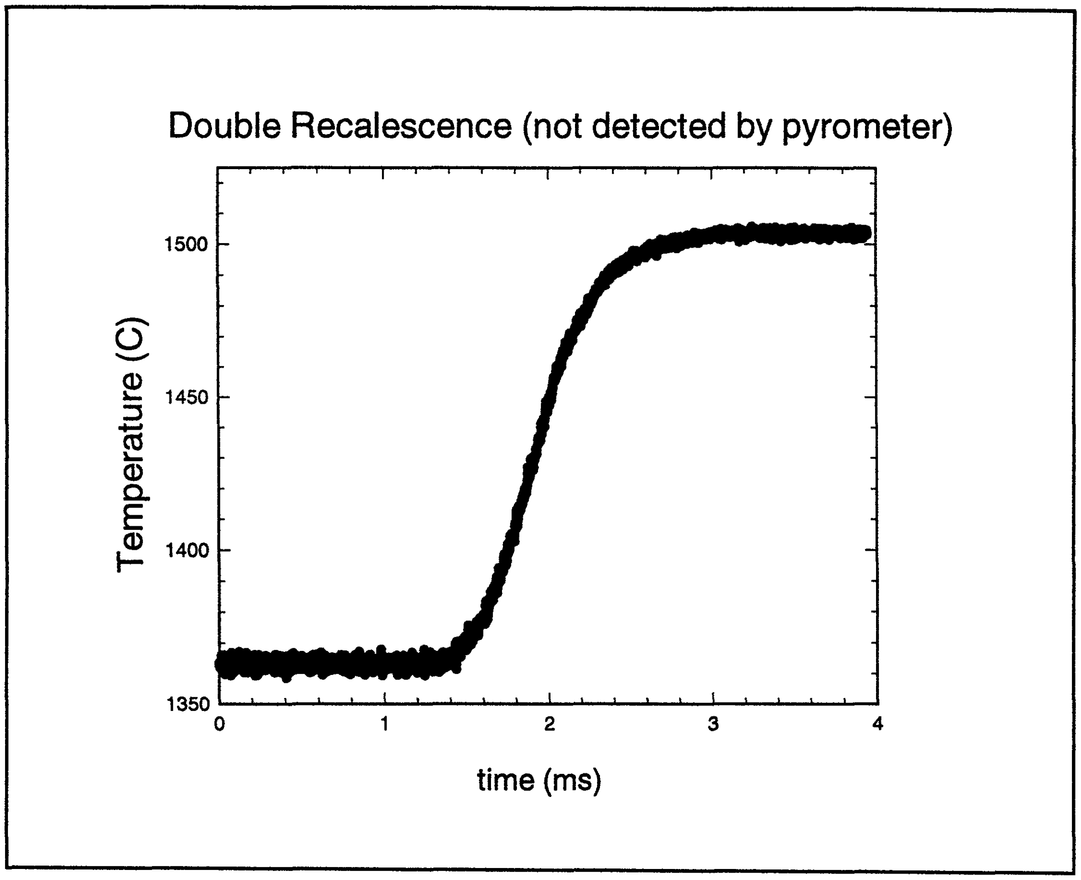

Figure 4.9: Pyrometer profile for Fe-10 wt.% Ni specimen which was revealed in the video record to have undergone double recalescence. No clear indication is present in the thermal record.

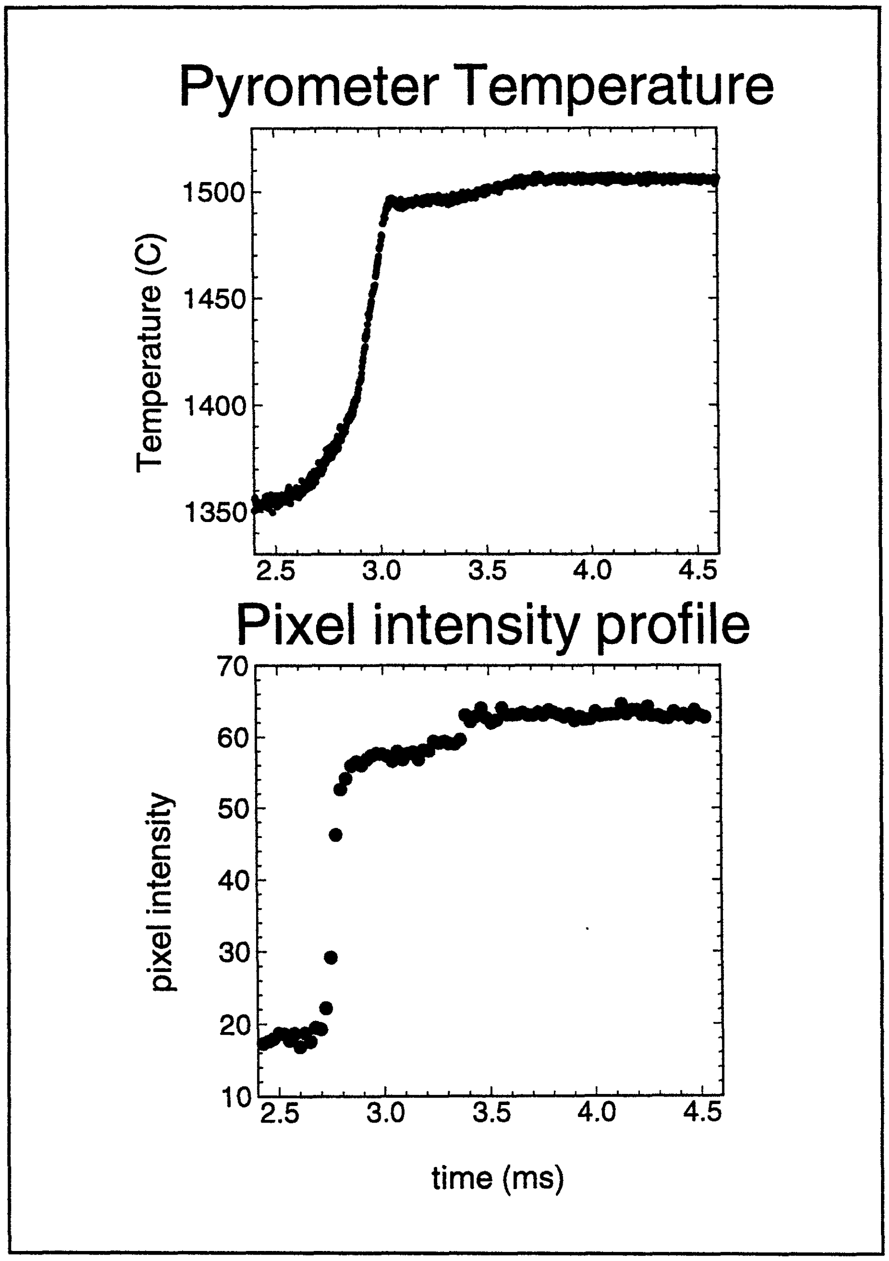

Figure 4.10: Comparison between pyrometer profile and pixel intensity profile for Fe-10 wt.% Ni specimen undergoing double recalescence. Details are much more sharply-defined in the pixel intensity profile.

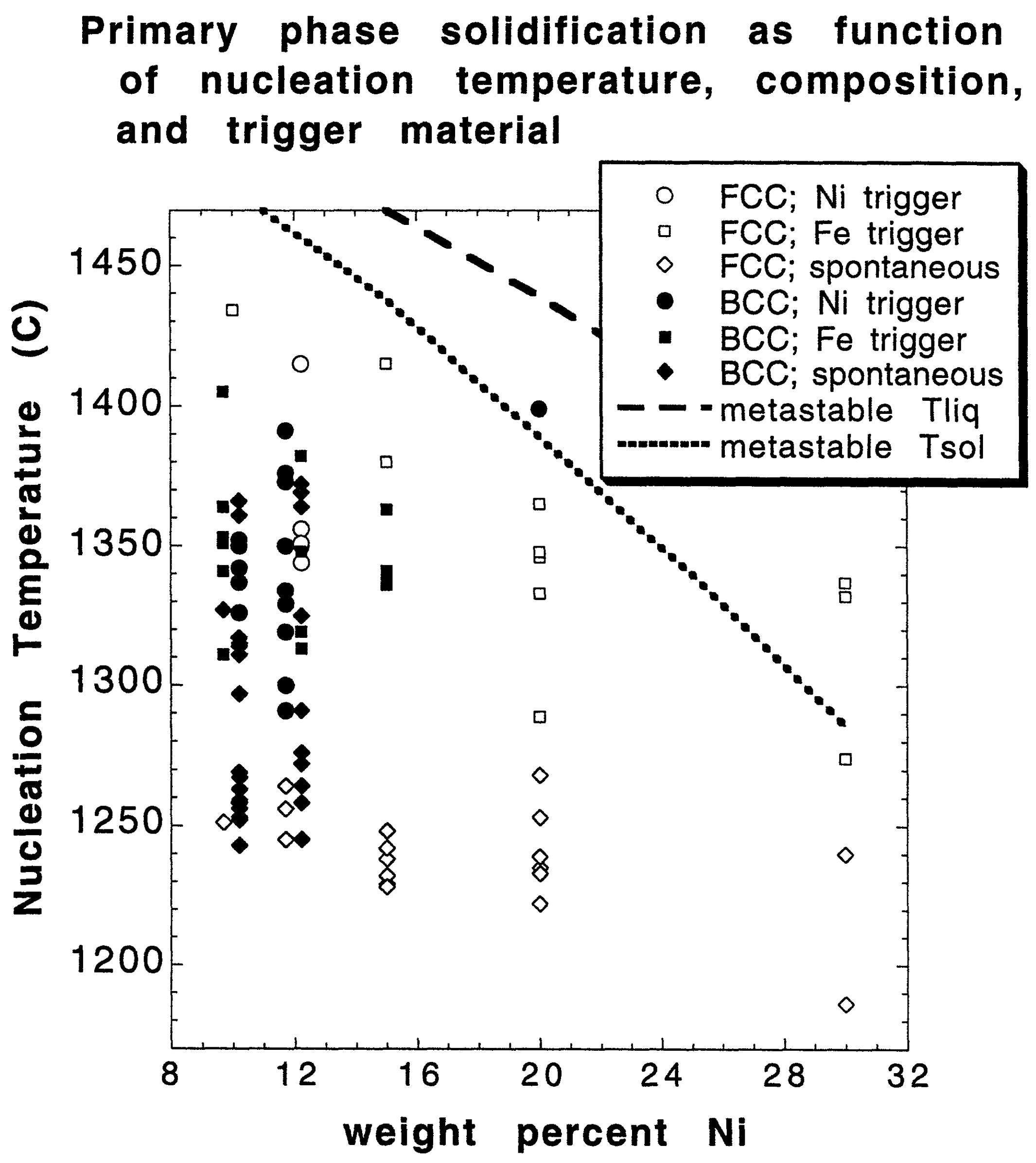

Figure 4.11: Solidification behavior in Fe-(10-30) wt.% Ni.

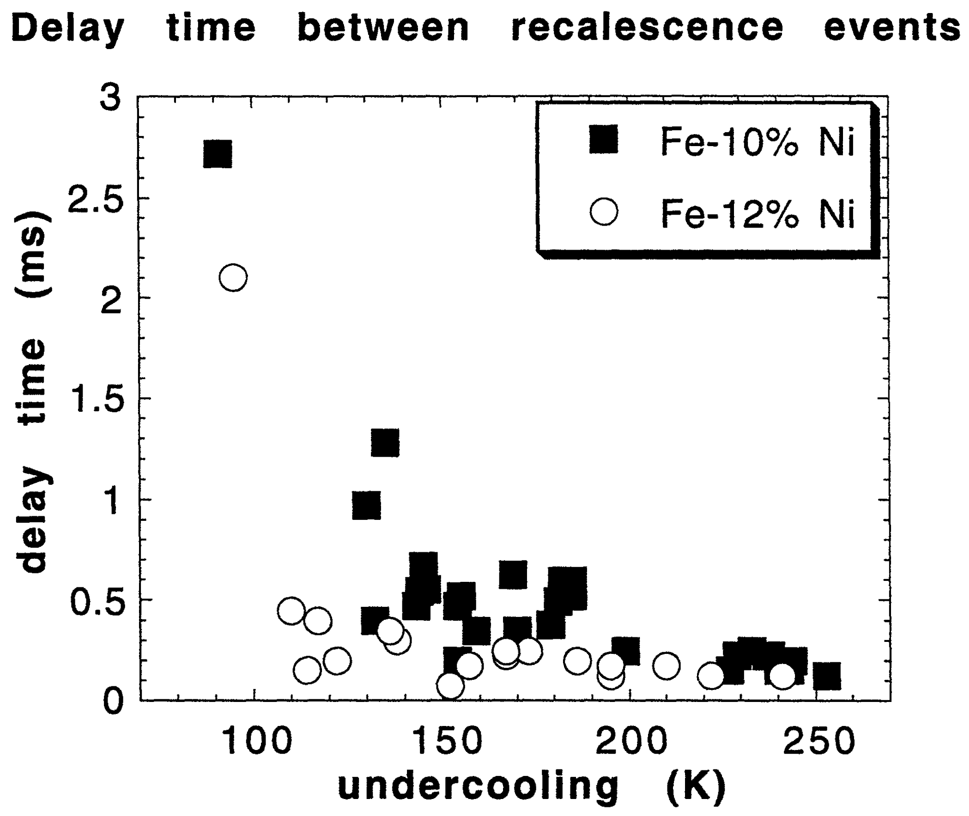

Figure 4.12: Delay time between recalescence events in Fe-10- and Fe-12 wt.% Ni specimens.

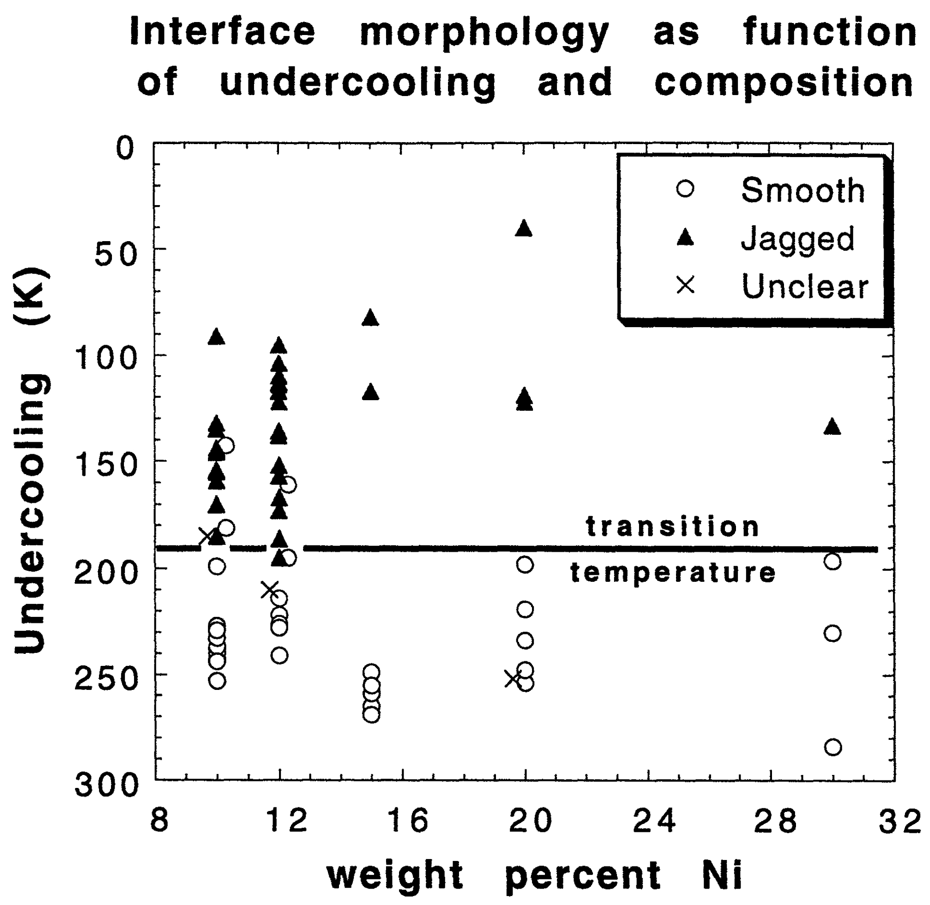

Figure 4.13: Interface morphology at various compositions and undercoolings. A transition from jagged to smooth structure appears at approximately 190 K undercooling.

