1 of 1

Measurement of Interfacial Tension By Use of
Pendant Drop Video Techniques

Topical Report

By
Melvin D. Herd
Charles P. Thomas
Gregory A. Bala
Gordon D. Lassahn

September 1993

Work Performed Under Contract No. DE-AC07-76ID01570

Prepared for
U.S. Department of Energy
Assistant Secretary for Fossil Energy

Rhonda Lindsey, Project Manager
Bartlesville Project Office
P.O. Box 1398
Bartlesville, OK 74005

Prepared by
Idaho National Engineering Laboratory
EG&G Idaho, Inc.
Idaho Falls, ID 83415

MASTER

EP

ABSTRACT

This report describes an instrument to measure the interfacial tension (IFT) of aqueous surfactant solutions and crude oil. The method involves injection of a drop of fluid (such as crude oil) into a second immiscible phase to determine the IFT between the two phases. The instrument is composed of an AT-class computer, optical cell, illumination, video camera and lens, video frame digitizer board, monitor, and software. The camera displays an image of the pendant drop on the monitor, which is then processed by the frame digitizer board and non-proprietary software to determine the IFT. Several binary and ternary phase systems were taken from the literature and used to measure the precision and accuracy of the instrument in determining IFTs. A copy of the software program is included in the report. A copy of the program on diskette can be obtained from the Energy Science and Technology Software Center, P.O. Box 1020, Oak Ridge, TN 37831-1020.

The accuracy and precision of the technique and apparatus presented is very good for measurement of IFTs in the range from 72 to 10^{-2} mN/m, which is adequate for many EOR applications. With modifications to the equipment and the numerical techniques, measurements of ultralow IFTs ($< 10^{-3}$ mN/m) should be possible as well as measurements at reservoir temperature and pressure conditions.

The instrument has been used at the Idaho National Engineering Laboratory (INEL) to support the research program on microbial enhanced oil recovery (MEOR). Measurements of IFTs for several bacterial supernatants and unfractionated acid precipitates of microbial cultures containing biosurfactants against medium to heavy crude oils are reported.

These experiments demonstrate that the use of automated video imaging of pendant drops is a simple and fast method to reliably determine interfacial tension between two immiscible liquid phases, or between a gas and a liquid phase.

SUMMARY

Nearly two thirds of the known U.S. oil resource remains in the ground after conventional recovery methods have been applied. The remaining oil can be classified as mobile or immobile oil. Mobile oil is remaining oil that is can be displaced by waterflooding or steam injection but is bypassed because of heterogeneities in reservoir rock formations and immobile oil is oil not mobile to such processes. Immobile oil remaining after conventional recovery is trapped in pore structures by viscous and capillary forces and cannot be removed by the pressures generated during waterflooding or steam injection. Chemical and miscible processes are the principal EOR methods for producing immobile light oil.

Immobile oil can be recovered if the ratio of viscous forces to capillary forces (capillary number) is sufficiently high, where viscous forces are determined by the viscosity of the displacing phase times the Darcy velocity of the displacing phase and capillary forces are determined by the interfacial tension (IFT) between the phases. The most effective method to significantly increase the capillary number is to lower the IFT. Ultra-low IFTs ($< 10^{-3}$ mN/m) are usually thought to be necessary to effectively mobilize oil trapped by capillary forces.

A primary goal of the INEL microbial enhanced oil recovery research (MEOR) program is to develop an understanding of the mechanisms by which surfactant producing microorganisms increase oil recovery and to use this improved understanding in the development of microbial systems for field testing. The purpose of developing the IFT apparatus was to determine the effectiveness of microbial surfactants and cosurfactants in lowering IFT between crude oils and the displacing phase.

The ability to digitize video images using personal computers enables the researcher to automate the IFT measurement process. Digital video methods provide a fast, reliable process for determination of IFT. A simple automated system for IFTs using the pendant drop method has been developed at the INEL and applied to measurement of the IFT of bacterial biosurfactants and crude oils. To determine IFT, the size and shape of a transparent or opaque drop of one fluid (e.g., crude oil) immersed in a second transparent fluid (e.g., surfactant solution) is displayed on a monitor with a CCD video camera. The video image on the monitor is digitized and stored by a computer-controlled system. Custom software (developed at the INEL) determines various drop shape measurements from this drop image, and then computes the IFT. The non-proprietary software is listed in Appendix A and B. Copies of the program on diskette can be obtained from the Energy Science and Technology Software Center, P.O. Box 1020, Oak Ridge, TN 37831-1020.

The pendant drop method using the video imaging techniques and software described in this report has been demonstrated to be capable of measuring IFTs in the range of 0.02 to 72 mN/m with good precision and accuracy. There are no theoretical reasons why the range cannot be extended to lower values of IFT with additional refinements to the system.

IFTs were determined for microbial cell free supernatants and two crude oils. The cell free supernatants did lower the IFT between the crude oils and the water phase. The lowest IFT measured was 1.01 ± 0.04 mN/m for Lick Creek crude oil (17.5°API at 60°F) and the cell free supernatant from Bacillus licheniformis JF-2 grown aerobically in Trypticase Soy Broth (TSB). None of the cell free supernatants tested resulted in ultra-low IFTs ($< 10^{-3}$ mN/m) generally accepted as required for excellent tertiary oil recovery.

ACKNOWLEDGEMENTS

This research is performed for the U.S. Department of Energy, Assistant Secretary for Fossil Energy, under the Idaho Field Office (DOE-ID) contract number DE-AC07-76ID01570. The Idaho Field Office Project Manager is Dr. Leonard Keay. The Technical Program Manager is Ms. Rhonda L. Patterson of the DOE Bartlesville Office. The authors thank the Bartlesville Project Office and Idaho Field Office for their support of this research.

CONTENTS

ABSTRACT	iii
SUMMARY	iv
ACKNOWLEDGEMENTS	v
 1. INTRODUCTION	 1-1
2. THEORY OF IFT DETERMINATION BY PENDANT DROPS	2-1
3. MEASUREMENT OF INTERFACIAL TENSION	3-1
3.1 Hardware	3-1
3.1.1 Optical Cell	3-1
3.1.2 Illumination	3-1
3.1.3 Video Camera and Lens	3-1
3.1.4 Computer, Video Frame Digitizer Board, and Monitor	3-2
3.2 Software	3-2
3.2.1 Using the Program	3-3
3.2.2 Real-Time versus Captured-Image Modes	3-3
3.2.3 System Calibration	3-4
3.2.4 Calculation of Interfacial Tension	3-4
3.2.5 Drop Stability Monitoring	3-5
3.2.6 Camera Focus	3-5
3.2.7 Saving and Restoring Images	3-5
3.2.8 Edge Detection	3-5
3.2.9 Noise Filter	3-5
3.2.10 Diagnostics	3-6
3.3 Operation and Verification of IFT Apparatus	3-7
3.3.1 Calibration	3-7
3.3.2 Camera Settings	3-7
3.3.3 Calibration of Interfacial Tensions	3-7
3.3.4 Reproduction of Literature Values of IFT	3-8
3.4 Results and Discussion	3-14
3.4.1 Reproduction of Literature IFT Values for Ternary Liquid/Liquid Systems	3-14
3.4.2 IFT Measurements of Cell Free Bacterial Supernatants	3-14
3.4.3 IFT Measurements of <u>B. licheniformis</u> Acid Precipitate	3-15
3.5 Conclusions	3-17
 4. REFERENCES	 3-18
 APPENDIX A	 A-1
APPENDIX B	B-1

FIGURES

2-1. Critical dimensions for calculating interfacial tensions	2-3
3-1. Plot of water surface tension vs. temperature	3-9
3-2. IFT of air and aqueous solution of crude Biosurfactant from <u>B. licheniformis</u>	3-16
3-3. IFT of aqueous crude Biosurfactant as a function of Salinity and crude oil	3-16

TABLES

3-1. Program Function Keys	3-3
3-2. Surface tension of air and water	3-9
3-3. Surface tension of air and hexane	3-10
3-4. Interfacial tension of 1-butanol and water	3-11
3-5. Interfacial tension between 2-butanol and water	3-12
3-6. Interfacial tension of sodium dodecyl sulfate (SDS)/1-butanol/brine/toluene	3-13
3-7. IFTs for Medium E cell free supernatants measured with Schuricht and Lick Creek crude oils	3-14
3-8. IFTs for TSB cell free supernatants measured with Schuricht and Lick Creek crude oils	3-15

1. INTRODUCTION

Nearly two-thirds of the known U.S. oil resource remains in the ground after conventional recovery methods have been applied. The remaining oil can be classified as mobile or immobile oil. Mobile oil is remaining oil that is considered mobile to waterflooding or steam injection but is bypassed because of heterogeneities in reservoir rock formations and immobile oil is oil not mobile to such processes.¹ Immobile oil remaining after conventional recovery is trapped in pore structures by viscous and capillary forces and cannot be removed by the pressures generated during waterflooding or steam injection. Chemical and miscible processes are the principal EOR methods for producing immobile light oil.¹

The immobile oil can be recovered if the ratio of viscous to capillary forces (expressed as the capillary number) is sufficiently high.² The capillary number is given by

$$N_c = \frac{V\mu}{\gamma} \quad (1)$$

where

V = *darcy velocity*

μ = *displacing phase viscosity*

γ = *interfacial tension.*

N_c can be increased by either increasing the viscosity of the displacing fluid or by decreasing the interfacial tension (IFT). The interfacial tension is defined as the minimum amount of work required to create the boundary between two immiscible phases. In order to increase the oil recovered by waterflooding, either the viscosity or the IFT of the displacing fluid must be modified to increase N_c . The interfacial tension can be lowered by adding a surfactant to the waterflood operation, while the viscosity may be increased by adding a water soluble polymer to the waterflood operation.

Oil can be displaced from porous media if the IFT between the oil and water (typically 40 to 50 mN/m) is lowered sufficiently. Calculations using models of trapped oil have estimated that interfacial tensions lower than 0.1 mN/m (usually $< 10^{-3}$ mN/m) are needed for oil recovery.^{3,4,5,6} Ultra-low IFTs (about 10^{-3} mN/m) mobilize the oil trapped by capillary forces. Lowering the IFT by use of surfactants (which align at the phase boundary) decreases the pressure required to force a non-wetting phase (oil) through a small capillary. Immobile oil trapped in pores can be released by flooding the reservoir with a surfactant solution capable of reducing the IFT to sufficiently low values.

Measurement of IFTs has been accomplished by a variety of methods, including pendant drop,^{7,8,9} sessile drop,¹⁰ and spinning drop¹¹ methods. The pendant or sessile drop methods of determining IFTs was first suggested by Worthington^{12,13} and Ferguson.¹⁴ All of these methods require measurements to be performed on the drop shape. In the past a photographic record was made of the drop, and measurements of drop shape and size were determined from this record.

Current developments in personal computers coupled with the ability to digitize video images¹⁵ enables the researcher to automate the IFT measurement process. Digital video methods provide a fast, reliable process to determine the IFT.^{16,17,18} Digital video methods have been applied to measuring the IFT of bacterial biosurfactants.¹⁹

In the video method of determining the IFT, the size and shape of a transparent or opaque drop of one fluid (crude oil) immersed in a second transparent fluid (surfactant solution) is displayed on a monitor with a CCD video camera. The video image on the monitor is digitized and stored by a computer-controlled system. Custom software (developed at the INEL) determines various drop shape measurements from this drop image, and then computes the IFT.

2. THEORY OF IFT DETERMINATION BY PENDANT DROPS

Micellar (surfactant) flooding works by lowering the interfacial tension between crude oil and water. Capillary forces between crude oil and formation water determine the pressures needed to displace the oil from formation pores. The pressure required to overcome capillary forces is given by the Young-Laplace equation.^{20,21,22,23}

$$P_c = \gamma J \quad (2)$$

where

P_c = capillary pressure

γ = interfacial tension

J = interface curvature.

The interface curvature, J , is the curvature of the drop surface. The curvature of the drop surface can be described by two radii, one in the X plane and one in the Y plane. If a set of Cartesian coordinates is chosen such that the XY plane contains the normal to the surface at some point P, then an equation may be derived to express the radius of curvature as follows:^{24,25}

$$\frac{1}{R_1} = \frac{\left(\frac{d^2y}{dx^2}\right)}{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}}} \quad (3)$$

A second radius of curvature is obtained by selecting a second plane that is perpendicular to the first plane. The radius of curvature for the second plane is given by

$$\frac{1}{R_2} = \frac{\left(\frac{dy}{dx}\right)}{x \left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{1}{2}}} \quad (4)$$

The sum of $1/R_1$ and $1/R_2$ is a constant and is equal to the interface curvature J . Equation (2) can be combined with equations (3) and (4) to determine the interfacial tension from a pendant or sessile drop as follows:²⁶

$$P_c = \gamma \left[\frac{\left(\frac{d^2y}{dx^2}\right)}{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}}} + \frac{\left(\frac{dy}{dx}\right)}{x \left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{1}{2}}} \right] \quad (5)$$

If the origin is set at the apex of the drop, the two radii are equal at this point and can be designated by b . The pressure, P_c , at the apex is

$$\frac{2\gamma}{b} \quad (6)$$

Inside the drop, the initial pressure, P_c , is increased by the amount

$$\Delta \rho g z \quad (7)$$

where

$\Delta \rho$ = density difference between drop material and external phase

g = acceleration due to gravity

z = distance from apex to a selected horizontal level z .

Equations (5), (6), and (7) can be combined to give

$$\frac{2\gamma}{b} + \Delta \rho g z = \gamma \left[\frac{\left(\frac{d^2 y}{dx^2} \right)}{\left[1 + \left(\frac{dy}{dx} \right)^2 \right]^{\frac{3}{2}}} + \frac{\left(\frac{dy}{dx} \right)}{\left[1 + \left(\frac{dy}{dx} \right)^2 \right]^{\frac{1}{2}}} \right] \quad (8)$$

To determine the interfacial tension, the radii of curvature at the apex b and at a particular value of z must be determined. When the radii of curvature at a particular value of z are both finite but unequal, equation (8) must be solved by numerical integration.

Theoretical evaluations of equation (8) to determine the interfacial tension are given in several literature reports.^{27,28,29} The analysis used in this work uses an empirical procedure³⁰ that defines a function, S (drop shape factor), to determine the drop shape. The maximum diameter, d_e , of the drop is determined and then used in the determination of the horizontal dimension, d_s , of the drop as shown in Figure 2-1. The ratio of these two measurements, d_s/d_e , is equal to the drop shape factor S .

Algorithms have been developed using S such that the IFT between the drop and the external solution can be calculated quickly.³¹ Equations (9) through (12) demonstrate how S is used to calculate the interfacial tension.³²

$$\gamma = \frac{\Delta \rho g b^2}{\beta} \quad (9)$$

where

$$S = \frac{d_s}{d_e} \quad (10)$$

$$\beta = 0.12836 - 0.7577S + 1.7713S^2 - 0.5426S^3 \quad (11)$$

$$b = \frac{d_e}{2(0.9987 + 0.1971\beta - 0.0734\beta^2 + 0.34708\beta^3)} \quad (12)$$

With the increased usage of high speed computers, new methods have been developed that measure the drop profile at numerous points.^{33,34,35} Increased accuracy and precision results by analyzing multiple points on the drop profile instead of two points (d_e and d_s). These methods require more complicated computer programs and greater computing power than the current method. The advantage of the present method is the speed in which the interfacial tension is determined and the fact that a low cost personal computer can be used for the calculations. As will be shown in the following section, the accuracy and precision is very good for measurement of IFTs in the range from 72 to 10⁻² mN/m, which is adequate for many EOR applications. Ultra-low IFT measurements (10⁻³ mN/m or less) may require additional equipment development and use of more sophisticated numerical techniques.

