AFOSR-TR- 88-1320

Whole Field Measurements of Vorticity in Turbulent and Unsteady Flows

by

R. E. Falco, PI
C. P. Gendrich
C. C. Chu

Final Report

Prepared from work done under

AFOSR Contract 86-0242

ORD No. 39880

Report TSL-88-5

Turbulence Structure Laboratory

Department of Mechanical Engineering

Michigan State University

East Lansing, MI 48824

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

OR are
Blank pgs
that have
Been Removed

BEST AVAILABLE COPY

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: Unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 88-1320
6a. NAME OF PERFORMING ORGANIZATION: Michigan State University; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: AFOSR
6c. ADDRESS (City, State, and ZIP Code): East Lansing, MI 48824	7b. ADDRESS (City, State, and ZIP Code): AFOSR/NA, Bolling Air Force Base, Washington, DC
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: AFOSR/NA; 8b. OFFICE SYMBOL (If applicable): NA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-86-0242
8c. ADDRESS (City, State, and ZIP Code): AFOSR/NA, Bolling AFB, DC 20332	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2307; TASK NO. A2; WORK UNIT ACCESSION NO 102

11 TITLE (Include Security Classification)
Whole Field Measurements of Vorticity in Turbulent and Unsteady Flows

12 PERSONAL AUTHOR(S)
R. E. Falco (PI), C. P. Gendrich, and C. C. Chu

13a TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 8/1/86 TO 7/31/88	1988 October 11	102

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Turbulence, Boundary Layer

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Measurements of cross-stream and streamwise vorticity have been made over important regions of an experimental simulation of the bursting process of turbulent boundary layers. Vorticity measurements have also been made of the starting vortex of an airfoil, in a vortex ring, and in a Stokes' layer. Using a new technique developed under this contract, these measurements have been made at approximately fifty simultaneous positions over the flow structures of interest. The technique can accurately measure vorticity and strain rate, as well as instantaneous Reynolds stress and velocities over a field. It requires only a clock and a ruler for calibration. Furthermore, it directly measures these quantities, avoiding indirect interpretations. It employs long time persistence of irradiation of a photochemical to mark fluid particles. The technique has been named LIPA, an acronym for Laser Induced Photochemical Anemometry. Its accuracy has been measured by comparing it to an exact solution of the Navier-Stokes equations. This indicated an absolute accuracy of cross-
(Continued on following page)

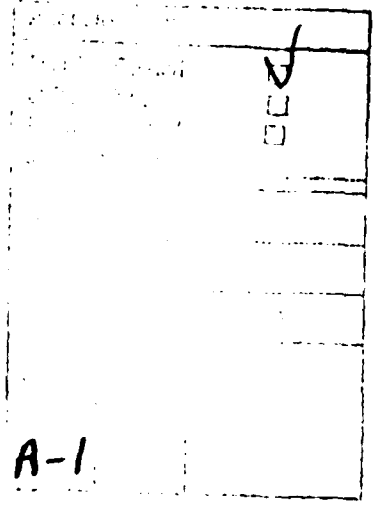
20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Dr. Jim McMichael	22b. TELEPHONE (Include Area Code): (202) 767-4935; 22c. OFFICE SYMBOL: AFOSR/NA

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT (Continued)

stream vorticity of ±1/sec. Data reduction procedures have been developed which utilize high resolution digitization and image processing. Algorithms have been written and tested which allow almost complete automation of the data reduction procedure.



INTRODUCTION

There is a need for measurements of flow quantities such as vorticity and strain rate as well as instantaneous Reynolds stress over and area in turbulent and unsteady flows. Examples include all aspects turbulence structure and entrainment studies, external and internal aerodynamics, studies of turbulence control, of mixing and combustion. In many cases there is a need to obtain more detailed information about these flows, but our understanding of a large number of simpler flows would also be increased. Existing probe techniques are at best able to provide single point estimates of gradient information. Of particular importance is the need to develop instrumentation to extend these capabilities to high speed flow studies. Another need is for instrumentation techniques that can enable quantitative Lagrangian information to be obtained. Evidence of these needs is the very high level of activity both in the US and in Europe aimed at developing quantitative field measurement techniques. It is well known that measurements of vorticity using a hot-wire probe at a single location are very difficult, but significantly less difficult than use of LDA to perform the same task. Thus, the goal of obtaining a large number of simultaneous vorticity measurements over an area (that may include an evolving coherent motion, for example) requires a different approach.