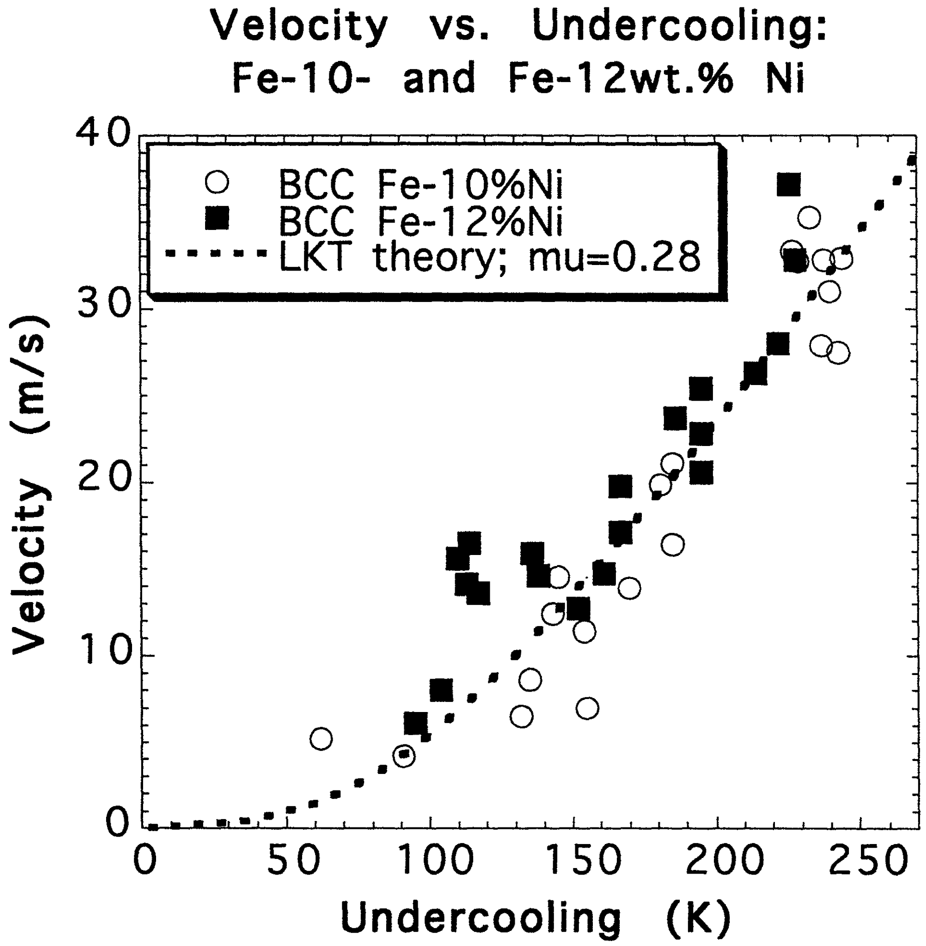

Figure 4.14: Solidification velocities for BCC Fe-10- and Fe-12 wt.% Ni.

5. Conclusions

5.1 General

1. The undercooling and subsequent solidification of high-purity Ni and hyperperitectic Fe-Ni specimens were accomplished, and the thermal front accompanying the solidification interface was directly observed through the use of an ultra-high-speed digital camera from Kodak. The degree of undercooling was evaluated for each heating cycle through the use of a two-color pyrometer and subsequent calibration experiments; a solidification velocity was calculated for each recorded solidification using digital image analysis and an original computer-based solidification model.

2. At no time were multiple nucleation sites observed during a single primary solidification event. Instead, at lesser undercoolings the solidification interface exhibited a "jagged" morphology, a phenomenon which may, at lower resolution, present the appearance of multiple nucleation sites

3. The computer-based solidification model was successful in determining solidification velocities with less data scatter than had previously been observed using pyrometric methods. The model can also be made to track the growth of subportions of the overall dendrite array, in the event that a study of growth velocities observed in different crystallographic directions is desired. Pyrometric analyses exhibit no such flexibility.

5.2 Pure Nickel

1. The relationship between solidification velocity and undercooling for pure Ni showed general agreement with many prior investigations; namely, there was an approximate power-law dependence at low undercoolings which is well-described by the LKT solidification model. Above a critical undercooling, $150\text{ K} < \Delta T^* < 180\text{ K}$, there appeared to be a partial shift in solidification mechanism, resulting in a secondary velocity dependence whose exact nature is unclear.

2. Observed at a similar critical undercooling ($160K < \Delta T^* < 170$ K) was a shift in interface morphology from "jagged" to macroscopically smooth, implying a change in controlling mechanism at the dendrite tips perhaps related to attachment kinetics, dendrite fragmentation, or fluid flow phenomena. The morphological and solidification velocity transitions appear to be coupled in pure Ni.

5.3 Iron-Nickel

1. Primary solidification of both the equilibrium FCC phase and the metastable BCC phase was observed in the hyperperitectic Fe-Ni specimens. The BCC phase appeared to be favored at lower Ni concentrations (10-12 wt.%), while FCC solidification was observed exclusively at 20 wt.% Ni and above.

2. A transition in interface morphology similar to that observed in pure Ni was also found in all compositions of Fe-Ni. The critical undercooling was well-defined around 190 K for Fe-10- and Fe-12 wt.% Ni specimens, and grew no larger at higher Ni concentrations.

3. Solidification velocities for BCC Fe-10- and Fe-12 wt.% Ni were plotted against undercooling and found to be nearly identical, as predicted by LKT theory. Agreement between experiment and theory was also excellent even at the highest undercoolings observed (250 K); there was no shift in velocity dependence as seen in pure Ni. A plausible explanation is that the shift actually occurs at a critical velocity, not a critical undercooling, but in any case it appears that interface morphology and velocity dependence are not coupled in Fe-Ni as they appeared to be in pure Ni.

6. Suggestions for Future Work

1. The structure of the solidification interface behind the dendrite tips has yet to be adequately understood. The high-speed images obtained in this work reveal an apparent thermal transition spanning a distance of several millimeters, but no attempt has been made here to develop a theoretical model for the behavior. One possible avenue for future investigation could be to use higher magnification to obtain high-resolution images of the apparent interface with the camera used in this work.

2. A theoretical basis is needed to explain the transitions in interface morphology and velocity dependence observed in this and similar work. Of the explanations offered to date (attachment kinetics, dendrite fragmentation, cavitation due to shrinkage effects, other fluid-flow arguments), none has managed to predict the phenomena experimentally observed in both alloys and pure materials.

3. A further study of the solidification behavior of Fe-Ni alloys using the high-speed camera could further characterize the double-recalescence phenomenon and help fill in the solidification map in this work at compositions outside Fe-(10-12) wt.% Ni. Experiments incorporating on-camera quenching, with subsequent microstructural analysis, might also shed light on the microstructural evolution that takes place following primary BCC solidification.

Appendices

A1. Solidification Theory

The dendrite growth theory used to create the "LKT" solidification velocity plots for Ni and Fe-Ni in this work stems from the work of Ivantsov [25] and Lipton, et al [9], with additional contributions by Boettinger, et al [29], Aziz [30], and Turnbull [31].

In short, Ivantsov proposed the first analytic equation to determine heat flow in a solidifying dendrite tip realistically modelled as a parabaloid of revolution, resulting in a single relationship linking R, dendrite tip radius, with V, solidification velocity. Lipton, et al later derived a secondary relationship between these two variables on the basis of morphological stability arguments. The combination of this work and Ivantsov's prior relationship formed the most widely accepted theory of dendrite growth established to date. Incorporating modifications by the other contributors listed above, researchers were able to predict accurately the undercooling-velocity behavior observed in many different elements and alloy melts.

For further information regarding LKT solidification theory, refer to the above work or individual treatments given by Piccone [26] and Barth, et al [21]. The next five pages show a numerical implementation of the growth model using the symbolic mathematics package Maple V 5.0[i], following closely the developments of Piccone and Barth.

[i] Maple is a trademark of Waterloo Maple Software

```
>
 LKT SOLIDIFICATION MODEL IMPLEMENTED IN "Maple version 5.0,"
 SYMBOLIC MATHEMATICS SOFTWARE AVAILABLE FOR UNIX WORKSTATIONS.

 NOTE: This is an effort to implement a create a clear, usable
 LKT model that will calculate undercoolings, radii, etc., at
 various solidification velocities.

 * All thermodynamic constants will be expressed in SI units.
```

>
Enthalpy of fusion: (assumed equal in liquid and solid)

> H := 1.746e9;

$$H := .1746\ 10^{10}$$

>
Heat capacity: (assumed equal in liquid and solid)

> c_p := 6.029e6;

$$c\_p := .6029\ 10^{7}$$

>
Solid/liquid interfacial energy:

> sigma := 0.28;

$$\sigma := .28$$

>
Thermal diffusivity of undercooled liquid:

> alpha := 5e-6;

$$\alpha := .5\ 10^{-5}$$

>
Solutal diffusivity of undercooled liquid:

> D_o := 6e-9;

$$D\_o := .6\ 10^{-8}$$

>
Marginal stability constant:

> sigma_star := 1 / (4 * Pi^2);

$$sigma\_star := \frac{1}{4}\frac{1}{\pi^2}$$

>

Assumed kinetic undercooling coefficient (if kinetic undercooling is to be ignored, set this to a very high value, such as 1e10):

```
> mu := 0.28;
```

$$\mu := .28$$

>
Solute concentration in atomic %, i.e. Fe-20 at.% Ni --> "20"

```
> C_o := 10.4;
```

$$C\_o := 10.4$$

>
Equilibrium liquidus temperature:

```
> T_f := 1770;
```

$$T\_f := 1770$$

>
Calculated value for entropy of fusion:

```
> delS := H / T_f;
```

$$delS := 986440.6779$$

>

>
Here is the approximate slope of the liquidus. Note this is a NEGATIVE number, in K/at %:

```
> m := -2.1;
```

$$m := -2.1$$

>
Next is the "equilibrium partition coefficient," used in the calculation of the velocity-dependent slope of the phase diagram of the material:

```
> ke := 0.74;
```

$$ke := .74$$

>
Also need the "atomic diffusive speed:"

```
> Vd := 20;
```

$$Vd := 20$$

```
> k := (V) -> (ke + V / Vd) / (1 + V / Vd);
```

$$k := V \rightarrow \frac{ke + \dfrac{V}{Vd}}{1 + \dfrac{V}{Vd}}$$

>
This produces a velocity-dependent slope in the effective phase diagram:

> m1 := m * (1 + (ke - k(V)*(1 - ln(k(V)/ke))) / (1 - ke));

$$m1 := -8.076923077 + 8.076923077 \frac{\left(.74 + \frac{1}{20} V\right)\left(1 - \ln\left(1.351351351 \dfrac{.74 + \frac{1}{20} V}{1 + \frac{1}{20} V}\right)\right)}{1 + \frac{1}{20} V}$$

>
With all physical constant input, here are the equations to predict growth:

>
First, the definition of the Ivantsov function (a function of a Peclet number):

> Iv := (P) -> P * exp(P) * Ei(1, P);

$$Iv := P \rightarrow P \, e^{P} \, \mathrm{Ei}(1, P)$$

>
Thermal Peclet number:

> P_t := (V, R) -> R * V / (2 * alpha);

$$P\_t := (V, R) \rightarrow \frac{1}{2} \frac{R V}{\alpha}$$

>
Solutal Peclet number:

> P_c := (V, R) -> R * V / (2 * D_o);

$$P\_c := (V, R) \rightarrow \frac{1}{2} \frac{R V}{D\_o}$$

>
Constitutional undercooling w Aziz velocity-dependent slope as per Barth paper:

> delT_c := (P_c(V, R), k(V))-> m * C_o * (1 - (m1 / m) / (1 - (1 - k) * Iv (P_c(V, R))));

$$delT\_c := (\mathrm{P\_c}(V,R), \mathrm{k}(V)) \rightarrow m\, C\_o \left(1 - \frac{ml}{m\,(1-(1-k)\,\mathrm{Iv}(\mathrm{P\_c}(V,R)))}\right)$$

>

Capillary undercooling:

```
> delT_r := (R) -> 2 * sigma / (R * delS);
```

$$delT\_r := R \rightarrow 2\,\frac{\sigma}{R\; delS}$$

>

Kinetic undercooling:

```
> delT_k := (V) -> V / mu;
```

$$delT\_k := V \rightarrow \frac{V}{\mu}$$

>

Ivantsov thermal undercooling solution:

```
> delT_t :=(P_t(V,R)) -> (H / c_p) * Iv (P_t(V,R));
```

$$delT\_t := \mathrm{P\_t}(V,R) \rightarrow \frac{H\,\mathrm{Iv}(\mathrm{P\_t}(V,R))}{c\_p}$$

>

This is the marginal stability solution for dendrite tip radius, as copied from Piccone's Sc.D thesis:

Defined first are two simplifying expressions, n and g:

```
> n := (P_t(V,R)) -> 1 / (sqrt(1 + 1 / (sigma_star * P_t(V,R)^2)));
```

$$n := \mathrm{P\_t}(V,R) \rightarrow \frac{1}{\mathrm{sqrt}\left(1 + \frac{1}{sigma\_star\,\mathrm{P\_t}(V,R)^2}\right)}$$

```
> g := (P_c(V,R), k(V)) -> 2 * k(V) / (1 - 2*k(V) - sqrt(1 + 1 / (sigma_star * P_c(V,R)^
  2)));
```

$$g := (\mathrm{P\_c}(V,R), \mathrm{k}(V)) \rightarrow 2\,\frac{\mathrm{k}(V)}{1 - 2\,\mathrm{k}(V) - \mathrm{sqrt}\left(1 + \frac{1}{sigma\_star\,\mathrm{P\_c}(V,R)^2}\right)}$$

>

Here is the actual equation defining R (here, R_th or theoretical) in terms of V and itself:

```
> R_th := (P_t(V,R), P_c(V,R), k(V)) -> (sigma / (sigma_star * delS)) / ((P_t(V,R) * H
  * (1 - n(P_t(V,R))) / c_p) - ((2 * m * C_o * (1 - k(V)) * (1 + g(P_c(V,R))) * P_c(V,R))
/ (1 - (1 - k(V)) * Iv(P_c(V,R)))));
```

$$R\_th := (\mathrm{P\_t}(V, R), \mathrm{P\_c}(V, R), \mathrm{k}(V)) \rightarrow \sigma \Big/ \left(sigma\_star\; delS \left(\frac{\mathrm{P\_t}(V, R)\, H\, (1 - \mathrm{n}(\mathrm{P\_t}(V, R)))}{c\_p} - 2\, \frac{m\, C\_o\, (1 - \mathrm{k}(V))\, (1 + \mathrm{g}(\mathrm{P\_c}(V, R)))\, \mathrm{P\_c}(V, R)}{1 - (1 - \mathrm{k}(V))\, \mathrm{Iv}(\mathrm{P\_c}(V, R))} \right) \right)$$

>
With the equation above, we can numerically solve for R given any V value:

```
> vval := 2.107;
```

$$vval := 2.107$$

```
> rval := fsolve(R = subs(V = vval, R_th(P_t(V,R), P_c(V,R), k(V))), R,  R=1e-9..1e-
  4);
```

$$rval := .4122501976\ 10^{-6}$$

>
>
This is the real meat of the development; it iterates through a user-defined series of velocities by a variable amount and computes each of the various dendrite tip radii, as well as the various undercooling contributions, at each velocity value.

The output is spewed into tab-delimited column format as the file "outfile".

*** NOTE: If the iteration refuses to converge for some conditions, you'll have to alter the search range defined for R in the "fsolve" step on line 3 of the expression below. The horrible run-on is due to the fact that I couldn't get the "continuation" behavior to work correctly for this command, perhaps because the "writeto" function is involved.

```
> vstart := 0.5; vend := 61; vstep := 4;
```

$$vstart := .5$$

$$vend := 61$$

$$vstep := 4$$

```
> writeto(outfile); print(printf(`delT\tvel\trad\tP_t\tP_c\tdelT_t\tdelT_c\tdelT_r\tdel
  T_k\n`)); for vval from vstart by vstep to vend do writeto(terminal); rval := fsolv
  e(R = subs(V = vval, R_th(P_t(V,R), P_c(V,R), k(V))), R,  R=1e-9..1e-4); Pt := P_t(
  vval, rval); Pc := P_c(vval, rval); delTt := evalf(subs (V = vval, R = rval, delT_t(P
  _t(V,R)))); delTc := evalf(subs(k = k(vval), (subs(V = vval, R = rval, delT_c(P_c(
  V, R), k(V)))))); delTr := delT_r (rval); delTk := delT_k (vval); delT := delTt + delT
  c + delTr + delTk; appendto(outfile); print(printf(`%e\t%e\t%e\t%e\t%e\t%e\t%e\
  t%e\t%e\n`, delT, vval, rval, Pt, Pc, delTt, delTc, delTr, delTk)); od; writeto(termi
  nal);
```

A2. Tabulated Thermodynamic Data

Thermodynamic data for Ni and Fe-Ni alloys were again drawn from the prior work of Piccone [26] and Barth, et al [21], and are given in the following table:

Property	Units	Pure Ni	BCC Fe-10 wt.% Ni	BCC Fe-12 wt.% Ni
thermal diffusivity, α	m^2/s	1.6×10^{-5}	5.0×10^{-6}	5.0×10^{-6}
solutal diffusivity, D_0	m^2/s		6.0×10^{-9}	6.0×10^{-9}
enthalpy of fusion, H_f	J/kg	2.98×10^5	2.40×10^5	2.41×10^5
heat capacity, c_p	J/kgK	7.34×10^2	8.28×10^2	8.29×10^2
solid/liquid interfacial energy, σ	J/m^2	0.255	0.28	0.28
marginal stability constant	[none]	$1 / 4\pi^2$	$1 / 4\pi^2$	$1 / 4\pi^2$
linear kinetic undercooling coefficient, μ	m/sK	0.40	0.28	0.28
solute concentration	at %		10.4	12.5
equilibrium liquidus temperature	K	1728	1769	1759
liquidus slope, m	K/at %		-2.1	-2.1
equilibrium partition coefficient, k_{eq}	[none]		0.74	0.74

A3. Tabulated Experimental Data

Undercooling-velocity data shown from other work was obtained either from a table supplied in the appendix of the work (Piccone [26]) or by scanning a published figure into a computer and using image-analysis techniques to determine plot values from the resulting images (Bassler, et al [8]; Walker [2]; Hofmeister, et al [6]; Schleip, et al [12]; Colligan and Bayles [3]; Barth [21]). The error inherent to the scanning technique is estimated at 0.5%.