Figure 2-1. Critical dimensions for calculating interfacial tensions.

3. MEASUREMENT OF INTERFACIAL TENSION

The complete IFT system consists of the following major components: optical cell, illumination, video camera and lens, computer, video frame digitizer board, monitor, and software. The hardware system provides a video image of a pendant drop suspended in another liquid, while the software analyzes the drop shape to calculate the IFT between the two liquids. This software is intended to run on an AT-class computer with a video frame digitizer board that interfaces to a video camera and video display.

3.1 Hardware

3.1.1 Optical Cell

The optical cell is comprised of microscope slides mounted into a machined aluminum body fitted with chromatography fittings for fluid introduction by a tube. For simplicity and cost, the cell windows are standard microscope slides that can be easily removed for cleaning or replaced when contaminated. A series of different internal diameter tubing are used to measure different ranges of IFTs. The lower the IFT to be measured, the smaller the diameter tubing needed. Standard 1/16 in. stainless steel tubing with an internal diameter (ID) of 1.58 mm and quartz tubing with an ID of 0.22 mm were used for the measurements reported in this work.

The main chamber of the cell holds the surfactant solution to be tested. A second phase, lighter in density (liquid or gas), such as crude oil, is introduced to the surfactant solution by syringe injection through the tubing, forming a drop. The drop floats above and is attached to the tubing. The crude oil drop is enlarged to the maximum size possible (near the buoyant release point) and the drop shape recorded. The optical cell is illuminated from behind so that the drop image formed on the monitor appears as a black drop shape on a white background. Transparent drops appear as dark outlines surrounding a lighter interior due to the drop acting as a lens.

The latest version of the cell permits temperature control of the cell using temperature controlled water flowing through holes drilled in the metal frame of the cell and also was designed with a smaller path length. The smaller path length was necessary to accommodate bacterial cultures containing biosurfactants in the droplet cell. The bacterial cultures are translucent instead of transparent.

3.1.2 Illumination

A fiber optics illumination system is used to provide the background light, reduce heating effects, and simplify mounting design. A ground glass diffusion lens between the light source and cell is used to ensure uniform illumination of the drop. Alternate methods of lighting that have been used in other devices include internal illumination of the pendant drop.³⁶

3.1.3 Video Camera and Lens

A Cohu model 6515 CCD camera is used for acquiring the video image. The sensor of the CCD camera is a 1/2 inch format (actual usable area is 6.4 by 4.8 mm). Sensor pixel count and magnification are chosen so that one obtains at least 200 pixels across the minimum dimension to be measured. The goal is to have measurements of at least 1% precision.

A Nikon AF Micro-Nikkor lens is used with the camera. The focal length of the lens is 60mm with a maximum aperture of f/2.8, a minimum aperture of f/32, and a minimum focusing distance of 219 mm. An adaptor is necessary to attach the lens to the camera (the lens has a standard Nikon mount and

the camera a C-mount adaptor). A series of telexenders are used to increase the magnification of the lens, as required by the different drop diameters. The telexenders may be used separately or in combination. Sizes of telexenders are 0.5, 5, 10, 20, and 40 mm.

3.1.4 Computer, Video Frame Digitizer Board, and Monitor

A PC based system was selected to control the video system and to perform the calculations. The personal computer being used is an IBM-AT with a Data Translation DT2853 video frame digitizer board installed in the computer. The computer controls data acquisition, computation, and display functions. A video monitor displays the video image output by the DT2853 board. The monitor is a Burle 13-in. black and white monitor.

3.2 Software

The software program MEOR4D analyzes the drop video image, obtains values for the drop shape factor, and calculates the IFT between two immiscible phases using equations (9) to (12). The software is written in the MicroSoft C language. The program is written to run on an AT-class personal computer with a Data Translation DT2853 video frame digitizer board. The software source code is listed at the end of this report.

The software controls the video frame digitizer and acquires, reduces, and displays the data. It uses commercially available Data Translation DT-IRIS device drivers intermixed with a combination of specifically developed subroutines. Subroutines from Data Translation's library and from Microsoft's C libraries are also used in the software. The software is menu driven for increased ease of use. The user interacts with the program by pressing special function keys indicated by the displayed menus. Among the functions available are calibration, video adjustments, and plots of video intensity. Other options include saving and restoration of video images to computer disks, data reduction, and printing of IFT data.

This program was written to run on either of two different hardware systems, called B7 and GDL. The B7 system includes a DT-2853 frame grabber and a video camera but no printer. The GDL system includes a DT-2861 frame grabber and a printer but no camera. Each system includes a file called SYSTEM.DEF which contains one line of ASCII characters, either "#define SYSTEM GDL" or "#define SYSTEM B7". This file is read during compilation of the program, and the compilation is adapted to the hardware system. The program can be tailored to different systems, primarily by editing those parts of the program that include conditional compilation directives that refer to the parameter SYSTEM.

Two files, MEOR4D.C and MEORS1.C, comprise the program MEOR4D. These files must be compiled separately. During the compilation, there must be a file SYSTEM.DEF (described previously), unless the programs have been changed to eliminate references to this file. There are no other special requirements for compilation.

The two compiled files must be linked with the Data Translation library, ISCLLIB, and the Microsoft QuickC library, GRAPHICS. (Other versions of Microsoft C libraries may also work.) The stack size option to be specified is /st:0x2000. The link command should be:

`LINK /st:0x2000 MEOR4D+MEORS1,.,ISCLLIB+GRAPHICS;`

assuming that the library files are in an accessible directory.

3.2.1 Using the Program

The program can perform several functions, including calculating the IFT, obtaining screen measurements, and diagnosing problems. The functions are initiated by pressing either the function keys or shift-function key as indicated in Table 3-1.

Table 3-1. Program function keys.

Key	Function
F1	"freeze" an image (captured-image mode)
F2	real-time images direct from camera
F3	drop stability; monitor drop edge positions
F4	focus; real-time intensity versus position
F5	calibrate routine
F6	calculate interfacial tension
F7	get saved image from a disk file
F8	save an image to a disk file
F9	execute an MS-DOS command
F10	stop execution of this program
shift-F1	graph image intensity along cursor row
shift-F2	graph image intensity along cursor column
shift-F3	monitor drop edge positions and write to a file
shift-F7	graph intensity along cursor row and column
shift-F8	toggle cursor on and off
shift-F9	display certain program variables
shift-F10	(when in main menu) display second menu
ENTER	(when in second menu) display main menu
CURSOR keys	move cursor

3.2.2. Real-Time versus Captured-Image Modes

The video system can operate in two modes, real time or captured-image. Key F2 initiates the real time mode in which the video monitor displays the real-time image detected by the camera. This mode is useful while creating a suspended liquid drop to be measured. When the F1 key is pressed the system is switched from the real-time mode to the captured-image mode. In the capture-image mode a complete image is stored in memory and is available for manipulation by the program. The captured-image mode is used for analyzing images and for saving images to disk. The system will automatically change from one mode to another mode if a program function is requested that requires a mode different from the current mode. For example, if the system is in the real-time mode and key F6 is pressed (calculation of IFT), the system automatically switches to the captured-image mode; the user does not have to switch to the captured-image mode before pressing key F6. It has been found useful for the operator to switch from the real-time mode to the captured-image mode before initiating the calculation of the IFT. This allows the operator to inspect the drop image before initiating the IFT calculation.

3.2.3 System Calibration

For IFT calculation, the software must convert screen image measurements (in pixels) to units of length (mm or inches). Prior to IFT measurements, the system must be calibrated. This is done by a calibration process that is initiated by key F5. For calibration, a standard (consisting of a spherical steel ball of known diameter mounted on a supporting rod) is installed in the test cell in the position normally occupied by the liquid drop. The program measures the calibration ball image diameter in pixels and then queries for the ball diameter in millimeters or inches. The calibration procedure then computes the number of pixels per unit length in both the X and Y axis.

A set of calibration standards were prepared from precision stainless steel balls to cover a range of droplet sizes. The calibration standards range as follows: 0.3124, 0.2811, 0.2499, 0.2186, 0.1564, 0.125, 0.0937, and 0.0625 inches. The calibration standard selected should be as close to the drop size as possible.

The calibration information is written to the disk file (MEOR.CAL) each time a calibration is done. Each time the program is executed, it reads the calibration information from this file. The latest calibration information is always available even if the program (or even the entire computer) has been stopped since the last calibration.

It is the operator's responsibility to be sure that the camera focus has not changed since the last calibration. A calibration should be done if the optical geometry is changed (as by changing the camera focus).

3.2.4 Calculation of Interfacial Tension

Key F6 initiates the calculation of the IFT from the displayed drop image. The calculation requires measuring d_e and d_s on the drop image (Figure 1). This requires that the drop's height be greater than its width. The operator must be sure that this condition is met. A horizontal line will be displayed on the screen before calculation of the IFT. This line can be moved vertically with the cursor keys, and must be set even to the end of the tube forming the drop. The system will ignore the image below this line. The purpose of rejecting part of the image is to prevent the system from taking measurements on the drop forming tube (which may be larger than the drop diameter). If the line is already in an acceptable position, the operator needs only press the ENTER key. The program then measures the d_e and d_s of the drop image. To complete the calculation, the density difference between the drop phase and the external phase must be entered. If the program has already been used for calculating IFTs, the density difference used in the prior calculation has become the default density difference. If the prior value is acceptable, pressing the ENTER key causes the program to accept the default density difference. If the default density difference is unacceptable, the new density difference is typed in and the ENTER key is pressed.

After the density difference has been entered, the interfacial tension will be calculated and displayed on the screen. The program then queries whether the results of the current IFT calculation are to be stored in a disk file. The file name must be supplied by the operator, and may include drive and path information, but the total characters entered should not exceed 49 characters. MEOR4D limits all file names to 49 characters, including drive and path information. If an already existing file is specified as the destination for the new results, the new data is appended to the data in that file.

3.2.5 Drop Stability Monitoring

In the stability monitoring mode (shift-F3), the program captures a drop image. The purpose of this function is to allow the operator to decide whether the suspended drop is stable enough to allow a reasonable measurement of IFT. The program locates the left edge, drop apex and right edge of the drop diameter as determined from the row and column of the cursor position. The left and right edge positions are measured in pixels from the left edge of the screen (512 pixels across the full screen width), and the apex is measured from the top edge of the screen (480 pixels from top to bottom of the screen). The program prints these three positions on the computer screen (or by using shift-F3, writes the three positions and the time to a disk file). This cycle is repeated every 2 seconds until the ENTER key is pressed. If the data are written to a disk file, MEOR4d uses a file name comprising eight numerals plus the extension "mon". The eight numerals represent the month, day, hour, and minute of the current time from the system clock.

3.2.6 Camera Focus

In the focus mode (key F4), the system repeatedly grabs an image and displays two graphs of the intensity versus position along the current cursor rows. One graph displays intensity versus the x row of the cursor while the second graph displays the intensity versus the y row of the cursor. The purpose of this is to allow the operator to see clearly whether the intensity changes abruptly (good focus) or gradually (poor focus) at the edge of the drop (or the calibration ball or other object in the image). Note that the system should be calibrated after any adjustment in the camera focus.

3.2.7 Saving and Restoring Images

Images can be stored in disk files by using the F8 key, and then restored (brought back into active program memory) with the F7 key. This makes it possible to save sequences of images for later analysis, and, also, to transfer images to different computers running the program. There is a practical limit to this process: each image requires approximately 250 Kbyte of disk space. This limits the number of images that can be stored to available disk space.

3.2.8 Edge Detection

The most critical part of the program is locating the edges of a drop in the image. The program contains two global variables that are used in this edge location process: step-height and step-width. The exact values of these parameters are not critical, but they must be reasonable for the hardware system being used. The supplied value of step-height = 150 was selected for a system in which the image intensity values are scaled to range from 0 through 255. If the images are not scaled this way, so that the difference between "black" and "white" pixels is not somewhat larger than 150, the value of step-height should be changed accordingly. Note that a too-small value of step-height will cause random noise to be mistaken for edges. The value of step-width = 4 is compatible with edges that make the transition from bright to dark (an intensity change of step-height) in no more than four pixels. If the images are not sharp, so that edge transitions are broader than four pixels, the step-width value should be increased. However, if step-width is too large, smooth intensity variations may be mistaken for edges.

3.2.9 Noise Filter

The program uses a non-linear filtering process (median filter) in an effort to reduce the effects of noise in the images. Median filtering preserves edges (or steps) in the image, and removes positive and negative peaks that are less than half the width of the filter window (in pixels). Depending on the

structure of the noise in the images and on the sharpness of the images, it may be advantageous to change the size of this filter window. This median filtering is a rather time-consuming process, and the time required for image processing is greater if the filter window size is increased.

When analyzing a drop image or a calibration ball image, the program first tries to find a minimum of three points on the edge of the drop (or ball) image, such that the points (a) span the maxima of drop width or drop height and (b) can be fitted with a quadratic polynomial to give an accurate indication of the position of the maxima. Errors can occur in image processing, usually because of excessive noise in the images, poor focus, or poor contrast. The resulting error messages including "fit-width", "fit-height", or "fit-apex", depending on which aspect of the drop or calibration ball is being measured when the error occurs. The error message includes a pair of numbers (n and m) separated by a period. The m value gives detailed information about exactly where in the program the error occurs and the n values indicate what type of error occurred. The values n=1 and n=2 indicate that the expected edge was not detected along the row or column being searched. n=3 indicates that an attempt to fit a quadratic function to the detected edge points failed, and the results are approximately correct but inaccurate. n=4 indicates that the initial search for edge points did not locate the desired maximum height or width. n=5 and n=6 indicate that the search for edge points went outside of the reasonable range. n=7 indicates that fewer than the required three edge points were found.

The remedy for these errors is to improve the image quality, or to adjust the program parameters edge-height and edge-width. The m=2, n=5 error sometimes occurs because the position of the base of the drop is set too conservatively. A very small drop can also cause these errors.

3.2.10 Diagnostics

A cross-hair cursor may be displayed on the video monitor. The position of the cursor is controlled by the normal cursor positioning (arrow) keys, and key F8 turns the cursor display on and off. The cursor is not usually required for normal operation, but it is sometimes useful for diagnosing unusual situations or malfunctions. Keys shift-F1, shift-F2, and shift-F7 allow graphing the image intensity versus the position along the cursor row, column, or both. These keys may be used to monitor the contrast and focus of the image much as the F4 key.

Key shift-F9 gives a list of values of important variables in the program, which is also a diagnostic aid. The first five rows of values printed represent the middle, top, bottom, left, and right of the drop (or ball), respectively. The four values in each of these rows are a floating and an integer value for the vertical (row) position and a floating and an integer value for the horizontal (column) position of the drop's center or edge. After these first five rows of data, the diagnostic printout lists the drop (or ball) width and height; the calibration factors; the position of the base of the drop; and the cursor position. The positions and dimensions in this diagnostic printout are in pixels, measured from left (0) to right (511) and from top (0) to bottom (480) of the image.

During processing of a drop or calibration ball image, the program indicates which pixels in the image are being used for locating the edges of the drop or ball. The program draws lines to indicate where drop edges have been detected. The lines drawn are offset from the edges to inside the drop image for ease of operator viewing. These lines should be close to the edges being sought. If the lines indicated are not in reasonable locations, this is an indication of a problem (usually noise) in the image. Note that the image displayed with these pixels marked is just a copy of the image used for analysis; changing these pixels to "white" for display does not change the pixels in the images being analyzed or stored to disk files.