The technological transfer between physicists, plasma physicists, chemists, computer scientists and fluid dynamicists, has been the basis for the application of new techniques to measure these quantities at single or multiple points. Several new techniques for turbulence measurements in fluids have thus been adapted from physics and chemistry. They are based on atomic or molecular properties of the medium and changes in these properties. The techniques have rapidly become useful tools because of advances in laser and computer technology. Typical examples of the diagnostics are: Laser Induced Fluorescence (LIF), Forced Rayleigh Scattering (FRS). Anti-Stokes Raman Diffusion (ASRD), Fourier

Densitometry, and Laser Speckle. These advanced techniques can lead to information of a new kind, or may enable measurements to be performed in currently difficult flows. To this list we are now able to add Laser Induced Photochemical Anemometry (hereafter call LIPA). It has specific advantages over many of the others, which are discussed below.

Although many of the new optical techniques are essentially single point measurement techniques, LIPA is a field measurement technique, as is PV (Particle Velocimetry, which includes LPV, Laser Particle Velocimetry). Of those above which are field measurement techniques, all require optical data acquisition, and thus they have in common several advantages and disadvantages. Advantages include either no intrusion or very little intrusion on the flow to be measured, and very fast response. Many of these new techniques can be used in compressible, supersonic or hypersonic flows, where the conventional techniques (hot-wires, laser doppler anemometry) are difficult to use. LIPA can also be used in high speed flows, although these applications were not part of the current work. Common disadvantages include the need for optical interrogation of the results making conversion to digital form expensive.

Most of these new field techniques suffer from limitations of interpretation (for example fourier densitometry provides wave vectors), or from limitations to low speed liquid flows (particle velocimetry), or from inherent inaccuracies (for example the spatial ambiguities of speckle), or from inaccurate data reduction potential (holographic particle velocimetry). LIPA can surmount almost all of these difficulties. The outline of this report will first include a discussion of the technique and its variations, then a discussion of results from the present investigation, and finally an overview of new applications and the specific advantages and disadvantages of using LIPA to study them.

RESULTS

A) The LIPA technique

LIPA depends essentially upon marking specific fluid particles so that they can be tracked as they move in a flow. We must record the tagged particles' motion. We have restricted our data acquisition to the visible range of wavelengths, but the technique is not, in principle, confined to this range. In the visible range we have two types of photochemical tagging available, either color change or emitted light. Color change upon irradiation is called photochromism, while light emission is broadly called florescence. Both types have been used in experiments reported herein, and thus a review of the pros and cons of each are now presented.

At this stage in the development of LIPA, a limiting factor is the number of chemicals that are available to be irradiated with lifetimes long enough to allow the flow to move appreciably. This limitation has made the lifetime of the chemical govern the experiments that have been performed. The chemicals have also often been the determining factor in the choice of fluid that has been used for the experiments. In general photochromic chemicals have a lifetime of 1ms to several seconds. This lifetime is long enough so that they are suitable for almost all experiments, from very low speed to very high speed. However, these chemicals can only be dissolved in organic liquids. Thus, the range of fluid dynamic experiments with photochromic chemicals is limited to those in liquids. On the other hand, photoluminescent chemicals tend to have very short lifetimes. Typical florescent lifetimes are the order of nanoseconds, far too short for the radiation to last long enough for all but hypersonic flows to appreciably move the tagged fluid particles. Thus, special photoluminescent chemicals are needed which have longer lifetimes. These tend to be specially constructed molecules which have a 'shield' of other atoms surrounding them that prevent collisional quenching, but are transparent to the incident and emitted radiation. One of

these is the proprietary product called Flowlite I from Flowmod Corp. This chemical can be dissolved in water and has a 1.5ms lifetime. We have used it in preliminary experiments in water and in a water droplet aerosol. Another chemical we have used is $K_2Pt_2(POP)_4$. This was synthesized at Michigan State University by professor D. Nocera It has a 10μs lifetime, and has been used successfully as a water droplet aerosol in N_2. We envision a rapid growth of the number of chemicals synthesized for use in LIPA as the value of the technique becomes apparent.