The following tables summarize the undercooling-velocity values both of the prior work and the current investigation of Ni and Fe-Ni alloys:

Part I: Pure Ni

Piccone	
ΔT (K)	velocity (m/s)
6	0.03
6	0.03
8	0.06
8	0.05
10	0.03
12	0.04
12	0.08
16	0.14
25	0.16
32	0.32
46	0.15
47	0.14
49	0.20
60	0.35
61	0.67
114	20.0
115	69.8
132	29.3
141	35.7

Walker	
ΔT (K)	velocity (m/s)
27.6	1.0
40.6	1.7
47.5	2.9
53.2	4.3
59.8	4.3
61.2	6.2
71.2	7.3
75.8	7.2
80.4	8.3
86.8	11.1
91.3	11.6
95.2	11.7
99.3	13.4
100.3	16.0
109.8	14.1
108.9	15.4
111.9	20.6
121.9	19.5
120.8	20.0

152	13.8
169	62.9
178	56.6
190	64.7
194	43.1
196	45.5
213	46.2
215	54.2
260	46.2
263	44.6
264	50.8
271	102.3
277	67.7
277	60.4
279	68.7
280	62.5
284	48.1
286	80.4
286	55.6
286	48.4
288	84.9
288	54.5
289	52.9
294	50.6
298	83.3
302	69.2
304	52.6
306	62.5
307	55.9
308	70.3
309	52.9
315	59.2
316	55.9
317	49.5
317	62.1

123.3	20.8
126.1	20.9
121.3	24.6
133.6	25.1
130.4	26.9
138.8	27.6
143.0	29.0
146.6	28.8
158.7	28.9
154.1	32.9
158.9	36.5
166.7	34.7
171.7	36.2
169.2	38.7
182.7	36.6
182.5	38.5
191.8	37.6
188.6	40.7
169.0	44.7
177.7	45.5
188.4	43.5
201.9	41.4
200.5	44.9
232.2	33.1
223.3	41.1
214.2	47.6
248.9	48.7
211.9	61.1
233.4	64.4
218.3	70.0
241.1	77.7

Bassler	
ΔT (K)	velocity (m/s)
133.0	9.7
132.0	10.1
145.7	12.7
149.5	13.4
153.6	13.6
167.6	17.8
165.7	18.3
169.4	18.3
173.0	18.4
182.7	18.1
177.7	20.3
178.9	19.3
180.3	20.7
182.5	19.0
184.1	19.8
187.2	19.4
190.3	20.6
200.5	19.4
201.0	19.8
206.2	19.6
240.8	20.8
260.4	20.2
266.9	20.2
293.6	21.7
292.5	20.8
295.7	20.8
294.3	20.0
294.3	20.3
297.7	20.2
300.8	20.3

Hofmeister	
ΔT (K)	velocity (m/s)
28.1	2.7
30.6	3.1
34.7	3.8
54.1	9.3
71.5	9.3
80.6	11.5
82.4	11.9
112.1	13.3
125.8	14.6
145.0	15.2
148.4	15.7
156.7	16.1
161.7	16.5
161.7	15.9
179.5	15.9
194.6	16.0
202.6	16.1
214.2	16.6
254.4	16.2
267.4	16.4
289.6	16.3
298.5	16.2
239.5	34.9

Colligan/Bayles	
ΔT (K)	velocity (m/s)
23.0	1.0
31.2	1.5
36.4	2.3
42.4	2.8
46.1	3.5
54.2	4.3
60.2	5.2
55.0	3.6
68.4	5.2
63.9	5.8
79.5	5.8
80.2	6.3
79.5	6.6
78.8	6.9
76.5	7.7
78.0	8.4
80.2	9.1
78.0	9.3
89.9	8.4
95.1	8.9
95.8	9.2
102.5	9.9
94.4	11.3
112.9	14.5
115.2	14.5
116.7	14.5
118.1	16.7
116.7	17.5
111.5	17.9
115.2	17.9
117.4	17.9
128.5	19.6
130.0	20.0

Schleip	
ΔT (K)	velocity (m/s)
67.6	0.9
81.4	2.2
89.0	3.7
96.9	5.1
106.0	5.0
108.9	7.4
118.6	8.6
128.2	7.8
130.6	8.0
134.7	9.9
136.7	12.1
147.5	12.2
148.1	17.4
151.1	17.7
151.6	18.5
156.0	17.8
174.2	31.1
182.1	34.4
184.7	34.2
185.3	34.7
200.2	40.5
202.0	40.1
188.8	45.1
209.0	39.9
209.0	41.4
210.2	43.9
208.7	47.6
221.3	48.4
228.9	44.0
238.9	50.8
253.5	56.0
264.1	55.8
303.9	66.0

142.7	21.4
139.7	21.7
138.2	22.9
150.1	26.3
150.1	26.5
145.6	26.8
144.1	27.3
146.4	29.4
151.6	30.3
152.3	30.1
154.5	30.2
162.7	32.1
166.4	32.9
167.9	32.5
171.6	32.0
187.2	34.9
188.7	35.0
186.5	35.7
187.2	37.1
193.9	38.0
202.8	37.0
168.7	37.6
158.3	38.2
164.9	39.9
189.5	40.0
184.3	41.9

323.2	72.2

Lum (this work)	
ΔT (K)	velocity (m/s)
262	41
296	46
101	11
143	19
110	12
128	20
187	35
113	11
93	8.2
287	47
283	50
285	46
284	45
280	46
203	32
273	38
74	4.6
131	28
119	16
122	19
236	38

Part II: Fe-Ni

Barth FCC Fe-25 at.% Ni	
ΔT (K)	velocity (m/s)
73.3	2.7
77.1	1.9
88.5	2.1
93.4	3.0
96.7	6.0
97.7	5.1
110.8	4.1
122.7	9.5
144.4	14.9
150.4	17.7
153.1	10.8
174.8	40.2
179.2	30.3
180.8	32.3
196.0	32.9
197.7	30.2
198.7	32.1
203.1	31.5
211.2	30.9
217.2	34.4

Barth FCC Fe-30 at.% Ni	
ΔT (K)	velocity (m/s)
69.4	4.1
85.1	4.5
93.2	4.7
95.9	7.3
101.3	5.3
118.7	6.2
137.1	14.7
139.2	11.1
142.0	10.6
147.4	13.5
172.8	22.1
175.5	16.8
189.1	26.5
202.1	22.0
203.7	33.6
205.3	31.2

Lum BCC Fe-10 wt.% Ni	
ΔT (K)	velocity (m/s)
238	32.8
243	27.5
240	31
181	19.9
237	27.9
233	35.3
132	6.5
154	11.4
227	33.3
143	12.4
135	8.6
244	32.9
145	14.5
91	4.2
185	21.1
155	7
62	5.2
185	16.4
229	32.7
170	13.9

Lum BCC Fe-12 wt.% Ni	
ΔT (K)	velocity (m/s)
195	22.8
117	13.6
114	16.5
104	8
138	14.6
222	28
167	17.1
186	23.7
113	14.1
110	15.6
214	26.3
195	25.4
226	37.2
228	32.8
167	19.8
136	15.9
95	6.1
152	12.7
195	20.6
161	14.7

A4. Computer Code

The following pages contain the original computer code developed by the author during this work.

Listed first is "**Nic-xfer.c**" (three pages), the routine used to translate Nicolet oscilloscope data from an older, unwieldy "LOTUS 1-2-3" format to a more usable tab-delimited column format.

Next is "**mk-cyl-data.c**" (two pages), the program which input text-style composite interface IPLab images and produced datafiles suitable for input to the solidification model (cylsim11.c), simultaneously querying the user for various calibration values.