3.3 Operation and Verification of IFT Apparatus

3.3.1 Calibration

The system must be calibrated prior to IFT measurements (see SYSTEM CALIBRATION). An automated calibration procedure computes the number of pixels per unit length in both the X and Y axis. These calibration values are used to convert all pixel measurements into actual dimensions. Video camera and lens settings are adjusted to provide a high contrast image of the ball and the focus is adjusted to provide a sharp image. Once calibrated, the settings and focus must not be changed or the system will have to be recalibrated.

After the calibration is performed, the calibration standard is removed, and the test cell is placed in front of the lens. A drop is formed, and a x,y,z-translator is used to bring the drop into focus. The lens focus must not be changed, or the system will have to be recalibrated.

All lenses have a plane (known as the focal plane) that varies in width with the F/stop setting. Any object placed in this plane will be in focus to the lens and camera. The result of a wide focal plane is that the calibration standard and test cell droplet may vary in position in the focal plane, but still be in focus. In order to ensure that the calibration standard and test cell droplet reside in the same position, a narrow focal plane is used. Decreasing the F/stop narrows the focal plane. Final positioning within a focal plane is accomplished by using the x,y,z-translator to position the calibration standard, and later the test cell droplet, as close to the lens as possible and still have a sharply focused image.

A set of calibration balls were prepared from precision stainless steel balls to cover a range of droplet sizes. The calibrations range as follows: 0.3124, 0.2811, 0.2499, 0.2186, 0.1564, 0.125, 0.0937, and 0.0625 inches.

3.3.2 Camera Settings

Camera settings have been optimized for image contrast and sharpness. The following settings have been found to give the best IFT results:

- Offset = preset setting
- Gain = auto setting
- Gamma = 1.0
- Sharpness control = full counterclockwise
- Camera shutter = high
- Autoblock = off

A specific setting for the lens and light source was used to determine these camera settings. The lens F/stop was set at 5.6 with the fiber optics light source used at the 1/3 intensity (low) setting.

3.3.3 Calibration of Interfacial Tensions

After calibration, the operator changes the calibration standard for the test cell, and generates a drop of the test liquid. The automated data acquisition and data reduction process is then initiated. The system acquires a video image of the drop, then locates the maximum horizontal diameter, d_t , and the apex (or top) of the drop. These values are then used to locate and measure the drop diameter, d_d . The program also requests certain physical properties of the test fluids (such as the density difference between

the drop material and cell contents), then computes the IFT of the system. The density values used must be accurate for the fluids and the conditions of the measurement.

3.3.4 Reproduction of Literature Values of IFT

In order to ensure the proper operation of the hardware and software, IFTs were determined for a variety of gas/liquid and liquid/liquid systems. The systems were selected based on finding published values for IFTs for these systems. Both air/liquid³⁷ and binary and ternary liquid/liquid³⁸ systems were used to test the system.

3.3.4.1 Air and Water Surface Tension. An air and water system was used for initial experimentation. The density difference of air and water must be determined in order to calculate the IFT of air and water. The density of air varies as a function of temperature, barometric pressure, and moisture content, and can be calculated with the following formula:

$$D_{air} = \frac{0.0012929}{(1 + 0.0036613 T)} \frac{(B - 0.3783 E)}{760 \text{ mm}} \quad (13)$$

where

T = temperature in degrees C

B = barometric pressure in mm

E = vapor pressure of water in air in mm.

If the moisture content of air is neglected, equation (13) reduces to

$$D_{air} = \frac{0.0012929}{(1 + 0.0036613 T)} \frac{B}{760 \text{ mm}} \quad (14)$$

Density and surface tension values for water as a function of temperature be can found in many handbooks.^{37,39} At 20 and 25°C, the density of water is 0.9982058 and 0.9970474 g/cm³, respectively. The surface tension of water (IFT between air and water) varies linearly with temperature as shown in Figure 3-1.

Water for surface tension determinations was obtained from a Sybron/Barnstead Nanopure II water purification system and had a conductivity reading of 0.06 μmhos/cm. Air used in the measurements was room air. The results of surface tension measurements of air and water using the INEL IFT apparatus are listed in Table 3-2. The surface tension varies from 71.34 to 72.03 mN/m while the fractional standard deviation (FSD, also known as the per-cent error) ranges from 0.1 to 0.36. These values compare well with literature values, both for the surface tension and the FSD. It is obvious from the averages listed in Table 3-2 that a systematic error must be present (the average changes significantly from day to day). The error may be due to the focal plane problem described in the calibration procedure; it may also be due to vibrations transmitted to the apparatus.

Figure 3-1. Plot of water surface tension vs. temperature.

Table 3-2. Surface tension of air and water.^a

Test	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
1	71.49	71.23	71.73	71.67	71.16	71.95
2	71.43	71.30	71.56	71.43	71.30	71.90
3	71.60	71.17	71.50	71.50	71.22	71.80
4	71.60	71.54	71.61	71.45	71.37	72.07
5	71.55	71.38	71.97	71.48	71.29	72.16
6	71.68	71.34	71.54	71.47	71.50	72.17
7	71.39	71.31	71.68	71.48	71.60	71.92
8	71.61	71.17	71.67	71.57	71.78	71.94
9	71.50	71.41	71.88	71.47	71.71	72.07
10	71.70	71.54	71.88	71.47	71.93	72.28
Avg =	71.55	71.34	71.70	71.50	71.48	72.03
SD ^b =	0.10	0.13	0.16	0.07	0.26	0.15
CI ^c =	0.16	0.20	0.24	0.11	0.39	0.23
FSD ^d =	0.14	0.18	0.22	0.10	0.36	0.21

a. All units in mN/m unless otherwise stated.

b. SD = Standard deviation, method of least squares.

c. CD = Confidence interval for small samples^{40,41} is calculated by $4.781 \times \text{standard deviation} / (\text{number of samples})^{1/2}$.

d. FSD = Fractional standard deviation;⁴² also known as the % error, or experimental precision and may be calculated by $(\text{standard deviation} / \text{average}) \times 100$.

3.3.4.2 Surface Tension of Air and Hexane. The surface tension of air and hexane was determined in the same manner as air and water. The data are given in Table 3-3. The hexane was Fisher HPLC grade and was used without purification. The density of hexane is 0.65937 and 0.65481 g/cm³ at 20 and 25 °C, respectively. The surface tension of hexane at 25°C is 17.91.³⁹ The surface tension may alternatively be calculated in the temperature range of 0 to 30°C by the following formula:⁴³

$$\gamma = 20.44 - 0.1022T \quad (15)$$

For the temperature range of 0 to 100°C, the following equation may be used⁴⁴:

$$\gamma = 20.44 - 0.1022T \quad (16)$$

The surface tension values measured for air and hexane agree well with literature values.

Table 3-3. Surface tension of air and hexane.^a

<u>Test</u>	<u>Set 1</u>
1	17.936
2	17.905
3	17.886
4	17.882
5	17.892
6	17.818
7	17.890
8	17.864
9	17.826
10	17.827
Avg	= 17.873
SD ^b	= 0.039
CI ^c	= 0.058
FSD ^d	= 0.216

a. All units in mN/m unless otherwise stated.

b. SD = Standard deviation, method of least squares.

c. CD = Confidence interval for small samples^{40,41} is calculated by $4.781 \times \text{standard deviation} / (\text{number of samples})^{1/2}$.

d. FSD = Fractional standard deviation;⁴² also known as the % error, or experimental precision and may be calculated by $(\text{standard deviation} / \text{average}) \times 100$.

3.3.4.3 Interfacial Tension of 1-Butanol and Water. The IFTs for an equilibrated system of 1-butanol and water were measured. The data are given in Table 3-4. Water was the same as discussed above. The 1-butanol used was Fisher reagent grade, and was used without further purification. Because 1-butanol is slightly soluble in water, a preequilibrated system was used. The system was preequilibrated by mixing 10 mL of water and 10 mL of 1-butanol in a flask. The mixture was stirred for 20 minutes, and let stand overnight. The aqueous (lower) phase and the organic (upper) phase were separated and used for the IFT measurements. The densities³⁸ used were 0.98586 g/cm³ for the aqueous phase and 0.84304 g/cm³ for the organic phase. The literature^{38,45} reports an IFT of 1.8 mN/m for this system. The values determined for the IFT agree well with the literature values, however the FSD is significantly higher for the 1-butanol and water system.

Interesting variations were noted when the calibration standard was changed. Using a 0.0937 in. calibration ball and 20 samples an average IFT of 1.701804 mN/m (standard deviation = 0.016) was obtained. When the calibration ball was changed to the lower value of 0.0625 in. the average IFT of another 20 samples increased to 1.938365 mN/m with a standard deviation of 0.034.

Table 3-4. Interfacial tension of 1-butanol and water.^a

Test	Set 1	Set 2	Set 3
1	1.721	1.888	1.715
2	1.719	1.904	1.686
3	1.717	1.925	1.668
4	1.723	1.962	1.670
5	1.717	1.936	1.670
6	1.711	1.897	1.710
7	1.714	1.885	1.679
8	1.716	1.904	1.696
9	1.713	1.923	1.708
10	1.713	1.896	1.668
Avg =	1.717	1.912	1.687
SD ^b =	0.004	0.024	0.019
CI ^c =	0.006	0.037	0.028
FSD ^d =	0.217	1.274	1.106

a. All units in mN/m unless otherwise stated.

b. SD = Standard deviation, method of least squares.

c. CD = Confidence interval for small samples^{40,41} is calculated by $4.781 \times \text{standard deviation} / (\text{number of samples})^{1/2}$.

d. FSD = Fractional standard deviation;⁴² also known as the % error, or experimental precision and may be calculated by $(\text{standard deviation} / \text{average}) \times 100$.

3.3.4.4 Interfacial Tension of 2-Butanol and Water. 2-Butanol, like 1-butanol, is slightly soluble in water. As with 1-butanol a preequilibrated system was used. The system was preequilibrated the same way as the 1-butanol system. The water used was the same as described above. The 2-butanol used was Fisher reagent grade, and was used without further purification. The aqueous (lower) phase and the organic (upper) phase were separated and used for the IFT measurements. The densities³⁸ used were 0.98487 g/cm³ for the aqueous phase and 0.83145 g/cm³ for the organic phase. The literature³⁸ reports an IFT of 2.1 mN/m for this system. The values determined for the IFT, given in Table 3-5, agree well with the literature values, however as in the 1-butanol system, the FSD is significantly higher for the 2-butanol and water system than for the surface tension of air and water.

Table 3-5. Interfacial tension between 2-butanol and water.

Test	Set 1	Set 2	Set 3
1	2.199	2.331	2.127
2	2.230	2.297	2.181
3	2.193	2.343	2.163
4	2.225	2.314	2.189
5	2.239	2.305	2.158
6	2.310	2.405	2.231
7	2.185	2.241	2.215
8	2.165	2.209	2.173
9	2.167	2.225	2.228
10	2.264	2.186	2.229
Avg =	2.218	2.286	2.189
SD ^b =	0.046	0.069	0.035
CI ^c =	0.069	0.104	0.054
FSD ^d =	2.062	3.001	1.620

a. All units in mN/m unless otherwise stated.

b. SD = Standard deviation, method of least squares.

c. CD = Confidence interval for small samples^{40,41} is calculated by $4.781 \times \text{standard deviation} / (\text{number of samples})^{1/2}$.

d. FSD = Fractional standard deviation;⁴² also known as the % error, or experimental precision and may be calculated by $(\text{standard deviation} / \text{average}) \times 100$.

3.3.4.5 Interfacial Tension of Methyl Acetate and Water. The apparatus could not find the top of a drop produced in this system. Virtually all adjustments were tried and compared. The software could not detect the drop edge because the drop image appeared as a thin dark outline. This change in the drop image from a black drop to the thin outline of a drop may be due to the similarity in the refractive index of water and methyl acetate. The refractive index value for methyl acetate is 1.3593, and for water is 1.33219. The refractive index for 1-butanol is 1.39931, and for 2-butanol it is 1.3954.

3.4 Results and Discussion

3.4.1 Reproduction of Literature IFT Values for Ternary Liquid/ Liquid Systems

IFTs were determined for a ternary liquid/liquid system. The system selected was sodium dodecyl sulfate (SDS)/1-butanol/brine/toluene mixture. After pre-equilibrating the system, the lower aqueous phase was placed in the cell, and the upper organic phase was injected through the tubing. The density differences, experimentally determined IFTs, and literature IFTs are given in Table 3-6.

These systems show decreased precision and accuracy as the IFT decreases. Two reasons cause this loss in precision and accuracy at low IFTs. First, the high magnifications required at low IFTs result in increased drop measurement errors. The second reason is the difficulty in precise alignment of the calibration standard and the drop formed, again resulting in drop measurement errors. Placement of the optical cell on a vibration table decreases the errors associated with IFT measurements at high magnifications. The results given in Table 3-6 demonstrate that IFTs as low as 10^{-2} mN/m can be measured with the INEL system. There are no theoretical reasons why even lower IFTs cannot be measured. Additional development of the apparatus involving smaller calibration standards, smaller injection tubes, larger magnification, different light source, vibration isolation, and possibly a different numerical approach may be required to accurately measure ultralow IFTs.

Table 3-6. Interfacial tension of sodium dodecyl sulfate (SDS)/1-butanol/brine/toluene.

System ^b	Density Difference (kg/m ³)	Interfacial Tensions		Standard Deviation ^f (mN/m)	Confidence Interval ^c (mN/m)	Per Cent Error ^d (%)
		Literature (mN/m)	This Work (mN/m)			
SDS/1-butanol/2.33% brine	153	0.24 ^a	0.2272	0.0166	0.0251	7.31
SDS/1-butanol/4.76% brine	147	0.0246 ^a	0.0391	0.0052	0.0079	13.32

a. M.J. Rosen, *Structure/Performance Relationships in Surfactants*, Washington, D.C.: American Chemical Society, 1984, p 329.

b. All measurements at 25°C; conductivity of water used was 0.06 μ mhos; all other chemicals are reagent grade.

c. 99.95% confidence interval for small samples.

d. % error = (standard deviation/interfacial tension)*100.

f. Number of samples = 10.

3.4.2 IFT Measurements of Cell Free Bacterial Supernatants

IFTs were measured between crude oils and cell free supernatants from cultures of isolated bacteria. Schuricht (25.6°API at 60°F) and Lick Creek (17.5°API at 60°F) crude oils were used for the measurements to represent a range of crude oil quality. The IFTs measured are given in Table 3-7. These naturally occurring bacterial isolates were obtained from samples taken from the Idaho Falls, Idaho sewage treatment plant (SEWBL) and from a Powder River Basin oilfield (COWBL). The cells were grown both aerobically and anaerobically in Medium E [American Type Culture Collection (ATCC)

1502] for 16 hr at 30° C, then centrifuged to remove the bacterial cells. Supernatant IFTs were compared to supernatant IFTs determined for *Bacillus licheniformis* JF-2 (ATCC 39307), a known producer of biosurfactant, grown under the same conditions. None of the IFTs were lowered to levels generally accepted as needed for effective EOR. As shown in the section 3.4.3, these young (16 hr) cultures appear to be producing concentrations of biosurfactants below the level of the critical micelle concentration (CMC). For both the aerobic and anaerobic cultures, the IFT decrease was greater for the heavier Lick Creek crude oil than for the Schuricht oil.