For the experiments described herein, tagging has been performed in a plane. This is not an inherent constraint; three-dimensional tagging and stereo recording is certainly possible, and is seen as an evolutionary step in the use of LIPA. Tagging in a plane, combined with stereo viewing will enable all three components of velocity in the neighborhood of a chosen plane to be obtained. Furthermore, creation of two closely spaced grids, combined with stereo data acquisition will enable all three components of the vorticity vector in the neighborhood of the bisecting plane to be obtained.

In the current version of the technique, we have chosen to develop passive beam splitting and steering devices, which divide the laser beam into 'n' beams (we have used n = 7, and 10) and cause these beams to cross within the fluid. It has the advantages that no incident light is lost, and that the line widths and spacings can be specified by the facet design. The devices are similar in appearance to an oversized diffraction grating (see Appendix B, Fig. 1 and 2), but it gives specular reflection. The subsequent excitation of the chemical along the multiple intersecting laser lines creates a grid of 'n' lines of marked fluid. At each of the intersections of the laser gridlines a 'fluid particle' is unambiguously tagged. This results in n^2 simultaneous measurements over the area of the grid. The distortion of the grid in a known time provides us with all of the kinematic information obtainable in a fluid in the plane of the grid. The

resolution of this information is equal to the applied grid mesh. For flows where differential resolution is natural, the grid can be set to match the needs of the flow.

Details of the use of the technique have been published (see Falco and Chu (1987); Falco, Chu, Hetherington and Gendrich (1988)) and are included in appendix B and C. Further description is given in Chu (1988) and Falco and Chu (1988). It is important to emphasize that the technique used requires only a known length and a known time between exciting the chemical with a laser and imaging the marked fluid particles. Precision placement of measurement points is another advantage of the technique. Because the laser excitation is in a plane, our information will be at selected points in (or very close to) selected planes in the flow. This ability to choose to have the measurement points near a center line or at $y^+ = 5$, for example, is extremely helpful.

B) Automated data reduction

The success of the development of LIPA into a tool that can be widely used for detailed fluid dynamic measurements will most probably depend upon the ease of data reduction. Thus, we have put considerable effort into automating the data reduction. Key to our success has been the use of the lines of radiation from the chemicals in the grids to provide redundant information for the determination of the points of intersection of the laser lines. This has made imperfections in the film or optics, film grain, dirt acquired in developing, etc. unimportant. The fact that two independent pieces of film are used further decouples the problems with parasitic images, and as a result we have a robust data reduction technique.

The algorithms used are included in the Appendix A. They are written in FORTRAN and have been run on a Silicon Graphics IRIS 3120, under Unix System V. Briefly, it allows the velocity, gradients, vorticity and instantaneous Reynolds stress to be calculated from four randomly spaced points. The field may have 'p'

groups of points, where 'p' is only limited by available computer memory. The algorithm will calculate the vorticity in contiguous polygons, if points are within distances that are less than a specified fluid mechanically determined limit, such as a Kolmogoroff scale. It requires that each of four points be identified in two photographs, separated by an arbitrary time interval.

C) Determination of the accuracy of the technique

We have calibrated the technique by making measurements in a flow in which we have an exact solution (a Stokes' layer). Results are presented in the papers in the Appendix B and C (also see Falco and Chu 1988). Essentially, we can obtain cross-stream vorticity to ± 1 sec^{-1} accuracy. This compared very well with estimates made using classical error analysis techniques. Since we can very accurately measure both a clock and a ruler, we should be able to maintain this accuracy over a wide range of flow speeds.

D) Cross-stream vorticity measurements:

Cross-stream vorticity measurements have been made in the Stokes' layer for calibration as mentioned above, in a vortex ring, and in the starting vortex of an airfoil. The Stokes' layer measurements are discussed in Appendix B and C.

The measurements in a vortex ring Falco and Chu (1987), show for the first time the distribution of vorticity in a single ring and its changes as we move circumferentially. Previous measurements with LDA have required averaging over many rings, and suggested greater symmetry. These changes in a real ring that was very carefully produced and very stable, indicate the insensitivity of the stability of the ring to the detailed shape of the vorticity distribution or of its circumferential symmetry. The other point of interest is that the circulation for our low Reynolds number ring is quite close to that found for a Hill's spherical vortex.

The measurements of the starting vortex formed when an airfoil is

impulsively started have also shown significant surprises. This study, described in Appendix C, indicated that only about 1/3 of the vorticity was in the starting vortex, which means that the remaining vorticity is in a shear layer between the wing trailing edge and the vortex. As a result, the Kutta condition can not be established. This has interesting implications for non-steady airfoil work and for calculations that depend on the Kutta condition being established. Since it has not been possible to make measurements of this type before, unsteady phenomena which require vorticity measurements over an area have recieved no experimental attention.