Listed finally is "**cylsim11.c**" (seventeen pages). This is the computer-based solidification model which made possible the determination of the solidification velocities of the Ni and Fe-Ni specimens investigated in this work.

```
/****************************************************************/
/*                                                              */
/*     This program is designed to read a file generated by     */
/*     "4094.ltu" and render a tab-delimited datafile arranged  */
/*     in this format:

   COL1       COL2        COL3        COL4       COL5       COL6
 ---------------------------------------------------------------------
 CH1 Time  CH1/A data  CH1/B data   CH2 time  CH2/A data  CH2/B data

(assuming that the data from every channel is desired by the user)

       The user specifies which channel he wants, and the       */
/*     program subsequently informs the user what the data      */
/*     acquisition rate of his channel was.                     */
/*                                                              */
/*     To be more specific, the files created using "4094.ltu"  */
/*     are structured as follows:

    COL1   COL2   COL3   COL4   COL5   COL6   COL7   COL8   COL9   COL10
 ------------------------------------------------------------------------
  data    CH1/A  CH1/A  CH2/A  CH2/A  CH1/B  CH1/B  CH2/B  CH2/B   CTL
  info    first  second first  second first  second first  second  chars
   |      half   half   half   half   half   half   half   half     |
   |       |      |      |      |      |      |      |      |       |
   |       |      |      |      |      |      |      |      |       |
  \ /     \ /    \ /    \ /    \ /    \ /    \ /    \ /    \ /     \ /
   '       '      '      '      '      '      '      '      '       '
         (2029   (1919
         points) points)
 ------------------------------------------------------------------------

   Where there are 2029 rows of actual data, and CH M/N
   refers to the Mth plugin (plugin 1 == LH side of 'scope,
   plugin 2 == RH side) and the Nth channel (Channel 1 ==
   top channel of a given side, Channel 2 == bottom channel).

   Note:  "4094.ltu" is one of the "Henry" data transfer
   programs supplied to us by Nicolet, for use with their
   Nicolet oscilloscopes, of which we have two.  It is
   located on the hard drive of the PC in the end lab, as
   well as on a backup floppy diskette that is either in
   the end lab or else in Dr. Kattamis' old office.

                                                                */
/*                       --J. Lum 4/25/95                       */
/****************************************************************/

#include <stdio.h>

#define TRUE (1)
#define FALSE (0)
#define NEWLINE ('\n')
#define TAB ('\t')
#define SPC (' ')
#define END ('\0')
#define QUOT ('"')
```

```
#define COL1_LINES (2029)
#define COL2_LINES (1919)

int get_field(char *, FILE **);

main()
{
  char readfile[20], writefile[20], field[128];
  FILE *fpread, *fpwrite;
  int i, j;
  char ch1[5], ch2[5], ch3[5], ch4[5];
  int chan[5];
  int count = 0, counter = 0;
  double perpoint1, perpoint2;
  double ch_vals[5][3950];
  double time1, time2;

/****************************************************************/
/*     Note: the ch_vals matrix will hold the various           */
/*     voltage values from the columns of the plugin desired.   */
/****************************************************************/

  printf("\nPlease enter the name of the \"read\" file:  ");
  scanf("%s", readfile);
  printf("\nPlease enter the name of the \"write\" file:  ");
  scanf("%s", writefile);
  printf("\nEnter the time of the first point of plugin 1:  ");
  scanf("%lf", &time1);
  printf("Enter the time per point of plugin 1 (sec)\n");
  printf("(enter 0 if plugin was not used):  ");
  scanf("%lf", &perpoint1);
  printf("\nEnter the time of the first point of plugin 2:  ");
  scanf("%lf", &time2);
  printf("Enter the time per point of plugin 2 (sec)\n");
  printf("(enter 0 if plugin was not used):  "),
  scanf("%lf", &perpoint2);

  printf("\n\nPlease verify which of the channels you want output:\n");
  printf("\tChannel 1 (PLUGIN 1A) (y/n):  "); scanf("%s", ch1);
  printf("\tChannel 2 (PLUGIN 2A) (y/n):  "); scanf("%s", ch2);
  printf("\tChannel 3 (PLUGIN 1B) (y/n):  "); scanf("%s", ch3);
  printf("\tChannel 4 (PLUGIN 2B) (y/n):  "); scanf("%s", ch4);
  for (i = 1; i < 5; i++)
    chan[i] = 0;
  if (strcmp (ch1, "y") == 0)
    chan[1] = TRUE;
  if (strcmp (ch2, "y") == 0)
    chan[2] = TRUE;
  if (strcmp (ch3, "y") == 0)
    chan[3] = TRUE;
  if (strcmp (ch4, "y") == 0)
    chan[4] = TRUE;

  fpread = fopen(readfile, "r");
  fpwrite = fopen(writefile, "w");

  if ((fpread == NULL) || (fpwrite == NULL)) {
```

```c
    printf("\n\nFile Error.  Exiting.\n\n\n");
    exit(0);
  }

  printf("\n**********************************\n");
  printf("*** SCANNING THROUGH 4094 FILE ***\n");
  printf("**********************************\n");

  for (i = 1; i <= COL1_LINES; i++)
    for (j = 1; j <= 10; j++) {
      count = get_field (field, &fpread);
      if (count == 0) break;
      if ((j != 1) && (j != 10)) {
        if (strcmp ("\"\"", field) != 0) {
          if ((j != 1) && (j != 10)) {
            if ((j % 2) == 0)
              sscanf(field, "%lf", &ch_vals[j/2][i]);
            else
              sscanf(field, "%lf", &ch_vals[(j-1)/2][i+COL1_LINES]);
          }
        }
      }
    }

  printf("\n\n");

  printf("**********************************\n");
  printf("***    PRINTING OUTPUT FILE    ***\n");
  printf("**********************************\n");

/****************************************************************/
/*     Here is where the data is printed to file               */
/*     for whichever channels were chosen.                     */
/****************************************************************/

  for (i = 1; i <= (COL1_LINES + COL2_LINES); i++) {
    if (perpoint1 > 0) {
      fprintf(fpwrite, "%lf\t", time1);
      if (chan[1] == TRUE)
        fprintf(fpwrite, "%lf\t", ch_vals[1][i]);
      if (chan[3] == TRUE)
        fprintf(fpwrite, "%lf\t", ch_vals[3][i]);
    }
    if (perpoint2 > 0) {
      fprintf(fpwrite, "\t\t");
      fprintf(fpwrite, "%lf\t", time2);
      if (chan[2] == TRUE)
        fprintf(fpwrite, "%lf\t", ch_vals[2][i]);
      if (chan[4] == TRUE)
        fprintf(fpwrite, "%lf\t", ch_vals[4][i]);
    }
    fprintf(fpwrite, "\n");
    time1 += perpoint1;
    time2 += perpoint2;
  }

  fclose(fpread); fclose(fpwrite),
```

```c
}
/*----------------------------------------------------------------------*/
/****************************************************************/
/*     "get_field" is necessary because the data from the      */
/*     4094.ltu program is not very conveniently separated.    */
/*     The big problem is the fact that white spaces are       */
/*     mixed in with column 1 information for a portion of     */
/*     the datafile.                                           */
/****************************************************************/
int get_field(char *target, FILE **orig_fp) {
  int INSIDE_QUOTE = FALSE;
  char c = 'f';
  int counter = 0;
  FILE *fp;

  /* Set up a "local" file pointer to mirror the original one passed to us */
  fp = *orig_fp;

  /* FIRST, read in chars until no more whitespaces */
  do {
    c = getc(fp);
    /* If the end-of-file is reached, return 0 characters in this field */
    if (c == EOF) {
      /* Ensure that any changes we made to the file pointer are reflected */
      /*  in the ORIGINAL file pointer in the calling function */
      *(orig_fp) = fp;
      *target = END;
      return(0); } }
  while ((c == SPC) || (c == TAB) || (c == NEWLINE));

  /* AT this point, we have read one character deep into the next field */
  /*  (put this character into the target string */
  *(target + counter) = c;
  counter++;

  if (c == QUOT) {
    /* Take special action if a quote is the first character */
    do {
      c = getc(fp);

      if (c == EOF) {
        printf("ERROR: reached EOF inside a quotation\n");
        exit(1);
      }

      *(target + counter) = c;
      counter++;
    }
    while (c != QUOT);

    /* Ensure that any changes we made to the file pointer are reflected */
    /*  in the ORIGINAL file pointer in the calling function */
    *(orig_fp) = fp;

/*  I CHANGED THIS TO "counter" INSTEAD OF "counter + 1" AS THE */
/*  KNAVE HAD IT, BECAUSE I THINK HE WAS WRONG.                 */
    *(target + counter) = END;
    return(counter);
  }

  /* Otherwise, we have a "normal" (unquoted) field */
  do {
    c = getc(fp);

    if (c == EOF) {
      *(target + counter) = END;
      /* Ensure that any changes we made to the file pointer are reflected */
      /*  in the ORIGINAL file pointer in the calling function */
      *(orig_fp) = fp;
      return(counter);
    }

    if ((c != SPC) && (c != TAB) && (c != NEWLINE)) {
      *(target + counter) = c;
      counter++;
      /* perform some error checking */
      if (c == QUOT) {
        printf("Ach, gehfeh!  Encountered a lone quotation mark!\n");
        exit(1);
      }
    }
  }
  while ((c != SPC) && (c != TAB) && (c != NEWLINE));

  /* Ensure that any changes we made to the file pointer are reflected */
  /*  in the ORIGINAL file pointer in the calling function */
  *(orig_fp) = fp;

/*  THIS SHOULD ALSO BE "counter" INSTEAD OF "counter + 1" */
  *(target + counter) = END;
  return(counter);
}
```

```
/************************************************************************/
/*  THIS FILE WAS MADE IN ORDER TO TRANSFORM THE TEXT-STYLE             */
/*  "POINTFILES" CREATED IN IPLAB (REPRESENTING ALL THE INTERFACES      */
/*  FROM A GIVEN RECALESCENCE) INTO THE BASIS FOR A DATAFILE TO BE      */
/*  USED BY THE CYLINDER-TYPE SOLIDIFICATION MODEL PROGRAM, cylsim11.c */

/*  THERE IS A CORRESPONDING PROGRAM FOR THE LEVITATED, ELLIPSOIDAL     */
/*  APPROACH AS WELL (mk-lev-data.c).

/*

  Assumed input format:  an mxm-pixel IPLab datafile, saved as
text, resulting in something like the following:

------------------------------------------
128
128
I

0   0   0   1   4   2   0   1   2   ...
2   4   32  0   4   8   0   64  0   ...
------------------------------------------

Where the first two numbers refer to the width and height of the image
(in pixels), and the I refers to "image" or something.  This is a
typical IPLab image format, when saved as text.

Note that each pixel has a value of either 0 or some power of 2.  A
zero means that no interface pixels were recorded at that screen
location in any frame; a non-zero number indicates that there was at
least one interface pixel specified from some frame at that screen
location.

For simple power-of-two numbers, the frame associated with the
interface pixel is simply (log_2(number)) + 1, so that 1's indicate an
interface pixel occurring in the first frame only.

For non-exponential numbers, there were multiple interface points
specified at that screen location (points from more than one frame).
The individual frames in which that screen location was an interface
point can be determined by finding the component powers of two that
comprise the number given, and applying the log_2 relation above to
each of the components.
                                    --John Lum 5/8/96

*/

#include <stdio.h>
#include <math.h>

/*  MAXIMUM NUMBER OF FRAMES ALLOWED IS LIMITED BY IPLAB'S LARGEST */
/*  LONG INT; IT IS SOMEWHERE AROUND 2^25, SO YOU CANNOT RECORD MORE */
/*  THAN 25 SETS OF INTERFACE POINTS IN A SINGLE FILE. */
#define MAXFRAMES (25)

/*  THE MAXIMUM NUMBER OF POINTS WHICH CAN BE SPECIFIED FOR A SINGLE */
/*  FRAME IS CONSTRAINED HERE TO BE 60; THIS IS NOT DUE TO ANY */
/*  NUMERICAL LIMITATION, BUT SIMPLY BECAUSE HAVING MORE POINTS THAN */
/*  THIS FOR A GIVEN INTERFACE MEANS THAT YOU ARE GOING TO HAVE */
/*  TREMENDOUSLY SLOW PROGRAM EXECUTION. */
#define MAXPTS (60)
#define EXTRA_HEADER (1)

int get_frame_number (long int);

main()
{
  FILE *fin, *fout;
  char readfile[30], writefile[30], garbage[30];
  int i, j, a, index, max_a = 0;
  double width, height, liquidus, coeffexp;
  long int fval;
  double x[MAXFRAMES][MAXPTS], y[MAXFRAMES][MAXPTS];
  int numpoints[MAXFRAMES];
  int framenum, firstframe, skip, key;
  double scale, xc, yc, calib;

  for (i = 1; i <= MAXFRAMES; i++)
    numpoints[i] = 0;

  printf("\nFilename for read:  ");
  scanf("%s", readfile);
  printf("\n  \".new\" will be appended...\n\n");
  sprintf(writefile, "%s.new", readfile);

  printf("(Writefile name is %s.\n\n", writefile);
  fin = fopen(readfile, "r");
  fout = fopen(writefile, "w");
  if ((fin == NULL) || (fout == NULL)) {
    printf("\n**FILE ERROR: DATAFILE DOESN'T EXIST!\n\n");
    exit(0);
  }

/*  THIS PART STRIPS THE HEADER INFO FROM THE TOP OF THE IPLAB */
/*  DATAFILE AND STORES THE INFORMATION REGARDING THE NUMBER OF ROWS */
/*  AND COLUMNS IN THE DATAFILE FOR FURTHER USE. */
  fscanf(fin, "%lf%lf", &width, &height);
  for (i = 1; i <= EXTRA_HEADER; i++)
    fscanf(fin, "%s", garbage);
  if (height != width) {
    printf("Illegal image format: image must be square.\n\n");
    exit(0);
  }
  scale = height / 64.0;

  for (j = 0; j < height; j++)
    for (i = 0; i < width; i++) {
      if (fscanf(fin, "%ld", &fval) != 1) {
        printf("\n\n**FORMAT ERROR: IMAGE FILE WRONG LENGTH!\n\n");
        exit(0);
      }
      else while (fval > 0) {
        a = get_frame_number (fval);
/*  RECORDS LARGEST FRAME INDEX YET SEEN */
```

```
        if (a > max_a)
          max_a = a;
        index = ++numpoints[a];
/*  MUST NORMALIZE IMAGE SIZE INTO 64X64 GRID TO ALLOW COMPATIBILITY */
/*  WITH THE SOLIDIFICATION MODEL PROGRAM */
        x[a][index] = ((double) i / scale);
/*  IMAGE Y-AXIS IS REVERSE OF SOLIDIFICATION MODEL'S Y-AXIS: */
        y[a][index] = ((double) ((height - 1)-j)/scale);
        fval = fval - pow(2, a);
      }
    }

  for (i=1; i<=max_a; i++)
    printf ("frame number is: %d  points read in: %d\n",i, numpoints[i]);

/*  AT THIS POINT, THE PROGRAM PROMPTS FOR THE HEADER INFORMATION FOR */
/*  THE FINISHED DATAFILE--THINGS SUCH AS NUMBER OF FRAMES, ROTATIONS */
/*  AND CENTERPOINTS, ETC. */
  printf ("\nWhat is the total number of frames to be analyzed?\n");
  scanf ("%d", &framenum);
  printf ("\nHow many interface frames to ignore at the beginning? \n");
  printf("(I.E., in case there are bad or unnecessary frames of data):  ");
  scanf ("%d", &firstframe);
  printf ("\nWhat was the skip rate for recorded frames?\n");
  printf ("Enter \"0\" for every frame, \"19\" for every 20th frame...\n");
  scanf ("%d", &skip);
  printf ("\nWhich frame is to be \"key\" frame?  (Regardless of\n");
  printf ("\"actual\" number, give index from above list\n");
  printf("(1, 2, 3, 4...); NOTE: key frame used for model's initial\n");
  printf("guess of nucleation position, and is not crucial:  ");
  scanf ("%d", &key);
  printf ("\nPlease enter x and y centerpoints:\n");
  printf("NOTE: This are PIXEL values, with the same image size and\n");
  printf("axis orientation as in the interface point datafile, and\n");
  printf("the rest of IPLab in general:  ");
  scanf ("%lf%lf", &xc, &yc);
  xc = ((double) xc / scale);
  yc = ((double) ((height - 1)-yc)/scale);
  printf ("\nPlease enter calibration value (mm/pixel):  ");
  scanf ("%lf", &calib);
  calib = calib * scale;
  printf("\nEnter liquidus of material (K)\n");
  printf("(Will not matter if exp_coeff of 0 is chosen):  ");
  scanf("%lf", &liquidus);
  printf("Enter desired coefficient of expansion (0 if unwanted):  ");
  scanf("%lf", &coeffexp);
  printf ("\n\nFrame rate assumed to be 40500/sec...\n");

  /*  FINALLY, THE DATA CAN BE WRITTEN TO FILE:  */
  fprintf (fout, "%d\n%d\n\n", framenum, 40500);

  skip++;
  for (a=1; a<=(framenum*skip); a+=skip)
  fprintf (fout, "%d ", a);

  fprintf (fout, "\n\n%d\n\n", key);
```

```
  fprintf (fout, "%lf   %lf\n\n", xc, yc);

  fprintf (fout, "y\n%lf\n%lf\n%lf\n\n", calib, liquidus, coeffexp);

  fprintf (fout, "%lf\n%lf\n", 6.25, 5.5);


  for (i = firstframe+1; i <= max_a; i++) {
    if (numpoints[i] > 0) {
      fprintf(fout, "\n%d\n", numpoints[i]);
      for (j = 1; j <= numpoints[i]; j++)
        fprintf(fout, "%lf %lf\n", x[i][j], y[i][j]);
    }

/*
    else
      fprintf(fout, "\n0\n");
*/
  }
  fclose(fin); fclose(fout);
  printf("\n\nAll done!\n\n");

}

int get_frame_number (long int fval) {
  int frame = 1;

  while (pow(2, (frame - 1)) <= fval)
    frame++;

  return (frame - 1);
}
```

```
/***************************************************************/
/*   This is a program for the interpretation of video results  */
/*   from undercooling studies carried out using the EktaPro    */
/*   4540 camera from Kodak.                                    */
/*                                                              */
/*   This particular version of the general program is         */
/*   designed to interpret results from the undercooled        */
/*   specimens which were processed in quartz test tubes, on   */
/*   beds of quartz glass.  The program's main purpose is to   */
/*   determine a velocity of solidification for a given        */
/*   solidification event, and this verison assumes that the   */
/*   velocity is a "BULK" velocity; that is, the solidified    */
/*   material grows as a sphere in the undercooled melt,       */
/*   expanding at a constant rate.                             */
/*                                                              */
/*   SHORT OVERVIEW OF THE STRUCTURE OF THIS PROGRAM:          */
/*     1) read either from file or from keyboard several       */
/*        materials and/or geometric constants associated      */
/*        with the solidification event at hand.  (for         */
/*        example, the radius of the test tube used, the       */
/*        calibration factor to go from pixels to mm, etc)     */
/*                                                              */
/*     2) read, again from file or keyboard, the (x,y)         */
/*        coordinates of every point supplied in every frame.  */
/*        Place these values in respective x, y, and z         */
/*        matrices, where x[1][1] refers to the x coordinate   */
/*        of the first point in the first frame, and y[2][3]   */
/*        refers to the y coordinate of the third point given  */
/*        in the second frame given, etc.                      */
/*                                                              */
/*     3) determine, from the cylindrical geometry of the      */
/*        specimen, the z coordinate associated with each      */
/*        (x,y) pair.  (the positive z-direction is "out of    */
/*        the screen")                                         */
/*                                                              */
/*     4) calculate a rough estimate of the nucleation point-- */
/*        the origin of growth--determined by noting the       */
/*        apparent growth behavior of the interfaces given.    */
/*                                                              */
/*     5) from this spatial estimate, determine an approximate */
/*        "delay time," the time between the nucleation event  */
/*        (t = 0) and the recording of the first pixel of the  */
/*        first frame recorded by the camera.                  */
/*                                                              */
/*     6) now assign a "relative time" value to each (x,y)     */
/*        point; place this value into another matrix, r.      */
/*        The value is determined by summing the "delay time"  */
/*        estimate, the time of the elapsed frames between     */
/*        the first frame and the frame which contains the     */
/*        (x,y) point of interest, and a "raster delay time"   */
/*        which depends upon the position of the (x,y) point   */
/*        on the (assumed) 64x64 pixel video display.          */
/*                                                              */
/*     7) You now have an initial guess for (xn, yn, zn) and   */
/*        tn--the coordinates of the nucleation point and a    */
/*        guess for the "delay time."  Using these values,     */
/*        begin an iterative procedure which attempts to       */
/*        minimize the standard deviation of the average       */
/*        velocities observed for each frame.  That is,        */
/*        given the above values, it is possible to determine  */
/*        v1, v2, v3, ..., vf--the average velocities observed */
/*        for each individual frame.  The idea is to make it   */
/*        so these v values are all nearly the same, thus      */
/*        satisfying our "sphere expanding at a constant       */
/*        velocity" assumption.                                */
/*                                                              */
/*     8) Alternately try slightly different values for xn, yn, */
/*        zn, and tn, until virtually no further decrease in   */
/*        the standard deviation of the average velocities is  */
/*        observed.  At this point, you've reached some local  */
/*        minimum in some function of (xn, yn, zn, tn), and    */
/*        we assume that this point is the actual nucleation   */
/*        point, and that tn is the actual delay time.         */
/*                                                              */
/*     9) Given this final point and time, return the implied  */
/*        average velocity for each frame, as well as the      */
/*        overall average velocity and the standard deviation. */
/*        Assuming the point and delay time are physically     */
/*        reasonable, low standard deviations imply a good     */
/*        agreement between the model and reality, and         */
/*        therefore accurate values for the solidification     */
/*        velocity which is returned.                          */
/*                                                              */
/*                                                              */
/*   NOTE: There are several options embedded in the program.  */
/*     One is to restrict the guesses of the nucleation point  */
/*     to lie on the "round" surface of the cylindrical        */
/*     specimen--the glass/metal interface.  This is a fairly  */
/*     likely physical scenario.                               */
/*      Another option is to hold the delay time, tn, constant */
/*     while iterating only over xn, yn, and zn.  This is a    */
/*     helpful option at times.                                */
/*                                                              */
/*   NOTE: If you use the "read from file" option (which is    */
/*     certainly the most efficient way to go about things, the */
/*     datafile to be read must have a very specific structure. */
/*     This structure is shown in several "EXAMPLE" datafiles  */
/*     which accompany the program.                            */
/*                                                              */
/*   NOTE: The program records the details of every step of    */
/*     the iteration process in an "ouput" datafile, whose name */
/*     ther user supplies.  This way, problems with the        */
/*     iteration process can be more easily identified if, for */
/*     instance, the program were to "hang."                   */
/*                                                              */
/*                            --Doug Matson, John Lum  6/14/95 */
/***************************************************************/

/*  FORMAT OF INPUT DATAFILE:
-------------------------------------------------------------
[VALUE]                        [COMMENT]  (example begins next line)

9                              TOTAL #FRAMES TO BE ANALYZED
40500                          FRAME RATE
```

```
1 4 7 10 13 16 19 22 25 FRAME INDEXES OF FRAMES TO BE ANALYZED

1                               MEMBER OF ABOVE LIST WHICH IS "KEY" FRAME

32.500000   30.000000           X AND Y CENTERPOINTS (NORMALIZED TO 64X64)

y                               CHANGES TO DEFAULTS?  (ALWAYS YES)
0.100000                        CALIBRATION VALUE (mm/pixel)
1726.000000                     LIQUIDUS