Table 3-7. IFTs for Medium E cell free supernatants measured with Schuricht and Lick Creek crude oils

<u>Isolate</u>	<u>Culture Method</u>	<u>Crude Oil</u>	<u>Interfacial Tension^a</u> <u>(mN/m)</u>
Blank	-----	Schuricht	34.26 ± 0.54
Blank	-----	Lick Creek	23.28 ± 1.20
<u>B. licheniformis</u>	Aerobic	Schuricht	27.54 ± 1.06
COWBL	Aerobic	Schuricht	32.25 ± 1.27
SEWBL	Aerobic	Schuricht	30.31 ± 1.60
<u>B. licheniformis</u>	Aerobic	Lick Creek	14.13 ± 0.64
COWBL	Aerobic	Lick Creek	17.10 ± 0.49
SEWBL	Aerobic	Lick Creek	10.08 ± 1.81
<u>B. licheniformis</u>	Anaerobic	Schuricht	34.52 ± 0.46
COWBL	Anaerobic	Schuricht	33.39 ± 1.65
SEWBL	Anaerobic	Schuricht	32.66 ± 1.49
<u>B. licheniformis</u>	Anaerobic	Lick Creek	19.27 ± 0.31
COWBL	Anaerobic	Lick Creek	16.26 ± 1.08
SEWBL	Anaerobic	Lick Creek	20.49 ± 1.27

a. IFTs taken for incubation time of 16 hr; ± standard deviation by method of least squares; n=10.

IFT measurements for the same three isolates cultured in Trypticase Soy Broth (TSB) media under aerobic conditions are given in Table 3-8. These results indicate that with a rich growth medium supernatants from these isolates are capable of lowering the IFT to about 1 to 3 mN/m. As shown by the control data, TSB alone has an effect on the IFT over Medium E.

Table 3-8. IFTs for TSB cell free supernatants measured with Schuricht and Lick Creek crude oils.

<u>Isolate</u>	<u>Culture Condition</u>	<u>Crude Oil</u>	<u>IFT^a</u> <u>(mN/m)</u>
Blank	-----	Schuricht	22.57 $\pm$ 0.52
Blank	-----	Lick Creek	18.83 $\pm$ 0.77
<u>B. licheniformis</u>	Aerobic	Schuricht	2.20 $\pm$ 0.02
COWBL	Aerobic	Schuricht	14.52 $\pm$ 1.80
SEWBL	Aerobic	Schuricht	
<u>B. licheniformis</u>	Aerobic	Schuricht	1.01 $\pm$ 0.04

a. IFT $\pm$ standard deviation (method of least squares), n=7 except for SEWBL and COWBL measured against Schuricht.

3.4.3 IFT Measurements of B. licheniformis Acid Precipitate

B. licheniformis JF-2 was grown in aerobic 5 L cultures to isolate the produced biosurfactant. Cells were removed from the cultures by centrifugation. The supernatant was acidified to pH 2 with HCl and allowed to stand at 4°C until a precipitate formed. The precipitate was collected from the supernatant by centrifugation of the cold supernatant at 11,000 rpm for 10 minutes. The resulting pellet was dissolved in water, and then pH adjusted to 7.0. This solution was then lyophilized to a solid residue and designated as crude biosurfactant. This crude biosurfactant was dissolved in distilled water and the IFT (surface tension) between air and the solution measured. Figure 3-2 is a plot of the interfacial tension of air as a function of the crude biosurfactant concentration. The lowest IFT obtained was 46 mN/m at a crude biosurfactant concentration of 150 mg/mL. No change in the slope was detected, indicating that the highest concentration of crude biosurfactant used was below the CMC.

A plot of the IFTs of crude biosurfactant solutions against two crude oils is shown in Figure 3-3. Crude biosurfactant solutions were varied in salt concentration to study the effect of salinity on the IFT. The crude biosurfactant concentration was 150 mg/ml. Schuricht and Lick Creek oils were selected as the second phase for the IFT measurements. Figure 3-3 shows that increasing the salinity of the precipitate solution resulted in increasing IFTs with both oils. The IFTs measured for Lick Creek oil were consistently lower than for Schuricht oil.

Figure 3-2. IFT of Air and Aqueous Solution of crude Biosurfactant from B. licheniformis

Figure 3-3. IFT of Aqueous Crude Biosurfactant as a Function of Salinity and Crude Oil

3.5 Conclusions

- 1) Interfacial tensions can be rapidly and accurately measured using real time video data. The described instrument uses a low cost AT-class personal computer with an optical cell, video camera and lens, and frame digitizer board to obtain the interfacial tensions.
- 2) The pendant drop method of determining interfacial tensions can be used to measure interfacial tensions in the range of 0.02 to 72 mN/m. Good precision and accuracy are found for systems with interfacial tensions measured in this range.
- 3) Interfacial tensions have been determined for microbial cell free supernatants using a medium and heavy crude oil. The cell free supernatants did lower the IFT of crude oil and water. The lowest IFT measured was 1.01 ± 0.04 mN/m for Lick Creek crude oil and the cell free supernatant from Bacillus licheniformis JF-2 grown aerobically in TSB. In no case did the cell free supernatants display ultra-low IFTs ($< 10^{-3}$ mN/m), generally accepted as required for excellent tertiary oil recovery.
- 4) Crude biosurfactant obtained from cultures of Bacillus licheniformis JF-2 lowered the IFTs between medium and heavy crude oils and water; the concentrations tested were below the CMC. IFTs increased for both crude oils as the NaCl concentration was increased from 0 to 10% at a crude surfactant concentration of 150 mg/mL.

4. REFERENCES

1. "Oil Research Program Implementation Plan," U.S. Department of Energy, DOE/FE-01889, April 1990, pp.5-6.
2. N. R. Morrow, *Interfacial Phenomena in Petroleum Recovery*, N. R. Morrow (ed.), Surfactant Science Series, V. 36, New York: Marcel Dekker, Inc., 1991, p. 6.
3. L. W. Lake, *Enhanced Oil Recovery*, Prentice Hall: New Jersey, 1989.
4. J. C. Melrose and C. F. Brandner, "Role of Capillary Forces in Determining Microscopic Displacement Efficiency for Oil Recovery," *Journal of Canadian Petroleum Technology*, 13, 1974, p.54.
5. D. O. Shah, *Improved Oil Recovery by Surfactant and Polymer Flooding*, Academic Press: New York, 1977, p.383.
6. R. N. Healy, R. L. Reed, and C. W. Carpenter, "A Laboratory Study of Microemulsion Flooding," *Society of Petroleum Engineering Journal*, 15, 1975, p.87.
7. M. J. Rosen, *Structure/Performance Relationships in Surfactants*, Washington, D. C.: American Chemical Society, 1984, p.329.
8. R. D. Bagnall and P. A. Arundel, "The Profile Area of Pendant Drops," *Journal of Physical Chemistry*, 82, 1978, p.898.
9. C. E. Stauffer, "The Measurement of Surface Tension by the Pendant Drop Technique," *Journal of Physical Chemistry*, 69, 1965, p.1933.
10. C. Huh and R. L. Reed, "A Method for Estimating Interfacial Tensions and Contact Angles from Sessile and Pendant Drops," *Journal of Colloid and Interface Science*, 91, 1983, p.472.
11. K. L. Mittal, *Adsorption at Interfaces*, Washington, D. C.: American Chemical Society, 1974.
12. A. M. Worthington, "On Pendant Drops," *Proceedings of the Royal Society*, 32, 1881, p.362.
13. A. M. Worthington, "Note on a Point in the Theory of Pendant Drops," *Philosophical Magazine*, 19, 1885, p.46.
14. A. Ferguson, "Photographic Measurements of Pendant Drops," *Philosophical Magazine*, 23, 1912, p.417.
15. S. Inoue, *Video Microscopy*, New York: Plenum Press, 1987.
16. J. Satherley, H. H. J. Girault, and D. J. Schiffrin, "The Measurement of Ultralow Interfacial Tension by Video Digital Techniques," *Journal of Colloid and Interface Science*, 136, 1990, p.574.

17. S. H. Anastasiadis, J.-K. Chen, J. T. Koberstein, A. F. Siegel, J. E. Sohn, and J. A. Emerson, "The Determination of Interfacial Tension by Video Image Processing of Pendant Fluid Drops," *Journal of Colloid and Interface Science*, *119*, 1987, p.55.
18. H. H. J. Girault, D. J. Schiffrin, and B. D. V. Smith, "The Measurement of Interfacial Tension of Pendant Drops Using a Video Image Profile Digitizer," *Journal of Colloid and Interface Science*, *101*, 1984, p.257.
19. W. van der Vegt, H. C. van der Mei, J. Noordmans, and H. J. Busscher, "Assessment of Bacterial Biosurfactant Production through Axisymmetric Drop Shape Analysis by Profile," *Applied Microbiological Biotechnology*, *35*, 1991, p.766.
20. J. C. Melrose, "Interfacial Phenomena as Related to Oil Recovery Mechanisms," *Canadian Journal of Chemical Engineering*, *48*, 1970, pp.638-644.
21. J. C. Melrose, *Langmuir*, *3*, 1987, pp.661-667.
22. J. C. Melrose and C. F. Brandner, *Journal of Canadian Petroleum Technology*, *13*, 1974, pp.54-62.
23. N. R. Morrow, "Physics and Thermodynamics of Capillary Action in Porous Media," *Industrial Engineering Chemistry*, *62*, 1970, pp.32-56.
24. R. D. Vold and M. J. Vold, *Colloid and Interface Chemistry*, Reading: Addison-Wesley Publishing Company, 1983.
25. A. W. Adamson, *Physical Chemistry of Surfaces*, 3rd ed., New York: John Wiley, 1976.
26. E. Matijevic, *Surface and Colloid Science*, V.1, New York: Wiley-Interscience, 1969, pp.39-252.
27. A. W. Adamson, *Physical Chemistry of Surfaces*, New York: Interscience Publishers, 1976.
28. M. J. Jaycock and G. D. Parfitt, *Chemistry of Interfaces*, New York: John Wiley and Sons, 1981.
29. F. Bashford and J. C. Adams, *An Attempt to Test the Theories of Capillary Action*, Cambridge: University Press, 1883.
30. J. M. Andreas, E. A. Hansen, and W. B. Tucker, "Boundary Tension by Pendant Drops," *Journal of Physical Chemistry*, *42*, 1938, p.1001.
31. S. Fordham, "On the Calculation of Surface Tension from Measurements of Pendant Drops," *Proceedings of the Royal Society*, *194A*, 1948, p.1.
32. F. K. Hansen and G. Rodsrud, "Surface Tension By Pendant Drop. I. A Fast Standard Instrument Using Computer Image Analysis," *Journal of Colloid and Interface Science*, *141*, 1991, pp.1-9.
33. J. W. Jennings, Jr. and N. R. Pallas, "An Efficient Method for the Determination of Interfacial Tensions from Drop Profiles," *Langmuir*, *4*, 1988, p.959.

34. P. Cheng, D. Li, L. Boruvka, Y. Rotenberg, and A. W. Neumann, "Automation of Axisymmetric Drop Shape Analysis for Measurement of Interfacial Tensions and Contact Angles," *Colloids and Surfaces*, 43, 1990, p.151.
35. N. R. Pallas and Y. Harrison, "An Automated Drop Shape Apparatus and the Surface Tension of Pure Water," *Colloids and Surfaces*, 43, 1990, p.169.
36. D. C. Nguyen, I. R. McKinnon, and I. R. Wilson, "Internal Illumination of Pendant Drops for the Measurement of Interfacial Tensions," *Journal of Colloid and Interface Science*, 119, 1987, p.275.
37. R. C. Weast, *Handbook of Chemistry and Physics*, 54th edition, Cleveland: CRC Press, 1973.
38. H. M. Backes, J. J. Ma, E. Bender, and G. Maurer, "Interfacial Tensions in Binary and Ternary Liquid-Liquid Systems," *Chemical Engineering Science*, 45, 1, 1990, pp.275-286.
39. J. A. Riddick and W. B. Bunger, *Organic Solvents: Physical Properties and Methods of Purification*, New York: Wiley-Interscience, 1970, p. 67.
40. H. Arkin and R. R. Colton, *Statistical Methods*, New York: Barnes & Noble, 1970, p.146.
41. S. M. Selby, *Standard Mathematical Tables*, Cleveland: Chemical Rubber Company, 1969, p. 614.
42. H. D. Young, *Statistical Treatment of Experimental Data*, New York: McGraw-Hill Book Company, Inc., 1962, p. 17.
43. J. Livingston, R. Morgan, et. al., *Journal of the American Chemical Society*, 35, 1913, p.1821.
44. J. J. Jasper and E. V. Kring, *Journal of Physical Chemistry*, 59, 1955, p.1019.
45. D. Villers and J. K. Platten, "Temperature Dependence of the Interfacial Tension between water and Long-Chain Alcohols," *Journal of Physical Chemistry*, 92, 1988, p.4023.

Appendix A

Program Listing for MEOR4D.C

```
/*-----*/
/* This is derived from a program originally written by
   A. D. Watkins in January 1991. It was mostly re-written by
   Gordon Lassahn in March 1991. */

/*DESCRIPTION: This is development software for the Microbial
   Enhanced Oil Recovery project. This algorithm will drive the
   equipment which will measure the surface tension by the pendant
   drop technique. It is written specifically for use with the Data
   Translation DT-2853 low cost frame grabber, and a CCD camera with
   a resolution of 512x512 pixels. */

/*-----*/

#define GDL 1
#define B7 2
#define WATKINS 3
#include <system.def>
/* system.def defines SYSTEM */
/* SYSTEM = GDL or B7 or WATKINS */

#if SYSTEM == WATKINS
#include <c:\c\include\stdio.h>
#include <c:\c\include\process.h>
#include <c:\c\include\conio.h>
#include <c:\c\include\math.h>
#include <c:\c\include\graph.h>
#include <c:\c\include\bios.h>
#include <c:\c\include\stdlib.h>
#include "c:\vision\isdefs.c"
#define FRAME1 0
#endif

#if SYSTEM == B7
#include <c:\qc2\include\stdio.h>
#include <c:\qc2\include\process.h>
#include <c:\qc2\include\conio.h>
#include <c:\qc2\include\math.h>
#include <c:\qc2\include\graph.h>
#include <c:\qc2\include\bios.h>
#include <c:\qc2\include\stdlib.h>
#include <c:\qc2\source\isdefs.c>
#define FRAME1 1
#endif
```

```

#if SYSTEM == GDL
#include <c:\qc25\include\stdio.h>
#include <c:\qc25\include\process.h>
#include <c:\qc25\include\conio.h>
#include <c:\qc25\include\math.h>
#include <c:\qc25\include\graph.h>
#include <c:\qc25\include\bios.h>
#include <c:\qc25\include\stdlib.h>
#include <c:\librarys\isdefs.c>
#define FRAME1 1
#endif

void diagnose();
void init();
void focus();
void monitor();
void calibrate();
void analyze();
void set_base();
char *fit_width();
char *fit_height();
char *fit_apex();
void find_center();
void find_sides();
void find_top();
void find_both();
void graph();
void graph2();
void screen1();
void screen2();
float tension();
extern int defext();
extern int in_string();
extern void get_row();
extern void put_row();
extern void get_col();
extern void put_col();
extern float avgv();
extern float avgi();
extern void save();
extern void restore();
extern void datim();
extern int med001();
extern int fit_quadratic();

#define FDIM 50
/* dimension of some floating point arrays */