E) Streamwise vorticity measurements of a simulation of the production process in turbulent boundary layers

Streamwise vorticity measurements have been made in streamwise vortices created above a wall, similar to those found in turbulent boundary layers. These results have been reported by Chu (1988) and by Falco and Chu (1988). Briefly, we have observed this sequence in the turbulent boundary layer, and have subsequently been able to model it, in all its detail, by passing a vortex ring over a laminar shear layer (either a Stoke's or Blasius layer). It is true that the complete event is rarely seen in the turbulent boundary layer. However, a significant part of it was apparent during each of the bursts we have observed (and it must be noted that our techniques don't result in observation of the entire picture at all times). We are presently doing this in the turbulent boundary layer using either multiple dye slits, or the LIPA technique (it will be reported on later), which is allowing whole field real time measurements of vorticity and instantaneous Reynolds stress in the marked fluid.

In experiments designed to examine only the wall region, we observed that the production sequence starts with the formation of a pair of long streaks. Using two dye slits in series, we observed that subsequently a pocket forms

between the streak pair; often after streaks of several hundred x^+ have formed. It appears to us that the dye marks a strong vortex which forms in the pocket. Use of the LIPA technique, in simulations, has clearly showed that the pocket vortex is present and that it is the most intense vortex found in the wall region.

The other aspect that has been quantified, is that the long streaks are observed to undergo an instability that results in the formation of one or more hairpin vortices over each streak. Using LIPA, we have measured the streamwise vorticity in these vortices and found it to be weak with respect to the streamwise vorticity in the pocket vortex. Left to evolve on their own, these hairpin vortices would not contribute much to the production of turbulence.

However, this is not the case. The experiment shows that as the pocket vortex is stretched, these secondary hairpins are convected past it. Because the pocket vortex, which now has a primarily streamwise component, is much stronger, it induces the hairpin vortices around it. Our two-dimensional instantaneous vorticity maps of these features and their interactions gave us a detailed quantitative account of the ensuing interaction. We found that the pocket vortex was more than twice as strong as the hairpins, and that it lifted them completely around it leading to the breakup. Furthermore, practically all of the Reynolds stress associated with the streak...secondary hairpin...pocket vortex interaction is due to the pocket vortex and the motion it induces.

In addition, our observations have indicated that the streaks do not form as a result of a 'pumping' action by long streamwise vortices, and that evidence of vortices associated with the streaks is only found when the secondary hairpins form. LIPA unequivocally shows no concentration of streamwise vorticity during the time when streaks first appear.

DISCUSSION

There are two additional aspects that use of the LIPA technique can uniquely provide for researchers in turbulent and unsteady flows. They are 1) quantitative Lagrangian information and 2) quantification of the kinematics of high speed flows.

Since high speed movies can be taken (synchronized in combination with a high repetition rate Excimer laser) we can gain an understanding of the evolution of the dynamics of unsteady fluctuating flows, i.e. obtain quantitative Lagrangian information. This is of great importance in the study of control of turbulence, for the details of how the control interacts with the coherent structures is the essential design information. Experiments of this type are currently underway at the Turbulence Structure Laboratory to help understand turbulent boundary layer inner-outer interactions, and their role in the production process.

Another range of insights will come from far more detailed measurements of the dynamics of high speed flows. The capability of LIPA to obtain whole field measurements in gas flows (which are seeded with submicron water droplets containing one of the chemicals), makes possible measurements in high speed flows. Most uv lasers pulse in nanoseconds, and the reactions occur in even shorter time scales. Since this is a non-intrusive measurement technique, and since it requires only a clock and a ruler for calibration, it potentially can provide high accuracy results for flows at any speeds currently obtainable in wind tunnels. Recording the signals with image converter cameras appears entirely possible. Thus, potentially, transonic and supersonic flow studies could be performed. An interesting sidelight is that LIPA may prove to be a practical tool for aircraft development. Since many high Reynolds number experiments are now being performed in cryogenic tunnels, and since N_2 essentially prevents quenching, construction of an optical setup that would provide, say 100 to 1000

data points over a body, might prove very cost effect when compared to the tunnel time required for a similar number of LDA measurements.