0.000000                        COEFFICIENT OF EXPANSION

6.250000                        THESE ARE CALIBRATION VALUES TO CORRECT FOR THE
5.500000                        LENS-EFFECT OF THE CYLINDRICAL QUARTZ TUBE.
                                THE NUMBERS ARE, ROUGHLY, THE ACTUAL TUBE RADIUS
                                AND THE APPARENT TUBE RADIUS, AS OBSERVED.  THE
                                RATIO SHOULD ALWAYS BE ABOUT THE SAME, SO USE
                                THESE NUMBERS IF IN DOUBT.

5                               #POINTS IN FRAME 1
17.000000 57.500000             POINT 1
15.500000 56.000000             POINT 2
14.500000 54.000000             ..
14.000000 52.500000             ..
13.500000 51.000000             ..

6                               #POINTS IN FRAME 2
19.500000 57.000000             POINT 1
18.000000 55.500000             POINT 2
17.000000 54.000000             ..
16.000000 52.000000             ..
15.000000 50.000000             ..
14.500000 48.000000             ..
----------------------------------------------------------

/***************************************************************/
/*      These are typical header files seen in most programs:   */
/***************************************************************/
#include <stdio.h>
#include <math.h>

/***************************************************************/
/*      These "define" statements cause the program to         */
/*      specific values with specific keywords, so that the     */
/*      programmer can use meaningful arguments like "SURFACE"  */
/*      instead of resorting to numbers all the time.           */
/***************************************************************/
#define PI (3.141592654)
#define FMAX (50)
#define PMAX (50)
#define NEG_ROOT (-10000)
#define BIG_FLOAT (1e10)
#define INF (1e10)
#define FAIL (-1)
#define OK (0)
#define XITER (3)
#define YITER (4)
#define ZITER (2)
#define TITER (1)
#define FITER (6)
#define MITER (7)
#define INIT_ITER (8)
#define MID_ITER (9)
#define REP_ITER (5)
#define TMAX (0.5)
#define SURFACE (1)
#define BULK (2)
#define FRONT (0)
#define BACK (1)
#define SCR_HEIGHT (64.0)
#define SCR_WIDTH (64.0)
#define FPRINTOUT (1)
#define PRINTOUT (2)
#define NORMAL (0)
#define TIMEDEV (1)
#define GETNEWTIME (2)
#define TIMEMAX (0.025)

/***************************************************************/
/*      The following are either definitions of special data    */
/*      structures or else function prototypes.  In C, all      */
/*      user-defined functions must have these prototypes at    */
/*      the beginning of the program, so that the compiler      */
/*      knows what functions take what kinds of arguments, etc. */
/***************************************************************/
typedef struct coords {
  double xo; double yo; double zo;
  double xd; double yd; double zd;
  int status;
} Coords;

typedef struct iter_coords {
  double xn; double yn; double zn;
  double tn;
} Iter_coords;

typedef struct coords2 {
  double bx; double by; double rad_curv;
} Cent_Coords;

double get_zval (int, double, double);

Cent_Coords get_center (double *, double *,
                        int, int, int);

double spread (double *, double *, int, int,
               double, double);

Iter_coords gen_iterate (int, int, double, double, double,
                          double *, double *, double *,
                         double, double *, int, int *, double,
                         double, double, FILE *,
                         double *);
```

```
char * get_ittype (int);

double fill_vels (double *, double *, int, int *,
                  double, double, double,
                  double *, double *, double *,
                  double, double *, int, double, double *);

double get_dev (double *, double, int, double);

double get_frame_dev (int, double *, double, double, double,
                      FILE *);

void make_report (int, double, double, double, double,
                  double, double, double, FILE *,
                  double, double);


int was_change (double, double, double, double,
                double, double, double, double);

double create_timedev (int, double, int, int *, double, double,
                     double, double *, double *, double *,
                     double *, double, double *,
                     double, FILE *);

double return_xact (double, double, double, double),

double return_yact (double, double);

double return_xobs (double, double, double, double);

double return_yobs (double, double);


/**************************************************************/
/*      Here's where the program really "starts:"             */
/**************************************************************/
main()
{

/**************************************************************/
/*      I'll declare all of my "global" variables here.       */
/*      "double" means double-precision floating point number, */
/*      "int" means integer, "char" means character string.   */
/**************************************************************/
  double calib, tm, linear, radobs, radact, scale, random_val;
  char fig[10], achar[10], nuctype[5], cont_scan[5];
  char i_or_p[5], t_or_h[5], s_or_b[5];
  double vid[FMAX], timemin, timemax;
  int n[PMAX], m, mx, key, vidinc, p, nx, i, c_nc;
  char cformat[5], response[5];
  double bt, veloc, xobs, yobs, zobs, bxobs, byobs, bzobs;
  int w, v, count, frame, k, frontback, tecplot, contscan;
  double angl, rad_curv1, rad_curv2, ratio, up_bound, end_it;
  double xcoc[3][PMAX], ycoc[3][PMAX];
  double xcent, ycent, xact, yact, zact, frate;
  double r[FMAX][PMAX], x[FMAX][PMAX], y[FMAX][PMAX], z[FMAX][PMAX];
  double rad[FMAX], vel[PMAX*FMAX], xvid, yvid;
  double bx, by, bz, bx1, bx2, by1, by2, zcheck, rsurface;
  Coords drop_frame, curvcoords;
  Cent_Coords centcoords;
  Iter_coords itercoords;
  double delta, testtime, avevel, presdev;
  int numpoints = 0, ntype, ittype, change_total = 0, hundred_cycles = 0;
  FILE *fp, *fpout, *fpin, *fptec;
  double velocity[FMAX][PMAX], fvel[FMAX];
  double xold, xnew, yold, ynew, zold, znew, told, tnew;
  double xreplac, yreplac, zreplac, treplac;
  char outputfile[30], inputfile[30], sysstring[40], rdtype[5];
  char tecfile[30];

/**************************************************************/
/*      Input raw data and calculation options                */
/**************************************************************/
  printf("CONTENTS OF CURRENT DIRECTORY:\n");
  printf("------------------------------\n");
  system ("ls");

  printf("\n\n");
  printf("Read from file (f) or keyboard (other)?  ");
  scanf("%s", rdtype);
  if (strcmp(rdtype, "f") == 0) {
    printf("Name of read file:  ");
    scanf("%s", inputfile); }
  printf("Name of output file:  ");
  scanf("%s", outputfile);
  fpout = fopen(outputfile, "w");
  fpin = fopen(inputfile, "r");

/**************************************************************/
/*      This part for later plotting, temporarily disabled:   */
/**************************************************************/
/*
  printf("\nProduce tecplot-format datafile of interface points?\n");
  printf("  (enter 1 if yes, 0 if no):  ");
  scanf("%d", &tecplot);
  if (tecplot == 1) {
    printf("Enter name for tecplot datafile:  ");
    scanf("%s", tecfile);
    fptec = fopen(tecfile, "w");
    if (fptec == NULL)
      exit(0);
    else {
      fprintf(fptec, "ZONE T=\"Observed interfaces\"\n");
      fprintf(fptec, "  F=POINT\n\n");
    }
  }
*/

  tecplot = 0;

  printf("Number of frames of data: ");
  if (strcmp(rdtype, "f") == 0)
    fscanf(fpin, "%d", &m);
  else
    scanf("%d", &m);
  printf("Inherent frame rate (frames/sec):\n");
  printf("  NOTE: this is actual speed at which camera was recording:  ");
  if (strcmp(rdtype, "f") == 0)
    fscanf(fpin, "%lf", &frate);
  else
    scanf("%lf", &frate);

/****************************************************************/
/*     This for-next loop fills the vid[mx] array with the      */
/*     appropriate "frame" indices of interest. In case the     */
/*     user wanted, for instance, to use only every fifth       */
/*     frame, the values would become 1,6,11,16,21, etc.        */
/****************************************************************/
  printf("\nFor the frame number entries, the absolute values are\n");
  printf("unimportant.  Frame numbers are only meaningful when\n");
  printf("considered \"relative\" to the first frame entered.\n");
  printf("I.E., if your data are from frames 3, 7, 11, 15, etc,\n");
  printf("as taken from the camera (\"every 4\"), then you could\n");
  printf("enter as the relative frame numbers:\n");
  printf("(1, 5, 9, 13, 17, etc)  or\n");
  printf("(3, 7, 11, 15, 19, etc)  or\n");
  printf("any other sequence with each number four integers apart.\n\n");
  for (mx = 1; mx <= m; mx++) {
    printf("Entry Frame %d = Relative Frame #:  ", mx);
    if (strcmp(rdtype, "f") == 0)
      fscanf(fpin, "%lf", &vid[mx]);
    else
      scanf("%lf", &vid[mx]);
  }

/****************************************************************/
/*     This section pertains to the program's effort to make    */
/*     an initial guess as to the nucleation site, etc.  It     */
/*     needs to make this guess from the information on two     */
/*     frames where it can see the interface progressing...     */
/****************************************************************/
  printf("\nWhich entry is first of two key reference frames for ");
  printf("nucleation\nestimate (counting video index %d ", vid[1]);
  printf("as entry frame 1): ");
  if (strcmp(rdtype, "f") == 0)
    fscanf(fpin, "%d", &key);
  else
    scanf("%d", &key);

/****************************************************************/
/*     The following section requests the user to input the     */
/*     starting position of the cylinder's observed centerpoint */
/*     in the x and y directions.  This is quite important in   */
/*     the x-direction, but not important and only a matter of  */
/*     reference for the y-direction.  (since x can only range  */
/*     between (xcent - radius) and (xcent + radius), but y is  */
/*     essentially unbounded; the "height" of the cylinder is   */
/*     not crucial.)                                            */
/****************************************************************/
```

```
  printf("\nKey Frame %.0lf data: ", vid[key]);
  printf("\nAxis of cylinder is the \"y\" direction; \"x\" goes\n");
  printf("from left to right; \"z\" goes into and out of the screen.\n");
  printf("\nObserved centerpoint coordinates (x,y)\n");
  printf("(enter x, then y): ");
  if (strcmp(rdtype, "f") == 0)
    fscanf(fpin, "%lf%lf", &xcent, &ycent);
  else
    scanf("%lf%lf", &xcent, &ycent);

/****************************************************************/
/*     Here is where some "constants" are listed, and possibly  */
/*     changed.  At present, we don't take into account         */
/*     contraction upon solidification, but we could later.     */
/****************************************************************/

  printf("Default settings:\n\tScreen calibration = 0.10 mm/pixel\n");
  printf("\tTm = 1726 K\n\tCoef. of expansion = 0 microns/m degC\n");

  printf("\nAny changes (y/n)?");
  if (strcmp(rdtype, "f") == 0)
    fscanf(fpin, "%s", cformat);
  else
    scanf("%s", cformat);
  if (strcmp(cformat, "n") == 0) {
    calib = 0.1; tm = 1726; linear = 0; }
  else {
    printf("\tScreen size calibration: ");
    printf("\n(on the 64x64 screen, not the 128x128 screen)  ");
    if (strcmp(rdtype, "f") == 0)
      fscanf(fpin, "%lf", &calib);
    else
      scanf("%lf", &calib);
    printf("\tAlloy melting point: ");
    if (strcmp(rdtype, "f") == 0)
      fscanf(fpin, "%lf", &tm);
    else
      scanf("%lf", &tm);
    printf("\tAverage alpha: ");
    if (strcmp(rdtype, "f") == 0)
      fscanf(fpin, "%lf", &linear);
    else
      scanf("%lf", &linear);
    linear *= 1e-6; }

/****************************************************************/
/*     At this point, the user must enter the "observed,"       */
/*     "effective" radius of the cylindrical droplet, as        */
/*     observed or assumed from the digitized data.             */
/*     (refer further into work for explanation of the          */
/*     "effective" radius; this is likely a value greater       */
/*     than the actual, measured radius of the droplet,         */
/*     thanks to magnification characteristics of the quartz)  */
/****************************************************************/

  printf("\nPlease enter the \"observed\" radius of the\n");
  printf("cylindrical droplet (mm):  ");
  if (strcmp(rdtype, "f") == 0)
    fscanf(fpin, "%lf", &radobs);
  else
    scanf("%lf", &radobs);
  radobs /= calib;
/**************************************************************/
/*     The actual radius is also needed.                      */
/**************************************************************/
  printf("\nPlease enter the actual radius of the\n");
  printf("cylindrical droplet (mm) (inner test tube radius):  ");
  if (strcmp(rdtype, "f") == 0)
    fscanf(fpin, "%lf", &radact);
  else
    scanf("%lf", &radact);
  radact /= calib;

/**************************************************************/
/*     Here's where the guts of the program kick in.  We start */
/*     a big loop which begins to cycle through the frames and */
/*     assign (x,y) values to matrices, etc...                 */
/**************************************************************/

  for (mx = 1; mx <= m; mx++) {
    printf("\nFrame %d = Video screen %.0lf\n", mx, vid[mx]);
    vidinc = vid[mx] - vid[1];

/* The above gives us the number of video frames we are
   from the start.  */

    printf("\nPlease give the number of interface pixels: ");
    if (strcmp(rdtype, "f") == 0)
      fscanf(fpin, "%d", &p);
    else
      scanf("%d", &p);
    n[mx] = p;

/**************************************************************/
/*     Need not perform any rotation of translation of given    */
/*     coordinates, since there is no motion of the droplet.    */
/**************************************************************/

/**************************************************************/
/*     Actual points from the interface are read in or entered */
/*     starting at this point in the code.                     */
/**************************************************************/
    for (nx = 1; nx <= p; nx++) {
      printf("\n\nInput interface coordinates (x,y)\n");
      printf("(enter coordinates separated by a space)\n");
      printf("X(%d) Y(%d): ", nx, nx);
      if (strcmp(rdtype, "f") == 0)
        fscanf(fpin, "%lf%lf", &xobs, &yobs);
      else
        scanf("%lf%lf", &xobs, &yobs);

/**************************************************************/
/*     This bit saves the points "as observed" into a small     */
/*     array for both the "key" and "key+1" frames, so that a   */
/*     center of curvature can later be calculated from them.   */
/**************************************************************/
      if (mx == key) {
        xcoc[key][nx] = xobs; ycoc[key][nx] = yobs; }
      if (mx == key+1) {
        xcoc[key+1][nx] = xobs; ycoc[key+1][nx] = yobs; }

/**************************************************************/
/*     Now the "actual" x, y, and z values--that is, the values */
/*     corrected for lens distortion and in a reference frame   */
/*     with x=0 at the midplane of the as-observed cylinder--   */
/*     are calculated and stored in three arrays:               */
/**************************************************************/
      x[mx][nx] = xact = return_xact (xobs, xcent, radact, radobs);
      y[mx][nx] = yact = return_yact (yobs, ycent);

/**************************************************************/
/*     Here we use the geometry of a cylinder to obtain         */
/*     the unknown value of "z"--that is, the distance "out of  */
/*     the screen" that an observed point rests.                */
/**************************************************************/
      z[mx][nx] = zact = get_zval (FRONT, radact, xact);
      if (zact == NEG_ROOT) {
        if (strcmp(rdtype, "f") == 0) {
          printf("\n\n**************************************************\n");
          printf("*  Error in data point entry: point given not  *\n");
          printf("*    within bounds of droplet surface!         *\n");
          printf("**************************************************");
          printf("\nTrouble point was (%.2lf, % 2lf).\n",
                 xobs, yobs);
          printf("\n\tPlease remove the point from \"%s\"\n",
                 inputfile);
          printf("\tand run again.\n");
          exit(0);
        }
        printf("\n");
        printf("**************************************************\n");
        printf("*  Error in data point entry: point given not  *\n");
        printf("*    within bounds of droplet surface!         *\n");
        printf("*  Please enter again.                         *\n");
        printf("**************************************************");
        nx--;
      }
      else {
        zobs = get_zval (FRONT, radobs, xobs - xcent);
        printf("*** Verify:\n");
        printf("(%0.2lf, %0.2lf, z) = (%0.2lf, %0.2lf, %0.2lf)",
               xobs, yobs, xobs, yobs, zobs);
        printf(" apparent,\n");

        printf("Goes to (%0.2lf, %0.2lf, %0.2lf) actual",
               xact, yact, zact);

        if (tecplot == 1)
          fprintf(fptec, "%lf %lf %lf\n", xact*calib,
                  yact*calib, zact*calib);
      }
/*************************************************************/
/*     Here is where the "time residual" is defined for each    */
/*     point entered, corresponding to the raster motion of the */
/*     video capture.                                          */
/*     This assumes that the screen rastering is done in the    */
/*     following way: 4 passes, each the height of 1/4 of the   */
/*     screen size, starting at the top of the screen.          */
/*     (all vertical elements in an individual 1/4 screen pass  */
/*     are acquired simultaneously--parallel factor of 16,      */
/*     if the screen is 64 pixels high.)                        */
/*************************************************************/
      w = yobs; v = 0;
      while (w+(SCR_HEIGHT/4.0) < SCR_HEIGHT) {
        w = w + (SCR_HEIGHT/4.0); v = v + 1; }
      r[mx][nx] = (1 / frate) * (0.25) * (v + xobs / SCR_WIDTH);

      if (nx == p) {
        if (strcmp(rdtype, "f") != 0) {
          printf("\n\n\tREDO THIS FRAME? (enter y for yes): ");
          scanf("%s", achar); }
        if (!strcmp(achar, "y"))
          nx = 0; }
    }


  }
/*************************************************************/
/*     This merely calculates the total # of points given.      */
/*************************************************************/
  for (count = 1; count <= m; count++)
    numpoints += n[count];

/*************************************************************/
/*     Now the initial rough nucleation site estimate is made   */
/*************************************************************/

/*************************************************************/
/*     Function "get_center" takes arguments of the "key"       */
/*     interface points and returns the rough center of the     */
/*     best-fit circle, as fitted to the points in the "view"   */
/*     reference frame.                                         */
/*************************************************************/
  system("clear");
  centcoords = get_center (&xcoc[0][0], &ycoc[0][0],
                           n[key], 1, radobs);
  bx1 = centcoords.bx; by1 = centcoords.by;
  rad_curv1 = centcoords.rad_curv;
  printf("\nCenter of curvature for key interface");
  printf(" at (%.31f, %.31f)\n",
         bx1, by1);
  printf("  --with radius of curvature of %.21f pixels.\n\n",
         rad_curv1);
/*************************************************************/
/*     We do this for the [key+1] frame as well:                */
/*************************************************************/
```

```
  if (n[key+1] > 0) {
    centcoords = get_center (&xcoc[0][0], &ycoc[0][0],
                             n[key+1], 2, radobs);
    bx2 = centcoords.bx; by2 = centcoords.by;
    rad_curv2 = centcoords.rad_curv;
    printf("Center of curvature for (key+1) interface at (%.31f, %.31f)\n",
           bx2, by2);
    printf("  --with radius of curvature of %.21f pixels.\n\n",
           rad_curv2); }

/*************************************************************/
/*     Now time to analyze the relative motion of               */
/*     the interface.  First, I'll choose the target center as  */
/*     the one which had the lesser radius of curvature.        */
/*************************************************************/
  if (rad_curv2 > rad_curv1) {
    printf("Therefore, growth is outward as time progresses...\n\n");
    bx = bx1; by = by1; frame=1;
  }
  else {
    printf("Therefore, growth is inward as time progresses...\n");
    printf("  (point will be opposite of observed center)\n");
    bx = bx2; by = by2; frame=2;
  }

  bxobs = bx; byobs = by;

/*************************************************************/
/*     Remember here to put bx and by in the "droplet" frame,   */
/*     not the "observed" frame, for calculations.              */
/*************************************************************/
  bx = return_xact (bx, xcent, radact, radobs);
  by = return_yact (by, ycent);

/*************************************************************/
/*     Next, calculate whether the center of curvature point    */
/*     that was returned is in fact located within the bounds   */
/*     of the visible droplet surface.                          */
/*************************************************************/

  zcheck = get_zval (FRONT, radact, bx);

  if (zcheck == NEG_ROOT) {
    printf("  ** Screen center of key/key+1 frame off of droplet.\n");
    printf("     Interpolating to droplet surface ..\n");
/*************************************************************/
/*     If the point is outside the visible surface, then start  */
/*     from the center point and work your way out along the    */
/*     line toward the outlying point until you come within     */
/*     a small distance of the droplet surface.                 */
/*************************************************************/
    xvid = bx; yvid = by;
    for (ratio = .1; ratio > .001; ratio *= .1) {
      zcheck = 1;
      while (zcheck != NEG_ROOT) {
        xvid += ratio * (bx - xcent); yvid += ratio * (by - ycent);
        zcheck = get_zval (FRONT, radact, xvid),
      }
      xvid -= ratio * (bx - xcent); yvid -= ratio * (by - ycent);
    }
    bx = xvid; by = yvid;
  }

  bz = get_zval (FRONT, radact, bx);

/*************************************************************/
/*     Now we've got our point on the droplet.  However, the    */
/*     REAL rough nucleation point may be this one, or the one  */
/*     diametrically opposite it.  We can determine which it is */
/*     by looking at the rad_curv seen in (key) and (key+1).    */
/*     If R(key) < R(key+1), it means that the interface was    */
/*     expanding between the frames, and our guess at the       */
/*     nucleation point is correct.  If not, then the opposite  */
/*     point is the correct one.                                */
/*************************************************************/
  if (frame == 2) {
/*************************************************************/
/*     Desired point is opposite calculated one.  So, we're     */
/*     changing the x/y/z values to be opposite.  There really  */
/*     isn't a precise "opposite" for the y value, unless the   */
/*     actual cylinder is perfect and the ycent value given is  */
/*     exact; however, (-y) will be a good enough start.        */
/*************************************************************/
    bx = -bx; by = -by; bz = -bz;
/*************************************************************/
/*     These, recall, are in the "droplet" frame, so to speak.  */