```

```

int white=255; /* brightest intensity value on video display */
int step_height=150, step_width=4;
/* at the edge of a drop, the intensity changes by step_height
   within a distance of step_width pixels */
int iaa[512], iab[512];
float faa[FDIM], fab[FDIM], fac[FDIM], fad[FDIM];
int cursor_on_off=0, cursor_row=240, cursor_col=250;
int cal_flag=0;
float horiz_cal=0.0, vert_cal=0.0;
char cal_time[21];
FILE *fp; /* file pointer */
char dt[21]; /* date & time string */
char filename[50]; /* file name string */
float row_mid, row_top, row_bot, row_left, row_rite, height;
float col_mid, col_top, col_bot, col_left, col_rite, width;
int i_mid, i_top, i_bot, i_left, i_rite, i_base=450;
int j_mid, j_top, j_bot, j_left, j_rite;
int mode;
unsigned yes_no=0;
/*-----*/
void main ()
{int count=0;
 unsigned key;
 printf("\n MEOR version 4d\n revised 16 March 1991\n");
 init ();
 screen1();
 while(count <=0)
 {key=wait_for_key();
 switch(key)
 {
 case 15104:          /* F1 KEY - GRAB AN IMAGE */
 if (mode==2) IS_FREEZE_FRAME();
 mode=1;
 IS_SET_SYNC_SOURCE(0);
 screen1();
 break;

 case 15360:          /* F2 KEY _ PASS-THROUGH MODE */
 mode=2;
 IS_SET_SYNC_SOURCE(1);
 IS_PASSTHRU();
 screen1();
 break;

 case 15616:          /* F3 KEY - MONITOR DROP EDGE POSITIONS */
 if (mode==2) IS_FREEZE_FRAME();
 mode=3;
 IS_SET_SYNC_SOURCE(1);

```

```

monitor(0);
mode=2;
screen1();
break;

case 22016:/* SHIFT F3 KEY - MONITOR DROP EDGE POSITIONS */
    /* AND WRITE RESULTS TO A DISK FILE */
    if (mode==2) IS_FREEZE_FRAME();
    mode=13;
    IS_SET_SYNC_SOURCE(1);
    monitor(1);
    mode=2;
    screen1();
    break;

case 15872:          /* F4 KEY - TO FOCUS CAMERA */
    if (mode==2) IS_FREEZE_FRAME();
    mode=4;
    IS_SET_SYNC_SOURCE(1);
    focus();
    mode=2;
    screen1();
    break;

case 16128:          /* F5 KEY - CALIBRATES SYSTEM */
    if (mode==2) IS_FREEZE_FRAME();
    mode=5;
    IS_SET_SYNC_SOURCE(0);
    calibrate();
    mode=1;
    screen1();
    break;

case 16384: /* F6 KEY -- CALCULATE INTERFACIAL TENSION */
    if (mode==2) IS_FREEZE_FRAME();
    mode=6;
    IS_SET_SYNC_SOURCE(0);
    analyze();
    mode=1;
    screen1();
    break;

case 16640: /* F7 KEY -- RESTORE AN IMAGE FROM DISK */
    if (mode==2) IS_FREEZE_FRAME();
    mode=7;
    IS_SET_SYNC_SOURCE(0);
    restore();
    mode=1;

```

```

    screen1();
    break;

case 16896:          /* F8 KEY - SAVE IMAGE TO DISK */
    if (mode==2) IS_FREEZE_FRAME();
    mode=8;
    IS_SET_SYNC_SOURCE(0);
    save();
    mode=1;
    screen1();
    break;

case 17152: /* F9 KEY - EXECUTE A MS-DOS SYSTEM COMMAND */
    printf(" TYPE SYSTEM COMMAND:\n");
    in_string(filename,50);
    system(filename);
    printf(" HIT 'ENTER'\n");
    wait_for_key();
    screen1();
    break;

case 17408:          /* F10 KEY - EXIT */
#if SYSTEM == B7
    IS_SET_SYNC_SOURCE(1);
    IS_PASSTHRU();
#else
    if (mode==2) IS_FREEZE_FRAME();
    IS_SET_SYNC_SOURCE(0);
#endif
    count=1;
    break;

case 21504:          /* SHIFT F1 KEY - GRAPHS CURSOR ROW */
    if (mode==2) IS_FREEZE_FRAME();
    mode=11;
    IS_SET_SYNC_SOURCE(0);
    get_row(cursor_row,iaa);
    graph(iaa,512);
    mode=1;
    screen1();
    break;

case 21760:          /* SHIFT F2 KEY - GRAPHS CURSOR COLUMN */
    if (mode==2) IS_FREEZE_FRAME();
    mode=12;
    IS_SET_SYNC_SOURCE(0);
    get_col(cursor_col,iaa);

```

```

graph(iaa,480);
mode=1;
screen1();
break;

case 23040: /* SHIFT F7 KEY - GRAPH CURSOR ROW & COLUMN */
if (mode==2) IS_FREEZE_FRAME();
mode=17;
get_row(cursor_row,iaa);
get_col(cursor_col,iab);
graph2(1);
mode=1;
screen1();
break;

case 23296: /* SHIFT F8 KEY - TOGGLES CURSOR ON OR OFF */
IS_SET_CURSOR_POSITION(cursor_row,cursor_col);
if(cursor_on_off==1) cursor_on_off=0;
else cursor_on_off=1;
IS_CURSOR(cursor_on_off);
break;

case 23552: /* SHIFT F9 KEY - DIAGNOSTIC DISPLAY */
diagnose(" diagnostics:");
screen1();
break;

case 18432: /* UP ARROW KEY - MOVES CURSOR UP */
if (cursor_row>0) cursor_row--;
IS_SET_CURSOR_POSITION(cursor_row,cursor_col);
break;

case 20480: /* DWN ARROW KEY - MOVES CURSOR DWN */
if (cursor_row<479) cursor_row++;
IS_SET_CURSOR_POSITION(cursor_row,cursor_col);
break;

case 19200: /* LEFT ARROW KEY - MOVES CURSOR LEFT */
if (cursor_col>0) cursor_col--;
IS_SET_CURSOR_POSITION(cursor_row,cursor_col);
break;

case 19712: /* RIGHT ARROW KEY - MOVES CURSOR RIGHT */
if (cursor_col<511) cursor_col++;
IS_SET_CURSOR_POSITION(cursor_row,cursor_col);
break;

case 23808: /* SHIFT F10 -- DISPLAY SECOND MENU */

```

```

        screen2();
        break;

    case 7181:          /* ENTER -- DISPLAY MAIN MENU */
        screen1();
        break;

    } /*END SWITCH STATEMENT */
} /* END WHILE LOOP */
IS_END();
} /* END main() FUNCTION */

/*-----*/
void init ()
{int i;
  IS_INITIALIZE();
  IS_SELECT_INPUT_FRAME(0);
  IS_SELECT_OUTPUT_FRAME(0);
  IS_DISPLAY(1);
  #if SYSTEM == B7
  IS_SET_SYNC_SOURCE(1);
  IS_PASSTHRU();
  mode=2;
  #else
  IS_SET_SYNC_SOURCE(0);
  mode=1;
  #endif
  if ((fp=fopen("meor.cal","r"))!=NULL)
  {fscanf(fp," %e\n %e\n",&horiz_cal,&vert_cal);
   for (i=0; i<20; i++) cal_time[i]=fgetc(fp);
   cal_time[20]='\0';
   printf("\n CALIBRATION DONE AT %20.20s HORIZONTAL %e  VERTICAL %e\n",
          cal_time,horiz_cal,vert_cal);

   fclose(fp);
   cal_flag=1;
  }
  else
  {printf(" NO PREVIOUS CALIBRATION AVAILABLE\n YOU MUST CALIBRATE!\n");
  }
}
/*-----*/

void screen1 ()
{printf("\n$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\n");
 printf(" PRESS:\n");
 if (mode==1) printf("> > > >");
   else printf(" ");
 printf(" F1 -- TO WORK WITH FROZEN IMAGE\n");

```



```

if (mode==2) printf("> > > ");
    else printf(" ");
printf(" F2 -- TO DISPLAY LIVE IMAGE\n");
if (mode==3) printf("> > > ");
    else printf(" ");
printf(" F3 -- TO MONITOR DROP STABILITY\n");
/* if (mode==13) printf("> > > ");
    else printf(" ");
printf(" SHIFT F3 -- TO MONITOR DROP STABILITY & WRITE FILE\n"); */
if (mode==4) printf("> > > ");
    else printf(" ");
printf(" F4 -- TO FOCUS CAMERA\n");
if (mode==5) printf("> > > ");
    else printf(" ");
printf(" F5 -- TO CALIBRATE SYSTEM\n");
if (mode==6) printf("> > > ");
    else printf(" ");
printf(" F6 -- TO CALCULATE INTERFACIAL TENSION\n");
if (mode==7) printf("> > > ");
    else printf(" ");
printf(" F7 -- TO RESTORE AN IMAGE FROM A DISK FILE\n");
if (mode==8) printf("> > > ");
    else printf(" ");
printf(" F8 -- TO SAVE AN IMAGE TO A DISK FILE\n");
if (mode==9) printf("> > > ");
    else printf(" ");
printf(" F9 -- TO EXECUTE AN MS-DOS SYSTEM COMMAND\n");
printf(" SHIFT F10 -- TO SEE DIAGNOSTIC MENU\n");
printf(" F10 -- TO STOP THIS PROGRAM\n");
printf("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\n\n");
}
/*-----*/
void screen2 ()
{printf("\n$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\n");
printf(" PRESS:\n");
printf(" CURSOR KEYS -- TO MOVE CURSOR\n");
printf(" SHIFT F1 -- TO GRAPH CURSOR ROW\n");
printf(" SHIFT F2 -- TO GRAPH CURSOR COLUMN\n");
printf(" SHIFT F3 -- TO MONITOR DROP STABILITY & WRITE FILE\n");
printf(" SHIFT F7 -- TO PLOT CURSOR ROW & COLUMN\n");
printf(" SHIFT F8 -- TO TOGGLE CURSOR ON/OFF\n");
printf(" SHIFT F9 -- TO DISPLAY SOME VARIABLES\n");
printf(" ENTER -- TO DISPLAY MAIN MENU\n");
printf("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\n\n");
}
/*-----*/
float tension (sigma)
/* calculate R0*R0/De/De/beta */

```

```

/* this version from F. K. Hansen and G. Rodsrud,
   "Surface Tension by Pendant Drop", Journal of Colloid
   and Interface Science, Vol. 141, No. 1, January 1991. */
float sigma; /* neck/width; Ds/De */
{float beta, tem;
  beta=0.12836+sigma*(-0.7577+sigma*(1.7713-sigma*0.5426));
  tem=0.9987+beta*(0.1971+beta*(-0.0734+beta*0.3408));
  tem=0.25/beta/tem/tem;
  return tem;
}
/*-----*/
float edge_inc (y,n)
/* find the location of the first increasing step in the array y,
   starting with element 0 and going to element n. */
int y[], n;
{int i, j, j_min, j_max;
  float deriv, sumy, sumyj;
  for (i=0; i<n-step_width; i++)
    {if ((y[i+step_width]-y[i]) >= step_height)
      {j_min=max(i-step_width/2,0);
       j_max=min(i+step_width*3/2,n-1);
       sumy=0.0;
       sumyj=0.0;
       for (j=j_min; j<j_max; j++)
         {deriv=(float)max((y[j+1]-y[j]),0);
          sumy+=deriv;
          sumyj+=deriv*(float)j;
         }
       return (sumyj/sumy+0.5);
      }
    }
  return (-1.0);
}
/*-----*/
float edge_dec (y,n)
/* find the location of the first increasing step in the array y,
   starting with element n-1 and going to element 0. */
int y[], n;
{int i, j, j_min, j_max;
  float deriv, sumy, sumyj;
  for (i=n-1-step_width; i>=0; i--)
    {if ((y[i]-y[i+step_width]) >= step_height)
      {j_min=max(i-step_width/2,0);
       j_max=min(i+step_width*3/2,n-1);
       sumy=0.0;
       sumyj=0.0;
       for (j=j_min; j<j_max; j++)
         {deriv=(float)max((y[j]-y[j+1]),0);

```

```

        sumy += deriv;
        sumyj += deriv*(float)j;
    }
    return (sumyj/sumy + 0.5);
}
}
return (-1.0);
}
/*-----*/
void find_sides (i,ptrleft,ptrrite,ptrwide)
/* find the left & right edges and the width, at row i. */
int i;
float *ptrleft, *ptrrite, *ptrwide;
{get_row(i,iaa);
 med001(iaa,iab,512,9);
 *ptrleft=edge_dec(iab,j_mid);
 *ptrrite=edge_inc(&iab[j_mid],512-j_mid)+(float)j_mid;
 *ptrwide= *ptrrite-*ptrleft;
#if SYSTEM == B7
 IS_PUT_PIXEL(FRAME1,i,(int)(*ptrleft+1.0),1,&white);
 IS_PUT_PIXEL(FRAME1,i,(int)(*ptrrite),1,&white);
#else
 IS_PUT_PIXEL(FRAME1,i,(int)(*ptrleft+0.5),1,&white);
 IS_PUT_PIXEL(FRAME1,i,(int)(*ptrrite+0.5),1,&white);
#endif
}
/*-----*/
void find_top (j,ptrtop)
/* find the top edge, at column j */
int j;
float *ptrtop;
{get_col(j,iaa);
 med001(iaa,iab,480,13);
 /* graph(iab,i_mid); */
 *ptrtop=edge_dec(iab,i_mid);
#if SYSTEM == B7
 IS_PUT_PIXEL(FRAME1,(int)(*ptrtop+1.0),j,1,&white);
#else
 IS_PUT_PIXEL(FRAME1,(int)(*ptrtop+0.5),j,1,&white);
#endif
}
/*-----*/
void find_both (j,ptrtop,ptrbot,ptrhite)
/* find the top and the bottom edges and the height, at column j. */
int j;
float *ptrtop, *ptrbot, *ptrhite;
{get_col(j,iaa);
 med001(iaa,iab,480,11);

```

```

*ptrtop=edge_dec(iab,i_mid);
*ptrbot=edge_inc(&iab[i_mid],480-i_mid)+(float)i_mid;
*ptrhite=*ptrbot-*ptrtop;
#if SYSTEM == B7
IS_PUT_PIXEL(FRAME1,(int)(*ptrtop+1.0),j,1,&white);
IS_PUT_PIXEL(FRAME1,(int)(*ptrbot),j,1,&white);
#else
IS_PUT_PIXEL(FRAME1,(int)(*ptrtop+0.5),j,1,&white);
IS_PUT_PIXEL(FRAME1,(int)(*ptrbot+0.5),j,1,&white);
#endif
}
/*-----*/
char *fit_width ()
/* find the width of the drop at its widest row. */
/* use and change the global variables i_mid and j_mid. */
/* set the global variables row_mid, col_mid, row_left, col_left,
   row_rite, col_rite, and width. */
{int delta_i, np, i, i_min, i_max;
float w_min, w_max, rowmid, colleft, colrite, wide, a, b, c;
np=0;
find_sides(i_mid,&faa[np],&fab[np],&fac[np]);
if (faa[np] < 0.0) return ("fit_width error 1.1");
if (fab[np] < 0.0) return ("fit_width error 2.1");
wide=fac[np];
delta_i=max(((int)wide/10),1);
i_min=i_mid;
w_min=wide;
i_max=i_mid;
w_max=wide;
colleft=faa[0];
colrite=fab[0];
np=1;
while ((i_mid <= i_min) || (i_mid >= i_max)
|| (w_min > (wide-5.0)) || (w_max > (wide-5.0)))
{if (np == FDIM) return ("fit_width error 4.1");
if (w_min >= w_max)
{i_min-=delta_i;
if (i_min <= i_top) return ("fit_width error 5.1");
find_sides(i_min,&faa[np],&fab[np],&fac[np]);
if (faa[np] < 0.0) return ("fit_width error 1.2");
if (fab[np] < 0.0) return ("fit_width error 2.2");
fad[np]=(float)i_min;
w_min=fac[np];
if (w_min > wide)
{wide=w_min;
i_mid=i_min;
colleft=faa[np];
colrite=fab[np];