Finally, LIPA using a gaseous phase chemical in a gas flow (this is conceptually possible, but has not been attempted), has the potential to give whole field measurements in high temperature, combusting or hypersonic flows.

Summarizing, the advantages of the LIPA technique are:

- Instantaneous measurements of velocity gradients, vorticity, strain rate, instantaneous Reynolds stresses and velocities can be made over an area
- Can be setup to give highly accurate space and time resolved measurements over a large range of velocities
- Non-intrusive
- Measurements can be made at chosen locations
- Simple calibration depends only on a clock and a ruler
- Simple data acquisition
- Not limited by interferometric quality access
- Data reduction can be automated
- Can be used in gases (currently in aerosol form) or liquids
- Can be used in non-Newtonian fluids and in two-phase flows
- Three-dimensional measurements are possible
- Standard wavelengths of uv lasers can be used
- Can be combined with flow visualization
- Mixing can be studied by using more than one chemical
- Potentially can be combined with LIF to give concentration information along with the kinematic data.

The chief disadvantage of LIPA is:

- Specific chemicals must be found for flow velocities and fluids of interest.

CONCLUSIONS AND FUTURE WORK

LIPA has been developed and used to measure derivative information at multiple points in complex flows. An algorithm has been written that essentially automates data reduction. The study of a model of the production process has shown that the pocket vortex is the strongest, easily overwhelming the hairpin vortices that develop over the streaks. Our study of the starting vortex indicates that significant vorticity exists in a shear layer between it and the airfoil at low Reynolds numbers.

Quantification of this kind has here-to-fore been impossible. LIPA enables quantitative measurements of velocity, vorticity, strain rate, and instantaneous Reynolds stress over an area in turbulent flows. Only two successive pictures are needed to calculate the distortion of the marked fluid. Calibration requires only a clock and a ruler. It works in reversed flows. The technique has been used in kerosene, water, nitrogen and air (with seeding). As such, it should be able to measure flows into the supersonic range. It can also be extended so as to make three-dimensional measurements of all kinematic quantities over an area. The potential for the use of LIPA in many other flows and particularly at high Reynolds numbers has been discussed.

Future work will include:

a) Measurement of coherent motions; their formation, evolutions and interactions in turbulent water flows
b) Development of technique in gases.
c) Use of the technique in water to develop its three-dimensional capability for both velocity and vorticity fields.
d) Development of simultaneous LIPA and LIF measurements.
e) Use of the technique in high speed flows.

PAPERS resulting from this work to date:

Chu, C. C. 1988 PhD Thesis Department of Mechanical Engineering, Michigan State University.

Falco, R. E. and Chu, C. C. 1987 "Measurement of two -dimensional fluid dynamic quantities using a photochromic grid tracing technique," SPIE Vol 814 pg 706-710.

Falco, R. E. and Chu, C. C. 1988a "A study of turbulence production using a new photochromic visualization technique," Yearly Report AFOSR Contract 87-0047 (also TSL -88-2 Dept. of Mech. Engr. MSU).

Falco, R. E. and Chu, C. C. 1988b "Experimental determination of skin friction modifications due to elastic compliant surfaces using quantitative visual techniques", Report TSL-88-3 Dept of Mech. Engr. MSU.

Falco, R. E. and Chu, C. C., Hetherington, M. H. and Gendrich, C.P. 1988 "The circulation of an airfoil starting vortex obtained from instantaneous vorticity measurements over an area," AIAA 88-3620.

Students trained in the development of this technique during the contract period

1. Mike Hetherington (currently working on his M.S.)
2. Sean Hilbert (currently working on his M.S.)
3. Chuck Gendrich (currently working on his M.S.)
4. Dan Chu, received the Ph.D. 1987, currently associate professor National Taiwan University
5. Dan Hamilton (currently working on his MS)
6. Harry Haufbauer (currently working on his MS)
7. Todd Parr (undergraduate)

APPENDIX A

This appendix contains the command files which run the image processing equipment, converting the visual information into digital form. The program which interrogates the visual information and calculates the fluid kinematic quantities is also included. The programs which are run on the image processor to perform standard operations like averaging and edge enhancement are too lengthy to include here.

The data flow is as follows. Either film or video tape is used to record an experiment. The image processor is required to digitize all of these frames and then individually reduce each image to a set of intersection points. A model command file can be applied to each image to perform this data reduction, but since our image processor does not allow one command file to call another, a large command file must be generated to automatically process many frames without personal intervention.