/*     We'll switch to "observed" frame when we tell the user   */
/*     what the verdict is:                                     */
/*************************************************************/
  }
  bxobs = return_xobs (bx, xcent, radact, radobs);
  byobs = return_yobs (by, ycent);
  bzobs = get_zval (FRONT, radobs, bxobs - xcent);

  printf("Assumed nucleation point is (%.3lf, %.3lf, %.3lf)\n",
         bxobs, byobs, bzobs);

/*************************************************************/
/*     Next, we calculate the revised radii of curvature of the */
/*     two frames' interfaces with respect to the proper guess  */
/*     of the nucleation point.                                 */
/*************************************************************/
  p = n[key]; rad_curv1 = 0;
  for (count = 1; count <= p; count++)
    rad_curv1 += sqrt(pow((x[key][count] - bx), 2) +
                      pow((y[key][count] - by), 2));
  rad_curv1 = rad_curv1 / (double)(p);

  p = n[key+1]; rad_curv2 = 0;
  for (count = 1; count <= p; count++)
    rad_curv2 += sqrt(pow((x[key+1][count] - bx), 2) +
                      pow((y[key+1][count] - by), 2));
  rad_curv2 = rad_curv2 / (double)(p);

/*************************************************************/
/*     Now, the initial rough guess of velocity and time delay  */
/*     (between nucleation and first frame) are assigned.       */
/*     Note that this doesn't take raster delay into effect.    */
/*     This will be accounted for in the fine calculations.     */
/*************************************************************/
/*************************************************************/
/*     The following expresses the velocity in pixels/sec, by   */
/*     dividing the change in radius from one frame to the next */
/*     by the time between the two frames.  This velocity isn't */
/*     accurate in any physical sense, but it is helpful in the */
/*     determination of the delay time, assuming that the       */
/*     nucleation point was visible on the surface of the drop. */
/*      If not, the delay time determined will be much too      */
/*     small, which is not a problem when it comes to the       */
/*     iteration step.                                          */
/*************************************************************/
  veloc = fabs(rad_curv2 - rad_curv1)/
    ((vid[key+1] - vid[key])*(1 / frate));
/*************************************************************/
/*     "bt" will be the delay time, the time between the        */
/*     nucleation event and the recording of the first pixel of */
/*     the first observed frame.  If the first frame also       */
/*     happens to be the "key" frame (as is usually the case),  */
/*     then this reduces to                                     */
/*       bt = (rad_curv1 / veloc)                               */
/*************************************************************/
  bt = (rad_curv1 / veloc) - ((1 / frate)*(vid[key] - vid[1]));
  printf("Guess of time between nucleation and first");
  printf("frame: %.3lf frames\n", bt*frate);

/*************************************************************/
/*     User-defined corrections:  this section gives the user a */
/*     chance to give his or her own set of initial values to   */
/*     the algorithm, if the ones determined by the computer    */
/*     don't look good.  Here is also where the user specifies  */
/*     a "bulk" nucleation search (nucleation can be within the */
/*     bulk of the droplet), or else a "surface" search (value  */
/*     of z is constrained to remain on surface, though x and   */
/*     y may vary.                                              */
/*************************************************************/
  printf("\n\n--------------------------------------------------\n");
  printf("  If you want NO changes to any of these values,\n");
  printf("  enter 1 now; otherwise, enter 0:  ");
  scanf("%d", &c_nc);
  if (c_nc == 0) {
    printf("\n\n*** You'll now be prompted for possible changes;\n");
    printf("\tat most prompts, an entry of -1 means");
    printf(" that you accept the default given.\n\n");

    printf("First, is the starting point for the search\n");
    printf("to be a (s)urface or (b)ulk point?  ");
    scanf("%s", s_or_b);

    printf("\n\tNew x (default is %.3lf):  ", bxobs);
    scanf("%lf", &xreplac);
    if (xreplac != -1) {
      bx = return_xact (xreplac, xcent, radact, radobs);
      bxobs = return_xobs (bx, xcent, radact, radobs);
    }

    printf("\tNew y (default is %.3lf):  ", byobs);
    scanf("%lf", &yreplac);
    if (yreplac != -1) {
      by = return_yact (yreplac, ycent);
      byobs = return_yobs (by, ycent);
    }

    if (strcmp(s_or_b, "s") == 0) {
      printf("Is nucleation site on front (default 0) or back (1)?  ");
      scanf("%d", &frontback);
      bz = get_zval (frontback, radact, bx);
      bzobs = get_zval (frontback, radobs, bxobs - xcent);
    }
    else {
      printf("\tNew z (there is no default):  ");
      scanf("%lf", &bz);
      bzobs = bz;
    }

    printf("\tNew time value (default is %.3lf):  ", bt*frate);
    scanf("%lf", &treplac);
    if (treplac != -1)
      bt = treplac/frate;
  }


  printf("\nYou have selected point (%.3lf, %.3lf, %.3lf), %.3lf\n",
         bxobs, byobs, bzobs, bt*frate);
/************************************************************/
/*     Conforming to the structure of the "ellipsoid" version   */
/*     of this program, I will rename these variables "xold,"   */
/*     "yold," "zold," and "told."                              */
/************************************************************/
  xold = bx; yold = by; zold = bz; told = bt;
  printf("In droplet coords: (%.3lf, %.3lf, %.3lf)\n",
         xold, yold, zold);
/************************************************************/
/*     User has the option of producing a std. dev. versus time */
/*     plot for the given (x,y,z) point instead of iterating,   */
/*     which can sometimes be helpful.                          */
/************************************************************/
  printf("\n\nWould you like to (c)ontinue with the normal iteration\n");
  printf("routine, or (p)roduce std dev vs. time data for this point?  ");
  scanf("%s", i_or_p);
  if (strcmp(i_or_p, "c") == 0) {
    printf("\nAllow (t)ime to vary, or (h)old time constant?  ");
    scanf("%s", t_or_h);
    printf("\n(s)urface or (b)ulk nucleation?  ");
    scanf("%s", nuctype); }
  else {
/************************************************************/
/*     Goes to the "create_timedev" function, which produces    */
/*     the desired datafile.                                    */
/************************************************************/
    random_val = create_timedev (TIMEDEV, 0, m, n, xold, yold,
                                 zold, &x[0][0], &y[0][0], &z[0][0],
                                 &r[0][0], frate, vid, calib, fpout);
    exit(0);
  }

/*  SEARCH ROUTINE BEGINS HERE  */
/************************************************************/
/*     Plan: perform several complete cycles of iteration over  */
/*     all of the dimensions (x, y, z, t).  Begin with the time */
/*     iteration, since we know that it will be off even if the */
/*     point coordinates happen to be exactly right.  This is   */
/*     because the initial time guess was obtained in 2-D space */
/*     and did not take into account the droplet's curvature.   */
/************************************************************/

  strcpy(cont_scan, "y");
  if (strcmp(nuctype, "s") == 0)
    ntype = SURFACE;
  else
    ntype = BULK;
  printf("\n\n");
  fprintf(fpout, "Initial coordinates, time:\n");
  fprintf(fpout, "(%.3lf, %.3lf, %.3lf), %.3lf frames\n",
          bxobs, byobs, bzobs, told*frate);

/************************************************************/
/* First call is just to get initial conditions for printout;   */
/*  "itercoords" will have no real value on this call.          */
/************************************************************/
  itercoords = gen_iterate (INIT_ITER, ntype, xold, yold, zold,
                            &x[0][0], &y[0][0], &z[0][0],
                            told, &r[0][0], m, n, radact,
                            frate, calib, fpout, vid);
  while (strcmp(cont_scan, "y") == 0) {
    printf("Iterating...\n");
    for (count = 1; count <= 1000; count++) {
      fprintf(fpout, "\nITERATION CYCLE #%d\n",
              (count + 1000*hundred_cycles));
      for (ittype = TITER; ittype <= REP_ITER; ittype++) {
/************************************************************/
/*     Here, we skip either 1) the time iteration, if the user  */
/*     has requested to hold time constant; or 2) the z value   */
/*     iteration, if the user has decided to restrict the       */
/*     nucleation point to the "round" surface of the cylinder. */
/************************************************************/
        if ((ittype == TITER) && (strcmp(t_or_h, "h") == 0)) {
          ittype++; }
        if ((ittype == ZITER) && (ntype == SURFACE)) {
          ittype++; }

        printf("\nITTYPE: %s iteration\n", get_ittype(ittype));
        printf("Original coords: (%2.3lf, %2.3lf, %2.3lf)",
               xold, yold, zold);
        printf(" / %.3lf frames delay\n", told*frate);
      itercoords = gen_iterate (ittype, ntype, xold, yold, zold,
                               &x[0][0], &y[0][0], &z[0][0],
                               told, &r[0][0], m, n, radact,
                               frate, calib, fpout, vid);
      if (ittype != REP_ITER) {
        xnew = itercoords.xn; ynew = itercoords.yn;
        znew = itercoords.zn; tnew = itercoords.tn;

        printf("New coords: (%2.3lf, %2.3lf, %2.3lf)",
               xnew, ynew, znew);
        printf(" / %.3lf frames delay\n", tnew*frate);

        if (was_change (xold, yold, zold, told,
                        xnew, ynew, znew, tnew)) {
          make_report (ittype, xnew, ynew, znew, tnew,
                       xcent, ycent, frate, fpout,
                       radact, radobs);

          xold = xnew; yold = ynew; zold = znew; told = tnew;
        }
        else {
          fprintf(fpout, "  (no change during this iteration)\n");
          change_total ++;
        }
      }
    }
/*************************************************************/
/*     THIS section is the "escape condition," i.e., "when is   */
/*     it done iterating?"  It is certainly "done" if the       */
/*     routine goes through a complete (x,y,z,t) cycle without  */
/*     generating any new guess of the nucleation point or      */
/*     time delay.  That's what the following says: if an       */
/*     entire cycle of "no changes" has occurred, it is done.   */
/*************************************************************/
    if (((strcmp (t_or_h, "t") == 0) &&
        (((change_total == 4) && (ntype == BULK)) ||
         ((change_total == 3) && (ntype == SURFACE)))) ||
       ((strcmp (t_or_h, "h") == 0) &&
        (((change_total == 3) && (ntype == BULK)) ||
         ((change_total == 2) && (ntype == SURFACE))))) {
      printf("++++++++++++++++++++++++++++++++++++++++++++\n");
      printf("The iteration procedure has reached the optimum\n");
      printf("point for every dimension.  The result is: \n\n");
      fprintf(fpout, "++++++++++++++++++++++++++++++++++++++++++++\n");
      fprintf(fpout, "The iteration procedure has reached the optimum\n");
      fprintf(fpout, "point for every dimension.  The result is: \n\n");
      make_report (FITER, xnew, ynew, znew, tnew,
                   xcent, ycent, frate, fpout,
                   radact, radobs);
      make_report (MITER, xnew, ynew, znew, tnew,
                   xcent, ycent, frate, fpout,
                   radact, radobs);
      fprintf(fpout, "\nCorresponding droplet coords (mm) are:\n");
      fprintf(fpout, "(%.3lf, %.3lf, %.3lf)\n",
              xnew*calib, ynew*calib, znew*calib);
/*************************************************************/
/*     The next two commands exist only so that the velocity    */
/*     array, deviation, and avg. velocity are printed out.     */
/*************************************************************/
      avevel = fill_vels (vel, fvel, m, n, xnew, ynew, znew,
                          &x[0][0], &y[0][0], &z[0][0], tnew,
                          &r[0][0], numpoints, frate, vid);
      presdev = get_frame_dev (PRINTOUT, fvel, avevel, m, calib,
                               fpout);

      if (ntype == BULK) {
        rsurface = sqrt(pow(xnew, 2) + pow(znew,2));
        printf("NOTE:  Final point is %.3lf mm below the\n",
               (radact - rsurface)*calib);
        printf("       surface of the specimen.\n");
        fprintf(fpout, "\nNOTE:  Final point is %.3lf mm below the\n",
                (radact - rsurface)*calib);
        fprintf(fpout, "       surface of the specimen.\n");
      }
      printf("\nEntire search summary printed in file \"%s.\"\n",
             outputfile);

      fclose(fpout);
      exit(1);
    }
    change_total = 0;

  }
  hundred_cycles++;
  itercoords = gen_iterate (MID_ITER, ntype, xnew, ynew, znew,
                            &x[0][0], &y[0][0], &z[0][0],
                            tnew, &r[0][0], m, n, radact,
                            frate, calib, fpout, vid);
  make_report (MITER, xnew, ynew, znew, tnew,
               xcent, ycent, frate, fpout, radact, radobs);
  printf("\n*****************************************\n");
  printf("The iteration has reached 1000 cycles without\n");
  printf("reaching a best point in all dimensions.\n\n");
  printf("\nAnother 1000 cycles? (y/n):  ");
  scanf("%s", cont_scan);
 }

 if (tecplot == 1)
   fclose(fptec);
 printf("\n\n");

}
/*************************************************************/
/*     ************  END OF MAIN  ***************             */
/*************************************************************/

/*************************************************************/
/*     From here out, there are only definitions of special     */
/*     functions which were called from the main body of the    */
/*     program:                                                 */
/*************************************************************/

/*************************************************************/
/*     This function returns a rough, (x,y) center of           */
/*      curvature, given a spread of points in x and y.          */
/*       The function is about the first thing that I wrote, and */
/*      it's a bit rough, I think.  It works, though, and I      */
/*      don't really feel like optimizing it.                    */
/**************************************************************/
Cent_Coords get_center (double *x, double *y,
                        int n, int frame, int radobs)
{
  double xi, yi, m, minv, xprev, yprev;
  double step, delx, dely, ave=0;
  double spreadprev, spreadpos, spreadneg;
  double xpos, ypos, xneg, yneg;
  Cent_Coords centcoords;
  int count;

/**************************************************************/
/*      This calculates the average of the first and last        */
/*      interface point given in the vector of points.           */
/**************************************************************/
  xi = (*(x + PMAX*frame + 1) + *(x + PMAX*frame + n)) / 2;
  yi = (*(y + PMAX*frame + 1) + *(y + PMAX*frame + n)) / 2;

/**************************************************************/
/*      Now we get the slope and the inverse slope.              */
/*      Bear in mind that m, the slope we are interested in, is  */
/*      the slope of the line normal to the line which connects  */
/*      our two points given.                                    */
/**************************************************************/
  if (xi != *(x + PMAX*frame + 1)) {
    minv = (yi - *(y + PMAX*frame + 1))/(xi - *(x + PMAX*frame + 1));
    m = -1/minv; }
  else {
    minv = INF;
    m = 0; }
  xprev = xi; yprev = yi;
  spreadprev = spread (x, y, n, frame, xi, yi);

if (fabs(m) <= 1) {
/**************************************************************/
/*      Base steps on the x axis (so x step is not too large)    */
/**************************************************************/
  for (step = 0.1; step > .0001; step *= .1) {
    xpos = xprev * (1 + 0.1*step);
    ypos = yprev + 0.1*m*xprev*step;
    xneg = xprev * (1 - 0.1*step);
    yneg = yprev - 0.1*m*xprev*step;
    spreadpos = spread (x, y, n, frame, xpos, ypos);
    spreadneg = spread (x, y, n, frame, xneg, yneg);

    if (spreadpos < spreadprev) {
/**************************************************************/
/*      If stepping in the positive direction yields a lower     */
/*      standard dev value (more equidistant from all the        */
/*      interface points), we'll step in the positive direction  */
/*      successively until this is no longer the case--getting   */
/*      progressively closer to the center of curvature as       */
/*      the "step" value decreases.                              */
/**************************************************************/
      while (spreadpos < spreadprev) {
        xprev = xpos; yprev = ypos;
        spreadprev = spreadpos;
        xpos = xprev * (1 + step); ypos = yprev + m*xprev*step;
        spreadpos = spread (x, y, n, frame, xpos, ypos);
/**************************************************************/
/*      This line checks to see if we have definitely wandered   */
/*      the observed surface of the droplet without finding a    */
/*      center of curvature.  If this is true, it means that we  */
/*      might as well quit right here and return the present     */
/*      guess as the "best" guess...                             */
/**************************************************************/
        if (sqrt(pow((xprev-xi), 2) + pow((yprev-yi), 2)) > 2*radobs) {
          spreadpos = INF; } } }
    else if (spreadneg < spreadprev) {
/**************************************************************/
/*     But, if stepping in the NEGATIVE direction yields a lower */
/*      standard dev value (more equidistant from all the        */
/*      interface points), we'll step in the negative direction  */
/*      successively until this is no longer the case.           */
/**************************************************************/
      while (spreadneg < spreadprev) {
        xprev = xneg; yprev = yneg;
        spreadprev = spreadneg;
        xneg = xprev * (1 - step); yneg = yprev - m*xprev*step;
        spreadneg = spread (x, y, n, frame, xneg, yneg);
        if (sqrt(pow((xprev-xi), 2) + pow((yprev-yi), 2)) > 2*radobs) {
          spreadneg = INF; } } }
    if ((spreadpos == INF) || (spreadneg == INF)) {
      centcoords.bx = xprev; centcoords.by = yprev;
      centcoords.rad_curv = spreadprev;
      return (centcoords); }
  }
}
else {
/**************************************************************/
/*      In this case, base steps on the y axis, since line is    */
/*      most nearly vertical.  This way, x steps won't get       */
/*      too large.                                               */
/**************************************************************/
  for (step = 0.1; step > .0001; step *= .1) {
    ypos = yprev * (1 + 0.01*step);
    xpos = xprev + 0.1*minv*yprev*step;
    yneg = yprev * (1 - 0.1*step);
    xneg = xprev - 0.1*minv*yprev*step;
    spreadpos = spread (x, y, n, frame, xpos, ypos);
    spreadneg = spread (x, y, n, frame, xneg, yneg);

    if (spreadpos < spreadprev) {
      while (spreadpos < spreadprev) {
        xprev = xpos; yprev = ypos;
        spreadprev = spreadpos;
        ypos = yprev * (1 + step);
        xpos = xprev + minv*yprev*step;
        spreadpos = spread (x, y, n, frame, xpos, ypos);
        if (sqrt(pow((xprev-xi), 2) + pow((yprev-yi), 2)) > 2*radobs) {
```

```
        spreadpos = INF; } }}
    else if (spreadneg < spreadprev) {
      while (spreadneg < spreadprev) {
        xprev = xneg; yprev = yneg;
        spreadprev = spreadneg;
        yneg = yprev * (1 - step);
        xneg = xprev - minv*yprev*step;
        spreadneg = spread (x, y, n, frame, xneg, yneg);
        if (sqrt(pow((xprev-xi), 2) + pow((yprev-yi), 2)) > 2*radobs) {
          spreadneg = INF; } } }
    if ((spreadpos == INF) || (spreadneg == INF)) {
      centcoords.bx = xprev; centcoords.by = yprev;
      centcoords.rad_curv = spreadprev;
      return (centcoords); } } }

  for (count = 1; count <= n; count++) {
    ave += sqrt(pow((*(x + PMAX*frame + count) - xprev), 2) +
      pow((*(y + PMAX*frame + count) - yprev), 2)); }
  ave = ave / (double)(n);

  centcoords.bx = xprev; centcoords.by = yprev;
  centcoords.rad_curv = ave;
  return (centcoords);
}

double spread (double *x, double *y, int n, int frame,
               double xcent, double ycent)
{
  int count;
  double dist, ydist, avg = 0, total = 0;

  for (count = 1; count <= n; count++) {
    dist = sqrt(pow((*(x + PMAX*frame + count) - xcent), 2) +
      pow((*(y + PMAX*frame + count) - ycent), 2));
    avg += dist;
  }
  avg = avg / (double)(n);

  for (count = 1; count <= n; count++) {
    dist = pow((*(x + PMAX*frame + count) - xcent), 2) +
                 pow((*(y + PMAX*frame + count) - ycent), 2);
    total += pow((avg - sqrt(dist)), 2); }
  total = sqrt(total / (double)(n-1));

  return (total);
}

/*****************************************************************/
/*     This is the general iteration algorithm which will home   */
/*     in on the best-fit nucleation point coordinates and       */
/*     time between nucleation and first frame.                  */
/*****************************************************************/
Iter_coords gen_iterate (int it_var, int type,
                         double xn, double yn, double zn,
                          double *x, double *y, double *z,
                         double tn, double *r,
                         int frames, int *points, double radact,
                         double frate, double calib,
                         FILE *fpout, double *vid)
{
  Iter_coords coords;
  double delta, step, stepmin, presdev, posdev, negdev, newdev;
  int numpoints = 0, count, sign;
  double vel[PMAX*FMAX], temp_val;
  double fvel[FMAX];
  double avevel = 0, varnew, varprev, zold, znew;
  char itervar[20];
  double presdevnorm, posdevnorm, negdevnorm, newdevnorm;

  for (count = 1; count <= frames; count++)
    numpoints += points[count];

  if (it_var == XITER)
    strcpy(itervar, "X iteration");
  else if (it_var == YITER)
    strcpy(itervar, "Y iteration");
  else if (it_var == ZITER)
    strcpy(itervar, "Z iteration");
  else if (it_var == TITER)
    strcpy(itervar, "Time iteration");
  else if (it_var == INIT_ITER) {
    avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn,
                        x, y, z, tn, r, numpoints, frate, vid);
    presdev = get_frame_dev (FPRINTOUT, fvel, avevel, frames, calib,
                             fpout);
    return (coords); }
  else if (it_var == MID_ITER) {
    avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn,
                        x, y, z, tn, r, numpoints, frate, vid);
    presdev = get_frame_dev (PRINTOUT, fvel, avevel, frames, calib,
                             fpout);
    return (coords); }

  if (it_var != REP_ITER) {
    printf("Doing %s now...\n", itervar);
    fprintf(fpout, "--------------------------------------------\n");
    fprintf(fpout, "Beginning %s now...\n", itervar);
    fprintf(fpout, "--------------------------------------------\n");
  }

/*****************************************************************/
/*     This first bit sets up the values for use in the          */
/*     "best-point-search" iteration routine, depending on       */
/*     whether the search is in time (TITER) or space (any of    */
/*     the others).                                              */
/*                                                               */
/*     -- "delta" is the initial small deviation to be used to   */
/*        determine which direction (+ or -) is the way to go    */
/*     -- "step" is the intial value which is added or           */