```

```

    }
}
else
{
    i_max += delta_i;
    if (i_max >= i_base) return ("fit_width error 5.2");
    find_sides(i_max,&faa[np],&fab[np],&fac[np]);
    if (faa[np] < 0.0) return ("fit_width error 1.3");
    if (fab[np] < 0.0) return ("fit_width error 2.3");
    fad[np]=(float)i_max;
    w_max = fac[np];
    if (w_max > wide)
    {
        wide=w_max;
        i_mid=i_max;
        colleft=faa[np];
        colrite=fab[np];
    }
}
np++;
}
i_min=max(i_mid-10,i_top);
i_max=min(i_mid+10,i_base);
np=i_max-i_min+1;
if (np < 3) return ("fit_width error 7.1");
rowmid=(float)i_mid;
/* printf(" %f %f %f %f\n",colleft,colrite,wide,rowmid); */
np=0;
for (i=i_min; i <= i_max; i++)
{
    find_sides(i,&faa[np],&fab[np],&fac[np]);
    if (faa[np] < 0.0) return ("fit_width error 1.6");
    if (fab[np] < 0.0) return ("fit_width error 2.6");
    faa[np]-=colleft;
    fab[np]-=colrite;
    fac[np]-=wide;
    fad[np]=(float)i-rowmid;
    /* printf(" %f %f %f %f\n",fad[np],faa[np],fab[np],fac[np]); */
    np++;
}
if (fit_quadratic(fad,faa,np,&a,&b,&c,&row_left,&col_left) < 0)
{
    printf("fit_width warning 3.3 left side\n");
    row_left=rowmid;
    col_left=colleft;
}
else
{
    row_left=min(max(row_left,(-10.0)),(10.0));
    col_left=a+row_left*(b+row_left*c)+colleft;
    row_left+=rowmid;
}
if (fit_quadratic(fad,fab,np,&a,&b,&c,&row_rite,&col_rite) < 0)

```

```

    {printf("fit_width warning 3.4 right side\n");
    row_rite=rowmid;
    col_rite=colrite;
    }
else
    {row_rite=min(max(row_rite,(-10.0)),(10.0));
    col_rite=a+row_rite*(b+row_rite*c)+colrite;
    row_rite+=rowmid;
    }
if (fit_quadratic(fad,fac,np,&a,&b,&c,&row_mid,&width) < 0)
    {printf("fit_width warning 3.5 width\n");
    row_mid=rowmid;
    width=wide;
    }
else
    {row_mid=min(max(row_mid,(-10.0)),(10.0));
    width=a+row_mid*(b+row_mid*c)+wide;
    row_mid+=rowmid;
    }
col_mid=(col_left+col_rite)/2.0;
j_mid=(int)(col_mid+0.5);
i_mid=(int)(row_mid+0.5);
/* diagnose("exit fit_width"); */
return (NULL);
}
/*-----*/
void find_center ()
/* find a first estimate of the position of the drop center. */
/* set global variables i_mid and j_mid. */
{int i, intavg;
j_mid=250;
get_col(j_mid,iaa);
med001(iaa,iab,480,11);
intavg=(int)avgv(iab,480);
for (i=0; i<480; i++)
    iab[i]=max((intavg-iab[i]),0);
i_mid=(int)avgi(iab,480);
/* diagnose("find_center exit"); */
}
/*-----*/
char *fit_height ()
/* find the height of the ball at its highest column. */
/* use the global variables col_left, col_rite, and width. */
/* set the global variables row_top, col_top, row_bot, col_bot,
row_mid, and height. */
{int delta_j, np, j, j_min, j_max;
float h_min, h_max, colmid, rowtop, rowbot, hite, a, b, c, dummy;
j_mid=(int)((col_left+col_rite)/2.0+0.5);

```

```

j_left=(int)((3.0*col_left+col_rite)/4.0);
j_rite=(int)((3.0*col_rite+col_left)/4.0);
delta_j=max(((int)width/20),1);
/* printf(%d %d %d %d\n",j_left,j_mid,j_rite,delta_j); */
find_both(j_mid,&faa[0],&fab[0],&fac[0]);
if (faa[0] < 0.0) return ("fit_height error 1.2");
if (fab[0] < 0.0) return ("fit_height error 2.2");
hite=fac[0];
j_min=j_mid;
h_min=hite;
j_max=j_mid;
h_max=hite;
rowtop=faa[0];
rowbot=fab[0];
np=1;
while ((j_mid <= j_min) || (j_mid >= j_max)
      || (h_min > (hite-5.0)) || (h_max > (hite-5.0)))
{if (np == FDIM) return ("fit_height error 4.1");
 if (h_min > h_max)
 {j_min-=delta_j;
  if (j_min <= j_left) return ("fit_height error 5.1");
  find_both(j_min,&faa[np],&fab[np],&fac[np]);
  if (faa[np] < 0.0) return ("fit_height error 1.2");
  if (fab[np] < 0.0) return ("fit_height error 2.2");
  h_min=fac[np];
  if (h_min > hite)
  {hite=h_min;
   j_mid=j_min;
   rowtop=faa[np];
   rowbot=fab[np];
  }
 }
 else
 {j_max+=delta_j;
  if (j_max >= j_rite) return ("fit_height error 6.1");
  find_both(j_max,&faa[np],&fab[np],&fac[np]);
  if (faa[np] < 0.0) return ("fit_height error 1.3");
  if (fab[np] < 0.0) return ("fit_height error 2.3");
  h_max=fac[np];
  if (h_max > hite)
  {hite=h_max;
   j_mid=j_max;
   rowtop=faa[np];
   rowbot=fab[np];
  }
 }
 np++;
}

```

```

j_min=max(j_mid-10,j_left);
j_max=min(j_mid+10,j_rite);
np=j_max-j_min+1;
if (np < 3) return ("fit_height error 7.1");
colmid=(float)j_mid;
np=0;
for (j=j_min; j <= j_max; j++)
{find_both(j,&faa[np],&fab[np],&fac[np]);
if (faa[np] < 0.0) return ("fit_height error 1.6");
if (fab[np] < 0.0) return ("fit_height error 2.6");
faa[np]-=rowtop;
fab[np]-=rowbot;
fac[np]-=hite;
fad[np]=(float)j-colmid;
np++;
}
if (fit_quadratic(fad,faa,np,&a,&b,&c,&col_top,&row_top) < 0)
{printf("fit_height warning 3.3 top\n");
col_top=colmid;
row_top=rowtop;
}
else
{col_top=min(max(col_top,(-10.0)),(10.0));
row_top=a+col_top*(b+col_top*c)+rowtop;
col_top+=colmid;
}
if (fit_quadratic(fad,fab,np,&a,&b,&c,&col_bot,&row_bot) < 0)
{printf("fit_height warning 3.4 bottom\n");
col_bot=colmid;
row_bot=rowbot;
}
else
{col_bot=min(max(col_bot,(-10.0)),(10.0));
row_bot=a+col_bot*(b+col_bot*c)+rowbot;
col_bot+=colmid;
}
if (fit_quadratic(fad,fac,np,&a,&b,&c,&colmid,&height) < 0)
{printf("fit_height warning 3.5 height\n");
height=hite;
}
else
{colmid=min(max(colmid,(-10.0)),(10.0));
height=a+colmid*(b+colmid*c)+hite;
}
row_mid=(row_top+row_bot)/2.0;
return (NULL);
}
/*-----*/

```



```

char *fit_apex ()
/* find the apex of the drop at its highest column. */
/* use the global variables col_left, col_rite, and width. */
/* set the global variables row_top and col_top. */
{int delta_j, np, j, j_min, j_max;
float h_min, h_max, coltop, rowtop, a, b, c, dummy;
/* diagnose("enter fit_apex"); */
j_top=(int)((col_left+col_rite)/2.0);
j_left=(int)((3.0*col_left+col_rite)/4.0);
j_rite=(int)((3.0*col_rite+col_left)/4.0);
delta_j=max(((int)width/20),1);
find_top(j_top,&faa[0]);
if (faa[0] < 0.0) return ("fit_apex error 1.2");
rowtop=faa[0];
j_min=j_top;
h_min=rowtop;
j_max=j_top;
h_max=rowtop;
np=3;
while ((j_top <= j_min) || (j_top >= j_max)
|| ((h_min < (rowtop+5.0)) || (h_max < (rowtop+5.0))))
{if (np == FDIM) return ("fit_apex error 4.1");
if (h_min < h_max)
{j_min-=delta_j;
if (j_min <= j_left) return ("fit_apex error 5.1");
find_top(j_min,&faa[np]);
if (faa[np] < 0.0) return ("fit_apex error 1.2");
h_min=faa[np];
if (h_min < rowtop)
{rowtop=h_min;
j_top=j_min;
}
}
else
{j_max+=delta_j;
if (j_max >= j_rite) return ("fit_apex error 6.1");
find_top(j_max,&faa[np]);
if (faa[np] < 0.0) return ("fit_apex error 1.3");
h_max=faa[np];
if (h_max < rowtop)
{rowtop=h_max;
j_top=j_max;
}
}
np++;
}
j_min=max(j_top-10,j_left);
j_max=min(j_top+10,j_rite);

```

```

np=j_max-j_min+1;
if (np < 3) return ("fit_apex error 7.1");
coltop=(float)j_top;
np=0;
for (j=j_min; j <=j_max; j++)
{find_top(j,&faa[np]);
if (faa[np] < 0.0) return ("fit_apex error 1.6");
faa[np]-=rowtop;
fad[np]=(float)j-coltop;
np++;
}
if (fit_quadratic(fad,faa,np,&a,&b,&c,&col_top,&row_top) < 0)
{printf("fit_apex warning 3.3\n");
col_top=coltop;
row_top=rowtop;
}
else
{col_top=min(max(col_top,(-10.0)),(10.0));
row_top=a+col_top*(b+col_top*c)+rowtop;
col_top+=coltop;
}
return (NULL);
}
/*-----*/
void analyze ()
/* measure the drop and calculate the interfacial tension. */
{char *errstr;
static int trial=0;
static float temp=0.0, density_diff=0.0;
static char strdum[50]={ ' ' }, sample[50]={ ' ' }, operator[50]={ ' ' };
int i, i_neck, i_min, i_max, np;
float row_neck, a, b, c, neck, S, recip_H, ift;
float colleft, colrite, wide, fdum1, fdum2;
set_base();
printf("\nDON'T BOTHER ME, I'M THINKING about interfacial tension.\n");
#if SYSTEM == GDL
IS_RESET();
IS_SELECT_INPUT_FRAME(0);
IS_DISPLAY(1);
#endif
IS_SELECT_OUTPUT_FRAME(FRAME1);
IS_FRAME_COPY(0,FRAME1);
#if SYSTEM == GDL
IS_DIVIDE_CONSTANT(0,2,FRAME1);
#endif
find_center(); /* set i_mid & j_mid */
errstr=fit_width(); /* set col_left, col_rite, col_mid, width */
/* set row_left, row_rite, row_mid */

```

```

if (errstr != NULL)
{printf("I CAN'T FIND THE WIDTH OF THIS DROP\n%s\n",errstr);
IS_SELECT_OUTPUT_FRAME(0);
return;
}
errstr=fit_apex(); /* set row_top & col_top */
if (errstr != NULL)
{printf("I CAN'T FIND THE TOP OF THIS DROP\n%s\n",errstr);
IS_SELECT_OUTPUT_FRAME(0);
return;
}
row_neck=row_top+width*horiz_cal/vert_cal;
i_neck=(int)(row_neck+0.5);
if (i_neck > i_base)
{printf("analyze error 4: neck is below base\n");
IS_SELECT_OUTPUT_FRAME(0);
return;
}
find_sides(i_neck,&colleft,&colrite,&wide);
if (colleft < 0.0)
{printf("analyze error 1.1, left neck\n");
IS_SELECT_OUTPUT_FRAME(0);
return;
}
if (colrite < 0.0)
{printf("analyze error 2.1, right neck\n");
IS_SELECT_OUTPUT_FRAME(0);
return;
}
i_min=i_neck-5;
i_max=min(i_neck+5,i_base);
np=0;
for (i=i_min; i<=i_max; i++)
{find_sides(i,&colleft,&colrite,&fac[np]);
if (colleft < 0.0)
{printf("analyze error 1.1, left neck\n");
IS_SELECT_OUTPUT_FRAME(0);
return;
}
if (colrite < 0.0)
{printf("analyze error 2.1, right neck\n");
IS_SELECT_OUTPUT_FRAME(0);
return;
}
fac[np]-=wide;
fad[np]=(float)i-row_neck;
np++;
}

```

```

if (fit_quadratic(fad,fac,np,&a,&b,&c,&fdum1,&fdum2) < 0)
{printf("analyze warning 3.1, neck quadratic fit\n");
a=0.0;
}
neck=a+wide;
S=neck/width;
recip_H=tension(S);
printf("Enter difference between fluid densities (g/cm^3) [%f]:\n",
density_diff);
if (in_string(strdum,50)>0) density_diff=atof(strdum);
ift=980.0*density_diff*recip_H*pow((width*horiz_cal*2.54),2);
datim(dt);
printf(" %9.5f = DENSITY DIFFERENCE [g/cc]\n",density_diff);
printf(" %9.5f = MAXIMUM DROP DIAMETER [cm]\n",width*horiz_cal*2.54);
printf(" %9.5f = MEASURED NECK WIDTH [cm]\n",neck*horiz_cal*2.54);
printf(" %9.5f = INTERFACIAL TENSION [dynes/cm]\n\n\n", ift);
#if SYSTEM != B7
printf(" DO YOU WANT THIS RESULT PRINTED ? (Y/N) :\n");
yes_no=wait_for_key();
if ((yes_no == 5497) || (yes_no == 5465))
{printf("\nENTER SAMPLE NAME (%s): ",sample);
if (in_string(strdum,50)>0)
{strepv(sample,strdum);
trial=1;
}
else trial++;
printf("\nENTER TRIAL NUMBER (%d): ",trial);
if (in_string(strdum,50)>0) trial=atoi(strdum);
printf("\nENTER OPERATOR (%s): ",operator);
if (in_string(strdum,50)>0) strepv(operator,strdum);
printf("\nENTER TEMPERATURE [C] [%f]: ",temp);
if (in_string(strdum,50)>0) temp=atof(strdum);
fprintf(stdprn,"RESULTS:\n\n");
fprintf(stdprn," %s = SAMPLE NAME\n", sample);
fprintf(stdprn," %d = TRIAL NUMBER\n", trial);
fprintf(stdprn," %s = OPERATOR\n", operator);
fprintf(stdprn," %s = DATE & TIME\n", dt);
fprintf(stdprn," %9.5f = TEMPERATURE [C]\n", temp);
fprintf(stdprn," %9.5f = DENSITY DIFFERENCE [g/cc]\n",density_diff);
fprintf(stdprn," %9.5f = MAXIMUM DROP DIAMETER [cm]\n",
width*horiz_cal*2.54);
fprintf(stdprn," %9.5f = MEASURED NECK WIDTH [cm]\n",neck*horiz_cal*2.54);
fprintf(stdprn," %9.5f = INTERFACIAL TENSION [dynes/cm]\n\n\n", ift);
}
#endif
printf("\nType file name (default extension = .dat) to save this result, OR\n");
printf("hit CARRIAGE RETURN (ENTER) to NOT save this result:\n");
i=in_string(filename,50);