/*        subtracted from the search variable.  It cannot be     */
/*        too large, or there is danger of stepping past the     */
/*        the local minima we are searching for.                 */
/*     -- "stepmin" is the limit on the resolution of the        */
/*        search.  It can be set any any desired level, but      */
/*        should be relatively small.                          */
/***************************************************************/

  if (it_var != TITER) {
    delta = 1e-6 * radact;
    step = 0.01 * radact;
    stepmin = 1e-6 * radact; }
  else {
    delta = 1e-6 / frate;
    step = 0.01 / frate;
    stepmin = 1e-4 / frate; }

/***************************************************************/
/*     The following is the guts of the routine which tests to  */
/*     see in which direction the iteration variable of choice  */
/*     should change in order to lower some standard deviation  */
/*     of velocity values of the interface points with respect  */
/*     to the assumed nucleation site and delay time.           */
/*                                                              */
/*     Many "if" statements are needed in order to deal with    */
/*     the four possible iteration variables: x, y, z, and t.   */
/***************************************************************/

/***************************************************************/
/*     First, an array is filled with the values of the        */
/*     velocities required to produce each interface point,    */
/*     assuming the initial nucleation point and time.         */
/*      The initial standard deviation of the velocities is    */
/*     also calculated and called "presdev."  This value is    */
/*     then normalized over the average velocity, to ensure    */
/*     that no preferential treatment is given to smaller      */
/*     velocities (and, hence, smaller std. devs)              */
/***************************************************************/

/*  DEBUGGING PRINTS
  printf("Initial ave args are: frames=%d,\n",
         frames);
  printf(" (xn, yn, zn)=(%lf,%lf,%lf),\n",
         xn, yn, zn);
  printf(" tn=%lf, numpoints=%lf, frate=%lf, vid=%lf\n",
         tn, numpoints, frate, vid);
  printf("Initial dev args are: avevel=%lf\n",
         avevel);
  printf(" frames=%d, calib=%lf\n", frames, calib);
*/

  avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn,
                      x, y, z, tn, r, numpoints, frate, vid);
  presdev = get_frame_dev (NORMAL, fvel, avevel, frames, calib,
                           fpout);
  presdevnorm = presdev / avevel;

/*  printf("Initial standard dev is %lf\n", presdev);  */

  if (it_var == REP_ITER) {
    temp_val = get_frame_dev (FPRINTOUT, fvel, avevel, frames, calib,
                              fpout);
```

```
    return (coords); }

/*  Here is the "old" standard deviation method, calculating
    the value with respect to each and every point.
  presdev = get_dev (vel, avevel, numpoints, tn);
*/

/***************************************************************/
/*     Next comes the "probe" in the positive direction.  The   */
/*     new values for the velocity array and deviation are      */
/*     determined in the case when the iteration variable is    */
/*     INCREASED ever so slightly (by "delta").                 */
/***************************************************************/
  if (it_var == XITER)
    avevel = fill_vels (vel, fvel, frames, points, xn+delta, yn, zn,
                        x, y, z, tn, r, numpoints, frate, vid);
  else if (it_var == YITER)
    avevel = fill_vels (vel, fvel, frames, points, xn, yn+delta, zn,
                        x, y, z, tn, r, numpoints, frate, vid);
  else if (it_var == ZITER)
    avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn+delta,
                        x, y, z, tn, r, numpoints, frate, vid);
  else if (it_var == TITER)
    avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn, x,
                        y, z, tn+delta, r, numpoints, frate, vid);
  posdev = get_frame_dev (NORMAL, fvel, avevel, frames, calib,
                          fpout);
  posdevnorm = posdev / avevel;

/*  printf("posdev is %lf\n", posdev);  */

/***************************************************************/
/*     Now the same thing is done again, only the "probe"       */
/*     a distance "delta" into the negative direction.          */
/***************************************************************/
  if (it_var == XITER)
    avevel = fill_vels (vel, fvel, frames, points, xn-delta, yn, zn,
                        x, y, z, tn, r, numpoints, frate, vid);
  else if (it_var == YITER)
    avevel = fill_vels (vel, fvel, frames, points, xn, yn-delta, zn,
                        x, y, z, tn, r, numpoints, frate, vid);
  else if (it_var == ZITER)
    avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn-delta,
                        x, y, z, tn, r, numpoints, frate, vid);
  else if (it_var == TITER)
    avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn, x,
                        y, z, tn-delta, r, numpoints, frate, vid);
  negdev = get_frame_dev (NORMAL, fvel, avevel, frames, calib,
                          fpout);
  negdevnorm = negdev / avevel;

/*  printf("negdev is %lf\n", negdev);  */

/*
  printf("  Average velocity: %.3lf pixels/sec\n", avevel);
  printf("  posdev: %lf   negdev: %lf   presdev: %lf\n",
         posdev, negdev, presdev);
```

```
*/

/*************************************************************/
/*     At this point, a decision can be made regarding which    */
/*     direction IMPROVES the overall fit.  We go in the        */
/*     direction of minimum normalized standard deviation, and  */
/*     if the                                                   */
/*     present position is the minimum, we know that we've      */
/*     determined the optimum point at this "step" resolution,  */
/*     and we can choose to increase the search resolution or   */
/*     else quit the search and return the optimum point.       */
/*************************************************************/
  if ((posdevnorm < presdevnorm) || (negdevnorm < presdevnorm)) {
    if (posdevnorm < negdevnorm) {
/*      printf("Positive is the way to go.\n");    */
      sign = 1; }
    else {
/*      printf("Negative is the way to go.\n");     */
      sign = -1; }

/*************************************************************/
/*     Here, the iteration variable is changed to a value which */
/*     lies a distance "step" in the direction of improvement.  */
/*                                                              */
/*     This new value is cranked through the system in order    */
/*     to determine whether or not the "step" taken was too     */
/*     large or not.  The hope is that there exists a local     */
/*     minimum in the standard dev. variation, and that we can  */
/*     eventually reach that local minimum.                     */
/*                                                              */
/*     Note:  special steps must be taken to force the          */
/*     y variable to remain on the droplet surface, in the      */
/*     case of a "SURFACE" search.                              */
/*************************************************************/

    while (step >= stepmin) {
      if (it_var == XITER) {
        varnew = xn + sign*step;
        if (type != BULK) {
          zold = zn;
          if ((pow(radact, 2) - pow(varnew, 2)) < 0)
/*************************************************************/
/*     In this case, we have stepped "off the edge of the       */
/*     specimen."  In order to avoid errors, I'll just force    */
/*     the x and z coords to stay right where they are.         */
/*************************************************************/
            varnew = xn;
          else {
            znew = sqrt(pow(radact, 2) - pow(varnew, 2));

/* Maintaining z coord on back of specimen, if it was previously;
    this implies that the nucleation position cannot move from the
    back to the front of the specimen, or vice-versa, during
    iteration.  The user must try separate back and front points,
    if the position of the nucleation might be on either surface:  */
            if (zold < 0)
              znew *= -1;
            zn = znew;
          }
        }

        avevel = fill_vels (vel, fvel, frames, points, varnew, yn, zn,
                            x, y, z, tn, r, numpoints, frate, vid);

/*
        printf("Initial ave args are: frames=%d,\n",
               frames);
        printf(" (xn, yn, zn)=(%lf,%lf,%lf),\n",
               xn, yn, zn);
        printf(" tn=%lf, numpoints=%lf, frate=%lf, vid=%lf\n",
               tn, numpoints, frate, vid);
        printf("Initial dev args are: avevel=%lf\n",
               avevel);
        printf(" frames=%d, calib=%lf\n", frames, calib);
*/

      }
      else if (it_var == YITER) {
        varnew = yn + sign*step;
        avevel = fill_vels (vel, fvel, frames, points, xn, varnew, zn,
                            x, y, z, tn, r, numpoints, frate, vid);
      }
      else if (it_var == ZITER) {
        varnew = zn + sign*step;
        avevel = fill_vels (vel, fvel, frames, points, xn, yn, varnew,
                            x, y, z, tn, r, numpoints, frate, vid);
      }
      else { /*  IF it_var == TITER  */
        varnew = tn + sign*step;
        if (varnew > TIMEMAX) {
          printf("Iterated delay time has exceeded .025 seconds!\n");
          printf("Try a different start point for the iteration,\n");
          printf(" or else perform just a 'surface' iteration.\n\n\n");
          exit(0);
        }
        avevel = fill_vels (vel, fvel, frames, points, xn, yn, zn, x,
                            y, z, varnew, r, numpoints, frate, vid);
      }

      newdev = get_frame_dev (NORMAL, fvel, avevel, frames, calib,
                              fpout);
      newdevnorm = newdev / avevel;

/*      printf("newdev is %lf\n", newdev);  */

/*************************************************************/
/*     If the new norm. std. dev. is less than the old, we know */
/*     that our "step" did in fact get us closer to the minimum */
/*     without causing us to overshoot that minimum.  This new  */
/*     value of the variable becomes the accepted "best" one.   */
/*************************************************************/
      if (newdevnorm < presdevnorm) {
/*        printf("** Stepped by %g\n", sign*step);  */
        if (it_var == XITER)
        xn = varnew;
      else if (it_var == YITER)
        yn = varnew;
      else if (it_var == ZITER)
        zn = varnew;
      else /*  IF it_var == TITER  */
        tn = varnew;

/*
      printf("Better guess: (%.3lf, %.3lf, %.3lf), ",
             xn, yn, zn);
      printf(" time %g sec\n", tn);
*/

      presdevnorm = newdevnorm; }
/**************************************************************/
/*     Otherwise, our latest "step" has taken us too far, past  */
/*     the local minimum, and we must retain the original       */
/*     "best" value and try a smaller "step."                   */
/**************************************************************/
     else {
/*     printf("** Step size diminished...\n");  */
/**************************************************************/
/*     Must reset the value of "zn," which has been changed to  */
/*     a "test" value when we restrict the point to lie on the  */
/*     surface and vary x or y.                                 */
/**************************************************************/
      if ((type != BULK) && (it_var == XITER))
        zn = zold;
      step *= 0.1;
     }
   }
  }

/**************************************************************/
/*     Arrival at this point in the function means that the     */
/*     search algorithm has determined the local minimum of     */
/*     velocity variation with respect to the search variable,  */
/*     to the resolution of "stepmin."  We have a new           */
/*     "best guess" point.                                      */
/**************************************************************/

/*
  if (it_var == YITER)
    step = get_frame_dev (FPRINTOUT, fvel, avevel, frames, calib,
                          fpout);
*/

  coords.xn = xn; coords.yn = yn; coords.zn = zn;
  coords.tn = tn;

  return (coords);

}

/**************************************************************/
/*     This routine does several things.  First, it calculates  */
/*     the apparent velocity observed at each of the interface  */
/*     points given.  It then stocks the velocity array with    */
/*     these values--one for each point.  Finally, it           */
/*     determines the average velocity overall, and returns it. */
/*                                                              */
/*     Actually, this routine was changed to help out with the  */
/*     "correct" standard deviation measurement.  I'll retain   */
/*     the array with the velocity of each point, but I'll also */
/*     use the function to create an array of average frame     */
/*     velocities (this is "fvel," as opposed to "vel."         */
/**************************************************************/
double fill_vels (double *vel, double *fvel, int frames, int *points,
                  double xn, double yn, double zn,
                  double *xp, double *yp, double *zp,
                  double tn, double *r, int numpoints,
                  double frate, double *vid)
{
  double avevel = 0; double favevel = 0;
  int fcount, pcount, index = 1;
  double dist, time;
  double x, y, z, t;

/**************************************************************/
/*     Here's where we fill up the velocity array with values   */
/*     for each frame.      Recall that "vid" contains the      */
/*     video frame index for each frame that was entered.       */
/**************************************************************/

  for (fcount = 1; fcount <= frames; fcount++) {
    for (pcount = 1; pcount <= points[fcount]; pcount++) {
      x = *(xp + (PMAX)*fcount + pcount);
      y = *(yp + (PMAX)*fcount + pcount);
      z = *(zp + (PMAX)*fcount + pcount);
      t = *(r + (PMAX)*fcount + pcount);

      dist = sqrt(pow((x - xn), 2) + pow((y - yn), 2) +
                  pow((z - zn), 2));
      time = tn + (vid[fcount] - vid[1])*(1/frate) + t;
      vel[index] = dist / time;
      avevel += vel[index];
      favevel += vel[index];
      index++;

/*
      printf("Frame %d; point %d: (%.2lf, %.2lf, %.2lf); r.d. %.3lf frames\n",
             fcount, pcount, x, y, z, (t*40500.0));
      printf("\tDistance: %.2lf      Time: %g      Velocity: %g\n",
             dist, time, vel[index-1]);
*/

    }
    favevel /= points[fcount];
    fvel[fcount] = favevel;
    favevel = 0;
  }

/*
```

```
  printf("\nNucleation point: (%.21f, %.21f, %.21f)\n",
         xn, yn, zn);
*/

  avevel /= (double)(numpoints);

/*
  printf("    Average velocity for this case: %g pixels/sec\n",
         avevel);
*/

  return (avevel);
}

/***************************************************************/
/*     We experimented with two different methods of           */
/*     determining a "best" point via some standard deviation  */
/*     calculation.  The first uses the velocity of EVERY POINT */
/*     for its standard deviation calculation.  We have        */
/*     discarded this method for several reasons:  Foremost,   */
/*     it favors the frames with more points over the frames   */
/*     with less points (which are likely to be the smaller,   */
/*     more accurate interfaces, anyway!)                      */
/***************************************************************/
/***************************************************************/
/*     This routine takes as its input the array of velocity   */
/*     values for all of the points, as well as the average    */
/*     velocity.  It returns the standard deviation of the     */
/*     velocity with respect to all of the interface points    */
/*     in all of the frames.   We don't actually use this.     */
/***************************************************************/
double get_dev (double *vel, double avevel, int numpoints,
                double time)
{
  double dev = 0;
  int count;

  for (count = 1; count <= numpoints; count++)
    dev += pow((avevel - vel[count]), 2);

  dev = dev / (double)(numpoints);
  dev = sqrt(dev);

  /*  printf(" * Average deviation: %g\n", dev);  */

  return (dev);
}

/***************************************************************/
/*     This is the standard deviation method which seems to    */
/*     make the most sense.  It calculates with respect to the */
/*     standard deviation of the average velocity for each     */
/*     frame.  Thus, if there are four frames, the standard    */
/*     deviation calculation involves four terms--the average  */
/*     velocity of each of the four frames                     */
/***************************************************************/
double get_frame_dev (int type, double *fvel, double avevel,
                      double frames, double calib, FILE *fpout)
{
  double dev = 0;
  int f;
  double fave = 0;

  for (f = 1; f <= frames; f++)
    dev += pow((avevel - fvel[f]), 2);

  dev = dev / (double)(frames);
  dev = sqrt(dev);

  if (type == NORMAL)
    return (dev);
  else if (type == PRINTOUT) {
    printf("\nAverage interface velocities for each frame:\n");
    for (f = 1; f <= frames; f++)
      printf("  Frame #%d: %.31f m/sec\n",
             f, fvel[f]*calib/1000.0);
    printf("\nOverall average velocity (every point): %.31f m/s\n",
           avevel*calib/1000.0);
    printf("** STANDARD DEVIATION:  %.51f\n\n", dev*calib/1000.0);
  }
  else if (type == FPRINTOUT) {
    fprintf(fpout, "\nAverage interface velocities for each frame:\n");
    for (f = 1; f <= frames; f++)
      fprintf(fpout, "  Frame #%d: %.31f m/sec\n",
             f, fvel[f]*calib/1000.0);
    fprintf(fpout, "\nOverall average velocity (every point): %.31f m/s\n",
           avevel*calib/1000.0);
    fprintf(fpout, "*** STANDARD DEVIATION:  %.31f\n\n", dev*calib/1000.0);
  }
}

void make_report (int type, double x, double y, double z, double t,
                  double xcent, double ycent, double frate, FILE *fpout,
                  double radact, double radobs)

{
  Iter_coords coordinates;
  char typename[50];
  double xobs, yobs, zobs;
  int frontback;

  if (z > 0)
    frontback = FRONT;
  else
    frontback = BACK;

  xobs = return_xobs (x, xcent, radact, radobs);
  yobs = return_yobs (y, ycent);
  zobs = get_zval (frontback, radobs, xobs - xcent);

  t = t * frate;

  if (type == TITER)
    strcpy(typename, "Time-iteration");
  else if (type == XITER)
    strcpy(typename, "X-iteration");
  else if (type == YITER)
    strcpy(typename, "Y-iteration");
  else if (type == ZITER)
    strcpy(typename, "Z-iteration");
  else if (type == FITER)
    strcpy(typename, "FINAL_ITERATION");
  else if (type == MITER) {
    printf("Nucleation coords and time after last iteration:\n");
    printf("(%.3lf, %.3lf, %.3lf), %.3lf frames delay\n",
            xobs, yobs, zobs, t);  }
  fprintf(fpout, "Nucleation coords and time after %s:\n", typename);
  fprintf(fpout, "(%.3lf, %.3lf, %.3lf), %.3lf frames delay\n",
          xobs, yobs, zobs, t);
  return;
}

int was_change (double xold, double yold, double zold, double told,
                double xnew, double ynew, double znew, double tnew)
{
  if ((xold == xnew) && (yold == ynew) &&
      (zold == znew) && (told == tnew))
    return (0);
  else
    return (1);
}

double get_zval (int side, double radius, double x)
{
  double z;

  z = pow(radius, 2) - pow(x, 2);

  if (z < 0) {
    /* Coords given cannot lie on droplet --> entry error */
    return (NEG_ROOT);
  }

  z = sqrt(z);
  if (side == FRONT)
    return (z);
  else
    return (-z);
}

/*****************************************************************/
/*     This function either creates a dataset full of times      */
/*     versus the corresponding frame standard deviations,       */
/*     given a certain nucleation point; or else, it iterates    */
/*     through time starting from 0 until it finds the point     */
/*     of minimum standard deviation, and returns that value.    */
/*      The behavior is controlled by the argument "type "  If   */
/*     the type is TIMEDEV, the behavior is the former above;    */
/*     If the type is GETNEWTIME, the behavior is the latter.    */
/*****************************************************************/
double create_timedev (int type, double tguess, int m, int *n,
                       double xold, double yold,
                       double zold, double *x, double *y, double *z,
                       double *r, double frate, double *vid,
                       double calib, FILE *fpout)
{
  double timemin, timemax, delta, avevel, presdev, testtime;
  double besttime=0, bestdev;
  int count, numpoints = 0;
  FILE *fp;
  double velocity[FMAX][PMAX], fveloc[FMAX];

  printf("\nPlease enter minimum, then maximum time values for\n");
  printf("delay/dev data to be considered, in portions of frames:  ");
  scanf("%lf%lf", &timemin, &timemax);
  timemin /= frate; timemax /= frate;

  delta = ((timemax - timemin)/1000.0);
  for (count = 1; count <= m; count++)
    numpoints += n[count];

  if (type == TIMEDEV) {
    printf("\n\nGenerating dataset...\n");
    fp = fopen("timedev.dat", "w");
    if (fp == NULL) {
      printf("Exiting due to NULL pointer to file!\n\n");
      exit(0);
    }
  }

  bestdev = BIG_FLOAT;
  for (testtime = timemin; testtime <= timemax; testtime += delta) {
    avevel = fill_vels (&velocity[0][0], fveloc, m, n, xold, yold,
                        zold, x, y, z,
                        testtime, r, numpoints, frate, vid);
    presdev = get_frame_dev (NORMAL, fveloc, avevel, m, calib,
                             fpout);
    presdev = presdev * calib / 1000.0;
/*    printf("New deviation: %lf\n", presdev);  */
    if (type == TIMEDEV)
      fprintf(fp, "%g\t%g\n", testtime*frate, presdev);
    else {
/* See if new dev value is lower than previous.  If so, the current
   time (minus the delta) is the best local delay time.  */
/*      printf("\nTime: %lf frames:\n", testtime*frate);  */
      if (presdev > bestdev) {
        printf("Old deviation was better...resetting best time.\n");
        besttime = testtime - delta;  }
      else {
/*      printf("New deviation is better; resetting bestdev.\n");  */
        bestdev = presdev; }
    }
  }
  if (type == TIMEDEV) {
    fclose (fp);
    printf("  Data is stored in file \"timedev.dat\".\n\n");
    return (0); }
  else
```

```
    return (besttime);
}

/****************************************************************/
/*     These final functions are pretty clear.  We are working  */
/*      with two frames of reference throughout this program:   */
/*     the "observed" frame, in which x and y vary from 1 to    */
/*     64; and the "actual" frame, where x=0 is set to be the   */
/*     center of the observed cylinder, where z reaches its     */
/*     maximum value of radobs.                                 */
/*      The function switch between these two reference frames. */
/****************************************************************/
double return_xact (double xobs, double xcent, double radact,
                    double radobs)
{
  return ((xobs - xcent)*(radact/radobs));
}

double return_yact (double yobs, double ycent)
{
  return (yobs - ycent);
}

double return_xobs (double xact, double xcent, double radact,
                    double radobs)
{
  return ((xact + xcent*(radact/radobs))*(radobs/radact));
}

double return_yobs (double yact, double ycent)
{
  return (yact + ycent);
}

char * get_ittype (int ittype)
{
  if (ittype == 1)
    return ("Time");
  if (ittype == 2)
    return ("Z-coord");
  if (ittype == 3)
    return ("X-coord");
  if (ittype == 4)
    return ("Y-coord");
  if (ittype == 5)
    return ("Report");
  if (ittype == 6)
    return ("FITER");
  if (ittype == 7)
    return ("MITER");
  if (ittype == 8)
    return ("Initial");
  if (ittype == 9)
    return ("MID_ITER");
  else
    return ("nothing");
}
```