```

```

if (i > 0)
{
    defext(filename,50,"dat");
    if ((yes_no!=5497)&&(yes_no!=5465))
    {
        printf("\nENTER SAMPLE NAME (%s): ",sample);
        if (in_string(strdum,50)>0)
        {
            strcpy(sample,strdum);
            trial=1;
        }
        else trial++;
        printf("\nENTER TRIAL NUMBER (%d): ",trial);
        if (in_string(strdum,50)>0) trial=atoi(strdum);
        printf("\nENTER OPERATOR (%s): ",operator);
        if (in_string(strdum,50)>0) strcpy(operator,strdum);
        printf("\nENTER TEMPERATURE [C] [%9.5f]: ",temp);
        if (in_string(strdum,50)>0) temp=atof(strdum);
    }
    if((fp = fopen(filename, "a")) == NULL)
    {
        printf("fopen failed      %s\n",filename);
    }
    else
    {
        fprintf(fp,"RESULTS:\n\n");
        fprintf(fp," %s = SAMPLE NAME\n", sample);
        fprintf(fp," %d = TRIAL NUMBER\n", trial);
        fprintf(fp," %s = OPERATOR\n", operator);
        fprintf(fp," %s = DATE & TIME\n", dt);
        fprintf(fp," %9.5f = TEMPERATURE [C]\n", temp);
        fprintf(fp," %9.5f = DENSITY DIFFERENCE [g/cc]\n",density_diff);
        fprintf(fp," %9.5f = MAXIMUM DROP DIAMETER [cm]\n",width*horiz_cal*2.54);
        fprintf(fp," %9.5f = MEASURED NECK WIDTH [cm]\n",neck*horiz_cal*2.54);
        fprintf(fp," %9.5f = INTERFACIAL TENSION [dynes/cm]\n\n\n", ift);
        fclose(fp);
    }
}
IS_SELECT_OUTPUT_FRAME(0);
}
/*-----*/
void calibrate ()
/* measure the ball and calculate the calibration factors. */
{
    char *errstr;
    float diameter;
    printf("\nDON'T BOTHER ME, I'M THINKING about calibration.\n");
#ifdef SYSTEM == GDL
    IS_RESET();
    IS_SELECT_INPUT_FRAME(0);
    IS_DISPLAY(1);
#endif
    IS_SELECT_OUTPUT_FRAME(FRAME1);
    IS_FRAME_COPY(0,FRAME1);
}

```

```

#if SYSTEM == GDL
IS_DIVIDE_CONSTANT(0.2,FRAME1);
#endif
find_center(); /* set i_mid & j_mid */
errstr=fit_width(); /* set col_left, col_rite, col_mid, width */
/* set row_left, row_rite, row_mid */
if (errstr != NULL)
{printf("I CAN'T FIND THE WIDTH OF THIS BALL\n%s\n",errstr);
IS_SELECT_OUTPUT_FRAME(0);
return;
}
errstr=fit_height(); /* set row_top, row_bot, height */
/* set col_top, col_bot */
if (errstr != NULL)
{printf("I CAN'T FIND THE HEIGHT OF THIS BALL\n%s\n",errstr);
IS_SELECT_OUTPUT_FRAME(0);
return;
}
printf("\nTYPE CALIBRATION BALL DIAMETER [INCHES]: ");
scanf("%f",&diameter);
horiz_cal=diameter/width;
vert_cal=diameter/height;
/* calibration factors are [INCHES/PIXEL] */
datim(cal_time);
fp=fopen("meor.cal","w");
fprintf(fp," %e\n %e\n%20.20s\n",horiz_cal,vert_cal,cal_time);
fclose(fp);
printf("CALIBRATION COMPLETE %f %f\n\n",horiz_cal,vert_cal);
printf("HIT 'ENTER' TO CONTINUE");
wait_for_key();
IS_SELECT_OUTPUT_FRAME(0);
}
/*-----*/

void monitor (kode)
int kode; /* kode=1 --> write disk file */
/* Repeatedly, grab a frame and display the pixel positions of
the left edge, the top edge, and the right edge of the drop. */
{float colleft, rowtop, colrite;
if (kode == 1)
{datim(dt);
filename[0]=dt[4];
filename[1]=dt[5];
filename[2]=dt[7];
filename[3]=dt[8];
filename[4]=dt[10];
filename[5]=dt[11];
filename[6]=dt[13];
filename[7]=dt[14];

```

```

filename[8]='.';
filename[9]='m';
filename[10]='o';
filename[11]='n';
filename[12]='\0';
if ((fp=fopen(filename,"w")) == NULL) kode=0;
}
while (kbhit() == 0)
{IS_ACQUIRE(0,1); /* acquire 1 frame in buffer 0 */
datim(dt);
find_center();
find_sides(i_mid,&colleft,&colrite,&rowtop);
find_top(j_mid,&rowtop);
if ((colleft < 0.0) || (rowtop < 0.0) || (colrite < 0.0))
printf("monitor error      HIT 'ENTER' TO STOP\n");
else
printf("%5.1f %5.1f %5.1f HIT 'ENTER' TO STOP\n",
colleft,rowtop,colrite);
if (kode == 1) fprintf(fp," %5.1f %5.1f %5.1f %s\n",
colleft,rowtop,colrite,&dt[10]);
}
if (kode == 1) fclose(fp);
wait_for_key();
}
/*-----*/
void graph (z,n)
/* graph z[k] versus k */
/* assume 0 <= z < 256 */
/* assume 0 <= k < 512 */
int z[], n;
{struct videoconfig vc;
int maxj,maxi,it,ib,jl,jr,ip,jp,k;
float maxx,maxy,scalex,scaley,x,y;
_setvideomode(_HRESBW);
_getvideoconfig (&vc);
maxj=vc.numxpixels;
maxi=vc.numypixels;
it=maxi/10;
ib=maxi*9/10;
jl=maxj/10;
jr=maxj*9/10;
maxx=525.0;
maxy=275.0;
scalex=(float)(jr-jl)/maxx;
scaley=(float)(it-ib)/maxy;
_moveto((int)(scalex*0.0)+jl,(int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*maxy)+ib);
}

```

```

    _lineto((int)(scalex*0.0)+jl, (int)(scaley*maxy)+ib);
    _lineto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
    for (x=0.0; x < maxx; x += 50.0)
    {
        _moveto((int)(scalex*x)+jl,(int)(scaley*(-8.0))+ib);
        _lineto((int)(scalex*x)+jl,(int)(scaley*0.0)+ib);
        _moveto((int)(scalex*x)+jl,(int)(scaley*maxy)+ib);
        _lineto((int)(scalex*x)+jl,(int)(scaley*(maxy+8.0))+ib);
    }
    for (y=0.0; y < maxy; y += 50.0)
    {
        _moveto((int)(scalex*0.0)+jl, (int)(scaley*y)+ib);
        _lineto((int)(scalex*(-10.0))+jl, (int)(scaley*y)+ib);
        _moveto((int)(scalex*maxx)+jl, (int)(scaley*y)+ib);
        _lineto((int)(scalex*(maxx+10.0))+jl,(int)(scaley*y)+ib);
    }
    for (k=0; k < n; k++)
    {
        ip=ib+(int)(scaley*((float)z[k]));
        jp=jl+(int)(scalex*((float)k));
        _setpixel(jp,ip);
    }
    printf("PRESS 'ENTER'\n");
    while (wait_for_key() == 0);
    _setvideomode(_DEFAULTMODE);
}
/*-----*/

void diagnose (str)
char str[];
{
    printf("%s\n",str);
    printf(" %f %d %f %d mid\n",row_mid,i_mid,col_mid,j_mid);
    printf(" %f %d %f %d top\n",row_top,i_top,col_top,j_top);
    printf(" %f %d %f %d bot\n",row_bot,i_bot,col_bot,j_bot);
    printf(" %f %d %f %d left\n",row_left,i_left,col_left,j_left);
    printf(" %f %d %f %d rite\n",row_rite,i_rite,col_rite,j_rite);
    printf(" %f = width, %f = height\n",width,height);
    printf(" %f = horiz, %f = vert cal.\n",horiz_cal,vert_cal);
    printf(" %d = base row\n",i_base);
    printf(" %d %d = cursor position\n",cursor_row,cursor_col);
    printf("\n HIT 'ENTER'\n");
    wait_for_key();
}
/*-----*/

void set_base ()
/* allow the operator to locate the top of the drop base */
{
    unsigned key;
    int count;
    printf("\n USE THE CURSOR KEYS TO SET THE BASE LINE\n");
    printf(" JUST ABOVE THE BASE OF THE DROP;\n");
    printf(" HIT THE 'ENTER' KEY WHEN FINISHED.\n");
    IS_CURSOR(1);
}

```



```

count=0;
while (count == 0)
{IS_SET_CURSOR_POSITION(i_base,1);
key=wait_for_key();
switch(key)
{case 7181:
IS_CURSOR(0);
count=1;
break;
case 18432:
i_base=max(i_base-1,1);
break;
case 20480:
i_base=min(i_base+1,479);
break;
}
}
}
/*-----*/
void focus ()
/* Repeatedly, grab a frame and graph the cursor row */
{struct videoconfig vc;
int maxj,maxi,it,ib,jl,jr,ip,jp,k;
float maxx,maxy,scalex,scaley,x,y;
_setvideomode(_HRESBW);
_getvideoconfig (&vc);
maxj=vc.numxpixels;
maxi=vc.numypixels;
maxx=525.0;
maxy=275.0;
jl=maxj/10;
jr=maxj*9/10;
scalex=(float)(jr-jl)/maxx;
while (kbhit() == 0)
{IS_ACQUIRE(0,1); /* acquire 1 frame in buffer 0 */
get_row(cursor_row,iaa);
get_col(cursor_col,iab);
it=maxi*5/100;
ib=maxi*45/100;
scaley=(float)(it-ib)/maxy;
_clearscreen(_GCLEARSCREEN);
printf(" HIT 'ENTER' TO STOP\n");
_moveto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*maxy)+ib);
_lineto((int)(scalex*0.0)+jl, (int)(scaley*maxy)+ib);
_lineto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
for (x=0.0; x < maxx; x += 50.0)

```

```

    {_moveto((int)(scalex*x)+jl,(int)(scaley*(-8.0))+ib);
      _lineto((int)(scalex*x)+jl,(int)(scaley*0.0)+ib);
      _moveto((int)(scalex*x)+jl,(int)(scaley*maxy)+ib);
      _lineto((int)(scalex*x)+jl,(int)(scaley*(maxy+8.0))+ib);
    }
  for (y=0.0; y < maxy; y += 50.0)
    {_moveto((int)(scalex*0.0)+jl, (int)(scaley*y)+ib);
      _lineto((int)(scalex*(-10.0))+jl, (int)(scaley*y)+ib);
      _moveto((int)(scalex*maxx)+jl, (int)(scaley*y)+ib);
      _lineto((int)(scalex*(maxx+10.0))+jl,(int)(scaley*y)+ib);
    }
  for (k=0; k < 512; k++)
    {ip=ib+(int)(scaley*((float)iaa[k]));
      jp=jl+(int)(scalex*((float)k));
      _setpixel(jp,ip);
    }
  it=maxi*55/100;
  ib=maxi*95/100;
  scaley=(float)(it-ib)/maxy;
  _moveto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
  _lineto((int)(scalex*maxx)+jl,(int)(scaley*0.0)+ib);
  _lineto((int)(scalex*maxx)+jl,(int)(scaley*maxy)+ib);
  _lineto((int)(scalex*0.0)+jl, (int)(scaley*maxy)+ib);
  _lineto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
  for (x=0.0; x < maxx; x += 50.0)
    {_moveto((int)(scalex*x)+jl,(int)(scaley*(-8.0))+ib);
      _lineto((int)(scalex*x)+jl,(int)(scaley*0.0)+ib);
      _moveto((int)(scalex*x)+jl,(int)(scaley*maxy)+ib);
      _lineto((int)(scalex*x)+jl,(int)(scaley*(maxy+8.0))+ib);
    }
  for (y=0.0; y < maxy; y += 50.0)
    {_moveto((int)(scalex*0.0)+jl, (int)(scaley*y)+ib);
      _lineto((int)(scalex*(-10.0))+jl, (int)(scaley*y)+ib);
      _moveto((int)(scalex*maxx)+jl, (int)(scaley*y)+ib);
      _lineto((int)(scalex*(maxx+10.0))+jl,(int)(scaley*y)+ib);
    }
  for (k=0; k < 480; k++)
    {ip=ib+(int)(scaley*((float)iab[k]));
      jp=jl+(int)(scalex*((float)k));
      _setpixel(jp,ip);
    }
}
_setvideomode(_DEFAULTMODE);
wait_for_key();
}
/*-----*/
void graph2 (kode)
/* graph integer arrays iaa & iab */

```

```

int kode;
{struct videoconfig vc;
int maxj,maxi,it,ib,jl,jr,ip,jp,k;
float maxx,maxy,scalex,scaley,x,y;
_setvideomode(_HRESBW);
_getvideoconfig (&vc);
maxj=vc.numxpixels;
maxi=vc.numypixels;
maxx=525.0;
maxy=275.0;
jl=maxj/10;
jr=maxj*9/10;
scalex=(float)(jr-jl)/maxx;
it=maxi*5/100;
ib=maxi*45/100;
scaley=(float)(it-ib)/maxy;
_moveto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*maxy)+ib);
_lineto((int)(scalex*0.0)+jl, (int)(scaley*maxy)+ib);
_lineto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
for (x=0.0; x < maxx; x += 50.0)
{ _moveto((int)(scalex*x)+jl,(int)(scaley*(-8.0))+ib);
  _lineto((int)(scalex*x)+jl,(int)(scaley*0.0)+ib);
  _moveto((int)(scalex*x)+jl,(int)(scaley*maxy)+ib);
  _lineto((int)(scalex*x)+jl,(int)(scaley*(maxy+8.0))+ib);
}
for (y=0.0; y < maxy; y += 50.0)
{ _moveto((int)(scalex*0.0)+jl, (int)(scaley*y)+ib);
  _lineto((int)(scalex*(-10.0))+jl, (int)(scaley*y)+ib);
  _moveto((int)(scalex*maxx)+jl, (int)(scaley*y)+ib);
  _lineto((int)(scalex*(maxx+10.0))+jl,(int)(scaley*y)+ib);
}
for (k=0; k < 512; k++)
{ip=ib+(int)(scaley*((float)iaa[k]));
jp=jl+(int)(scalex*((float)k));
_setpixel(jp,ip);
}
it=maxi*55/100;
ib=maxi*95/100;
scaley=(float)(it-ib)/maxy;
_moveto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*0.0)+ib);
_lineto((int)(scalex*maxx)+jl,(int)(scaley*maxy)+ib);
_lineto((int)(scalex*0.0)+jl, (int)(scaley*maxy)+ib);
_lineto((int)(scalex*0.0)+jl, (int)(scaley*0.0)+ib);
for (x=0.0; x < maxx; x += 50.0)
{ _moveto((int)(scalex*x)+jl,(int)(scaley*(-8.0))+ib);

```

```

    _lineto((int)(scalex*x)+jl,(int)(scaley*0.0)+ib);
    _moveto((int)(scalex*x)+jl,(int)(scaley*maxy)+ib);
    _lineto((int)(scalex*x)+jl,(int)(scaley*(maxy+8.0))+ib);
}
for (y=0.0; y < maxy; y += 50.0)
{
    _moveto((int)(scalex*0.0)+jl, (int)(scaley*y)+ib);
    _lineto((int)(scalex*(-10.0))+jl, (int)(scaley*y)+ib);
    _moveto((int)(scalex*maxx)+jl, (int)(scaley*y)+ib);
    _lineto((int)(scalex*(maxx+10.0))+jl,(int)(scaley*y)+ib);
}
for (k=0; k < 480; k++)
{
    ip=ib+(int)(scaley*((float)iab[k]));
    jp=jl+(int)(scalex*((float)k));
    _setpixel(jp,ip);
}
if (kode == 1)
{
    printf(" HIT ENTER");
    wait_for_key();
    _setvideomode(_DEFAULTMODE);
}
}
/*-----*/

```

APPENDIX B

Program Listing for MEORS1.C

```
#define GDL 1
#define B7 2
#define WATKINS 3
#include <system.def>

#if SYSTEM == WATKINS
#include <c:\c\include\stdio.h>
#include <c:\c\include\conio.h>
#include <c:\c\include\bios.h>
#include <c:\c\include\math.h>
#include "c:\vision\isdefs.c"
#endif

#if SYSTEM == B7
#include <c:\qc2\include\stdio.h>
#include <c:\qc2\include\conio.h>
#include <c:\qc2\include\math.h>
#include <c:\qc2\include\bios.h>
#include <c:\qc2\source\isdefs.c>
#endif

#if SYSTEM == GDL
#include <c:\qc25\include\stdio.h>
#include <c:\qc25\include\conio.h>
#include <c:\qc25\include\math.h>
#include <c:\qc25\include\bios.h>
#include <c:\librarys\isdefs.c>
#endif

/*-----*/
#if SYSTEM == GDL
void put_col (col,y)
/* put a column of pixels from int array y into image buffer 0. */
int col; /* column number desired */
int y[]; /* source array */
{int row;
 if ((col<0) || (col>511)) printf(" %d = column in put_col\n",col);
 for (row=0; row<512; row++) IS_PUT_PIXEL(0,row,col,1,&y[row]);
}
#endif
/*-----*/
#if SYSTEM == GDL
void get_col (col,y)
/* get a column of pixels from image buffer 0 into int array y. */
```

```

int col; /* column number desired */
int y[]; /* destination array */
{int row;
  if ((col < 0) || (col > 511)) printf(" %d = column in get_col\n", col);
  for (row = 0; row < 512; row++) IS_GET_PIXEL(0, row, col, 1, &y[row]);
}
#endif
/*-----*/
#if SYSTEM == B7
void put_col (col, y)
/* put a column of pixels from int array y into image buffer 0. */
int col; /* column number desired */
int y[]; /* source array */
{if ((col < 0) || (col > 511)) printf(" %d = column in put_col\n", col);
  IS_SET_ACTIVE_REGION(0, col, 512, 1);
  IS_PUT_REGION(0, y);
}
#endif
/*-----*/
#if SYSTEM == B7
void get_col (col, y)
/* get a column of pixels from image buffer 0 into int array y. */
int col; /* column number desired */
int y[]; /* destination array */
{if ((col < 0) || (col > 511)) printf(" %d = column in get_col\n", col);
  IS_SET_ACTIVE_REGION(0, col, 512, 1);
  IS_GET_REGION(0, y);
}
#endif
/*-----*/
void put_row (row, y)
/* put a row of pixels from int array y into image buffer 0. */
int row; /* row number desired */
int y[]; /* source array */
{if ((row < 0) || (row > 479)) printf(" %d = row in put_row\n", row);
  IS_SET_ACTIVE_REGION(row, 0, 1, 512);
  IS_PUT_REGION(0, y);
}
/*-----*/
void get_row (row, y)
/* get a row of pixels from image buffer 0 into int array y. */
int row; /* row number desired */
int y[]; /* destination array */
{if ((row < 0) || (row > 479)) printf(" %d = row in get_row\n", row);
  IS_SET_ACTIVE_REGION(row, 0, 1, 512);
  IS_GET_REGION(0, y);
}
/*-----*/

```

```

int wait_for_key ()
/* wait until a key is struck, return the integer value */
/* include conio.h */
{while (kbhit() == 0);
return (get_key());
}
/*-----*/

get_key()
{union REGS r;
r.h.ah = 0;
return int86(0x16, &r, &r);
}
/*-----*/

void save ()
/* save the image from buffer 0 to a disk file */
{char filename[80];
printf("Enter name of file (default extension = .img):\n");
scanf("%s",filename);
defext(filename,80,"img");
IS_SET_ACTIVE_REGION(0,0,512,512);
IS_SAVE(0,0,1,0,filename);
printf("IMAGE STORED TO DISK FILE  %s\n",filename);
}
/*-----*/

void restore ()
/* restore an image from a disk file to buffer 0 */
{char filename[80];
printf("Enter name of file (default extension = .img):\n");
scanf("%s",filename);
defext(filename,80,"img");
IS_SET_ACTIVE_REGION(0,0,512,512);
IS_RESTORE(0,0,0,filename);
printf("IMAGE RESTORED FROM DISK FILE  %s\n",filename);
}
/*-----*/

float avgv (y,n)
/* return the average value (float) of the n elements in the integer
   array y. */
int y[]; /* array of values */
int n; /* number of elements in array y */
{int i;
long sum;
if (n < 0) return 0.0;
sum=0;
for (i=0; i<n; i++) sum+=y[i];
return (((float)sum)/((float)n));
}
/*-----*/

```

```

float avgi (y,n)
/* treat the integer array y as a probability density function
   (or a histogram or a distribution) of values of the array
   index i, and calculate the (float) average value of i for
   this distribution. Return a negative value if an error is
   detected. */
int y[]; /* array of pdf values, each assumed >= 0 */
int n; /* number of values in array y */
{int i;
 long sumy,sumyi;
 if (n <= 0) return (-1.0);
 sumy=0;
 sumyi=0;
 for (i=0; i<n; i++)
 {sumy +=y[i];
  sumyi +=((long)y[i])*i;
 }
 if (sumy <= 0) return (-2.0);
 return (((float)sumyi)/((float)sumy));
}
/*-----*/

void derivative (x,y,n)
/* replace the values in y with the derivative of the values
   of x with respect to array index. x and y may be the same. */
int x[], y[];
int n; /* number of values in each array */
{int i;
 for (i=0; i<(n-1); i++) y[i]=x[i+1]-x[i];
}
/*-----*/

void datim (str)
char str[21];
{long lt, *plt;
 int year,month,day,hour,minute,second;
 int quad,mo,days[12]={31,28,31,30,31,30,31,31,30,31,30,31};
 char cn[10]={'0','1','2','3','4','5','6','7','8','9'};
 plt=&lt;
 time(plt);
/* lt = seconds since beginning of 1970. */
second=lt%60;
lt/=60;
/* lt = minutes since beginning of 1970. */
minute=lt%60;
lt/=60;
lt+=16; /* correction necessary for unknown reasons */
/* lt = hours since beginning of 1970. */
hour=lt%24;
lt/=24;
}

```



```

/* lt = days since beginning of 1970. */
/* 1 quad = 1461 days = 4 years including 1 leap year. */
quad=lt/1461;
lt%=1461;
/* lt = days remaining after removing all whole quads. */
year=0;
if (lt >= 365)
    {year += 1;
    lt-=365;
    }
if (lt >= 365)
    {year += 1;
    lt-=365;
    }
if (lt >= 366)
    {year += 1;
    lt-=366;
    }
/* year = years remaining after removing all whole quads. */
/* adjust days in February for leap year: */
if (year == 2)
    days[1]=29;
else
    days[1]=28;
/* find month: */
for (mo=0; mo<12; mo++)
    {if (lt >= days[mo])
        {lt-=days[mo];
        }
    else
        {month=mo+1;
        mo=13;
        }
    }
day=lt;
year += quad*4 + 70;
str[0]=' ';
str[1]=cn[year/10];
str[2]=cn[year%10];
str[3]='/';
str[4]=cn[month/10];
str[5]=cn[month%10];
str[6]='/';
str[7]=cn[day/10];
str[8]=cn[day%10];
str[9]=' ';
str[10]=cn[hour/10];
str[11]=cn[hour%10];

```

```

str[12]=': ';
str[13]=cn[minute/10];
str[14]=cn[minute%10];
str[15]=': ';
str[16]=cn[second/10];
str[17]=cn[second%10];
str[18]=' ';
str[19]=' ';
str[20]='\0';
}
/*-----*/
int med001 (src,dst,npts>window_size)
/* apply a median filter to the data in array src,
   and put the filtered data in array dst. */
int src[],dst[];
int npts; /* size of arrays src & dst */
int window_size; /* size of moving window; should be odd */
#define MAX_WIN 99
/* MAX_WIN is the largest acceptable window_size. */
{int med001a();
 int med001b();
 int med001c();
 int i,n,nhi,nlo,n1,n2,nk,j,k[MAX_WIN];
 /* k[] = list of indexes of src array elements in window.
    The list is ordered so that src[k[n]] >= src[k[m]] if n > m. */
 if (window_size > MAX_WIN) return (-1);
 if (window_size < 1) return (-2);
 if (window_size >= npts) return (-3);
 for (i=0; i<npts; i++) dst[i]=src[i];
 if (window_size < 3) return (0);
 n=(window_size-1)/2;
 n=n*2+1; /* n is size of window that we will use. */
 nhi=n/2;
 nlo=nhi+1;
 n1=n/2;
 n2=npts-n/2-1;
 nk=0; /* nk = number of elements in current window */
 nk=med001a(0,k,nk,src); /* start with one element in window */
 for (i=1; i<(npts-1); i++)
 {if (i<=n1) /* at start, add 2 elements to window */
 {j=2*i-1;
 nk=med001a(j,k,nk,src);
 j=2*i;
 nk=med001a(j,k,nk,src);
 }
 if ((i>n1)&&(i<=n2)) /* in middle, delete 1 and add 1 */
 {j=i-nlo;
 nk=med001b(j,k,nk);
 }
 }
}

```

```

    j=i+nhi;
    nk=med001a(j,k,nk,src);
}
if (i>n2) /* at end, delete 2 elements from window */
{
    j=npts-2*(npts-i);
    nk=med001b(j,k,nk);
    j=npts-2*(npts-i)-1;
    nk=med001b(j,k,nk);
}
dst[i]=med001c(k,nk,src);
}
return (0);
}
int med001a (j,k,nk,src)
/* Insert an index into the list of indexes of elements in the
window. Put it in the right place to maintain an ordered list. */
int j,k[],nk,src[];
{int srcj,l,m,n;
if (nk<=0)
{
    k[0]=j;
    return (1);
}
srcj=src[j];
for (l=0; l<nk; l++)
{
    m=k[l];
    if (src[m]>srcj)
    {
        for (n=nk; n>l; n--) k[n]=k[n-1];
        k[l]=j;
        return (nk+1);
    }
}
k[nk]=j;
return (nk+1);
}
int med001b (j,k,nk)
/* Remove an index from the list of indexes of elements in the
window. */
int j,k[],nk;
{int l,n;
if (nk<=0) return (0);
for (l=0; l<nk; l++)
{
    if (k[l]==j)
    {
        nk-=1;
        for (n=l; n<nk; n++) k[n]=k[n+1];
        return (nk);
    }
}
return (nk);
}

```

```

}
int med001c (k,nk,src)
/* Return the value of the source array element pointed to by the
   index in the middle of the list, which is the median value of
   all the values in the window. This is the desired filtered
   value for the current destination array index. */
int k[],nk,src[];
{int l;
 l=k[nk/2];
 return (src[l]);
}
#undef MAX_WIN
/*-----*/
int fit_quadratic (x,y,n,a,b,c,xex,yex)
/* Least squares fit  $y=a+bx+cx^2$ . Return negative integer if
   error occurs. Set values for a, b, c, yex=y value at extremum,
   and xex=x value at extremum. */
float x[], y[], *a, *b, *c, *xex, *yex;
int n; /* number of data points in arrays x & y */
{int i;
 double sumx0, sumx1, sumx2, sumx3, sumx4, sumx0y, sumx1y, sumx2y;
 double xx, yy, det, aaa, bbb, ccc, dd, aa, bb, cc, xext, yext;
 sumx0=(double)n;
 sumx1=0.0;
 sumx2=0.0;
 sumx3=0.0;
 sumx4=0.0;
 sumx0y=0.0;
 sumx1y=0.0;
 sumx2y=0.0;
 for (i=0; i<n; i++)
 {xx=(double)x[i];
 yy=(double)y[i];
 sumx1 += xx;
 sumx2 += xx*xx;
 sumx3 += xx*xx*xx;
 sumx4 += xx*xx*xx*xx;
 sumx0y += yy;
 sumx1y += xx*yy;
 sumx2y += xx*xx*yy;
 }
 /* a*sumx0 + b*sumx1 + c*sumx2 = sumx0y */
 /* a*sumx1 + b*sumx2 + c*sumx3 = sumx1y */
 /* a*sumx2 + b*sumx3 + c*sumx4 = sumx2y */
 det=sumx0*(sumx2*sumx4-sumx3*sumx3)
 -sumx1*(sumx1*sumx4-sumx3*sumx2)
 +sumx2*(sumx1*sumx3-sumx2*sumx2);
 aaa=sumx0y*(sumx2*sumx4-sumx3*sumx3)

```

```

    -sumx1y*(sumx1*sumx4-sumx3*sumx2)
    +sumx2y*(sumx1*sumx3-sumx2*sumx2);
bbb=sumx0*(sumx1y*sumx4-sumx2y*sumx3)
    -sumx1*(sumx0y*sumx4-sumx2y*sumx2)
    +sumx2*(sumx0y*sumx3-sumx1y*sumx2);
ccc=sumx0*(sumx2*sumx2y-sumx3*sumx1y)
    -sumx1*(sumx1*sumx2y-sumx3*sumx0y)
    +sumx2*(sumx1*sumx1y-sumx2*sumx0y);
aa=fabs(aaa);
bb=fabs(bbb);
cc=fabs(ccc);
dd=fabs(det);
if ((dd < aa/1.0E10) || (dd < bb/1.0E10) || (dd < cc/1.0E10)) return (-1);
aaa/=det;
bbb/=det;
ccc/=det;
*a=(float)aaa;
*b=(float)bbb;
*c=(float)ccc;
if (fabs(bbb) < fabs(ccc)*1.0e10)
{
    xext=-bbb/ccc/2.0;
    yext=aaa+xext*(bbb+xext*ccc);
    *xex=(float)xext;
    *yex=(float)yext;
    return (0);
}
else return (-2);
}
/*-----*/
int in_string (str,n)
/* get characters from the keyboard into string str;
   return the number of characters. */
char str[]; /* string to be filled with input characters */
int n; /* dimension of array str[] */
{int i;
char c;
for (i=0; i<n; i++)
{c=getch();
if (c=='\r')
{str[i]='\0';
putchar('\n');
return i;
}
str[i]=c;
putchar(c);
}
return n;
}

```

```

/*-----*/
int defext (filename,nbytefil,ext)
/* if filename has no extension (no '.'), add the extension ext */
char filename[], ext[];
int nbytefil; /* number of bytes in char array filename */
{int ncharext, i, j;
 ncharext=strlen(ext);
 for (i=0; i<nbytefil; i++)
 {if (filename[i] == '.') return 0;
  if (filename[i] == '\0')
  {if ((i+1+ncharext) >= nbytefil) return -1;
   filename[i]='.';
   for (j=0; j<ncharext; j++) filename[i+1+j]=ext[j];
   filename[i+1+ncharext]='\0';
   return 1;
  }
 }
 return -2;
}
/*-----*/

```

DATE
FILMED

12/17/93

END