References

1 A. D. Van Riemsdyk: *Ann. Chem. Phys.*, 1880, vol. 20, p. 66.
2 J. L. Walker: *Principles of Solidification*, ed. B. Chalmers, John Wiley and Sons, Inc., New York, 1964, p. 114.
3 G. A. Colligan and B. S. Bayles: Acta. Metall., 1962, vol. 10, pp. 895-897.
4 T. J. Piccone, Y. Wu, Y. Shiohara, and M. C. Flemings: *Solidification Processing 1987*, The Institute of Metals, 1988, pp. 268-270.
5 R. Willnecker, D. M. Herlach, and B. Feuerbacher: *Phys. Rev. Lett.*, 1989, vol. 62 (23), pp. 2707-2710.
6 W. H. Hofmeister, R. J. Bayuzick, and M. B. Robinson: *Rev. Sci. Instrum.*, 1990, vol. 61 (8), pp. 2220-2223.
7 K. Eckler and D. M. Herlach: *Mater. Sci. Eng. A*, 1994, vol. A178, pp. 159-162.
8 B. T. Bassler, W. H. Hofmeister, G. Carro, and R. J. Bayuzick: *Metall. Trans. A*, 1994, vol. 25A, pp. 1301-1307.
9 J. Lipton, W. Kurz, and R. Trivedi: *Acta Metall.*, 1987, vol. 35 (4), pp. 957-964.
10 J. L. Walker: *The Physical Chemistry of Process Metallurgy*, ed. G. R. St. Pierre, Interscience, New York, 1959, Pt. 2, p. 845.
11 G. A. Colligan, V. A. Suprenant, and F. D. Lemkey: *J. Met.*, 1961, vol. 13, p. 691-692.
12 E. Schleip, R. Willnecker, D. M. Herlach, and G. P. Görler: *Mater. Sci. Eng.*, 1988, vol. 98, pp. 39-42.
13 R. E. Cech: Trans. AIME, 1956, vol. 206, pp. 585-589.
14 T. F. Kelly and J. B. VanderSande: *International Journal of Rapid Solidification*, 1987, vol. 3, pp. 51-79.
15 Y. Kim, H. Lin, and T. Kelly: *Acta Metall.*, 1988, vol. 36 (9), pp. 2525-2536.
16 D. J. Thoma and J. H. Perepezko: *Metall. Trans. A*, 1992, vol. 23A, pp. 1347-1362.
17 M. R. Libera, P. P. Bolsaitis, R. E. Spjut, and J. B. VanderSande: *J. Mater. Res.*, 1988, vol. 3 (3), pp. 441-452.
18 Q. Zhao, T. J. Piccone, Y. Shiohara, and M. C. Flemings: *Rapid Quenching and Powder Preparation, Proc. MRS Int. Meeting on Advanced Materials*, 1989, vol. 3, MRS, Pittsburgh, PA, pp. 597-602.
19 D. M. Herlach, B. Feuerbacher, and E. Schleip: *Mater. Sci. Eng. A*, 1991, vol. A133, pp. 795-798.
20 Y. Chuang, K. Hsieh, and Y. Chang: *Metall. Trans. A*,1986, vol. 17A, pp. 1373-1380.
21 M. Barth, K. Eckler, and D. M. Herlach: *Mater. Sci. Eng. A*, 1991, vol. A133, pp. 790-794.
22 T. Z. Kattamis and M. C. Flemings: *Mod. Castings*, 1967, v. 52, pp. 191-198.
23 G. J. Abbaschian and M. C. Flemings: *Metall. Trans. A*, 1983, vol. 14A, pp. 1147-1157.
24 M. J. Haugh: *Theory and Practice of Radiation Pyrometry*, ed. D. P. DeWitt and G. D. Nutter, John Wiley and Sons, Inc., New York, 1988, p. 911.
25 G. P. Ivantsov: *Dokl. Akad. Nauk. SSSR*, 1947, vol. 58, p. 567.
26 M. Schwarz, A. Karma, K. Eckler, and D. M. Herlach: *Phys. Rev. Lett.*, 1994, vol. 73 (10), pp. 1380-1383.
27 K. K. Leung, C. P. Chiu, and H. W. Kui: *Scripta Metallurgica et Materiala*, 1995, vol. 32 (10), pp. 1559-1563.
28 T. J. Piccone: Sc.D. Thesis, Massachusetts Institute of Technology, 1990.
29 W. J. Boettinger and S. R. Coriell: *Science and Technology of the Undercooled Melt*, ed. P. R. Sahm, H. Jones, and C. M. Adam, Martinus Nijhoff, Dordrecht, Germany, 1986, p. 81.
30 M. J. Azia: *J. Applied Phys.*, 1982, vol. 53, p. 1158.
31 D. Turnbull: *Metall. Trans. A*, 1981, vol. 12A, p. 693.