The first command file (TapeCrunch) is a UNIX shell script which generates this large image processing command file. In this example it is setup to acquire up to 40 sequential frames from a video tape, store them, and then process them using a model command file. The second command file (TapeCrunch.cxm) is suitable for use on the MegaVision 1024XM image processor. Although up to 40 frames may be processed at once, we show how only five are done in the interests of brevity. The third command file (auto.cxm) contains the commands to reduce the digitized images to two sets of intersecting lines.

The program VORTICITY and all of its subroutines comprise the remainder of this appendix. It must be run after the intersections have been obtained. It takes the output of the image processing programs and converts that into velocity, velocity derivative, and vorticity information.

```
#! /bin/sh
#
# Description:
#       This unix shell script generates a large MegaVision command file
#       to crunch many frames from a video tape.  It is necessary to
#       generate one large MegaVision command file since the image
#       processing software doesn't permit nesting from one command
#       to another then back again.
#
# Use of Arguments $0, $1, $2, ...:
#   - The command file ($FILE) created will have the name $0.cxm.
#       NOTE: Any previous contents of $FILE are wiped out by this
#       shell script.
#   - The root filename used to store the video data frames is take from
#       $1.  If $1 is not set (or null), the string 'VidFram' is used.
#   - The root filename used to store the intersection data will be $2 or
#       'Intersect'.
#   -  The model command file to crunch the data will be $3 or 'auto.cxm'.
#
# History:
# <cpg> 6 oct 88 -- 1st version.  Always did it by hand before...
#
FILE=$0.cxm
FRAMES=5
arg=$1 , VROOT=`pwd`/${arg:=VidFram}
arg=$2 , IROOT=`pwd`/${arg:=Intersect}
arg=$3 , COMND=${arg:=auto.cxm}


echo "!
! Video tape crunch command file.
! Read in $FRAMES frames, store them, and then recall and crunch
! each individually using $COMND as the command file template.
!
! This file was created on `date`.
!
!       First read the video tape
ERALL
INIT
TVPACK t 525,$FRAMES,0
!
! Then store the data" > $FILE

# We need to look at A1, A2, A3, A4, B1, ....       2  |  1
# until we've stored as many frames as were         ---+---
# scanned in.  The frames are stored like this:     4  |  3
# (See the MegaVision TVPACK help.)  This
# determines the values for Xoff and Yoff and the order in which
# they change...

FRM_NUM=0

for MEMORY in A1 A2 A3 A4 B1 B2 B3 B4 C D E F
do
  for Yoff in 0 512
  do
```

```
    for Xoff in 512 0
    do
      FRM_NUM=`expr $FRM_NUM + 1`
      if test $FRM_NUM -gt $FRAMES
      then
         break 3
      fi
      FNAME=$VROOT$FRM_NUM
      echo "NSTORE $MEMORY $FNAME 1,512,512,$Xoff,$Yoff,1,1" >> $FILE
    done
  done
done

echo "!
! DATA STORED.
!" >> $FILE

FRM_NUM=1

while test $FRM_NUM -le $FRAMES
do

VNAME=$VROOT$FRM_NUM
INAME=$IROOT.$FRM_NUM
echo "!
! Now processing frame number $FRM_NUM
!
RECALL $VNAME B 0,0,2,2
AVERAGE B A4 3,5,0,1
`cat $COMND`
!
! Calculate and store the intersections
LOTS B,A4 C,$INAME
!" >> $FILE

FRM_NUM=`expr $FRM_NUM + 1`
done
```

```
!
! Video tape crunch command file.
! Read in 5 frames, store them, and then recall and crunch
! each individually using model.cxm as the command file template.
!
! This file was created on Thu Oct  6 11:01:36 EDT 1988.
!
!      First read the video tape
ERALL
INIT
TVPACK t 525,5,0
!
! Then store the data
NSTORE A1 /usr/thor/gendrich/vision/Video1 1,512,512,512,0,1,1
NSTORE A1 /usr/thor/gendrich/vision/Video2 1,512,512,0,0,1,1
NSTORE A1 /usr/thor/gendrich/vision/Video3 1,512,512,512,512,1,1
NSTORE A1 /usr/thor/gendrich/vision/Video4 1,512,512,0,512,1,1
NSTORE A2 /usr/thor/gendrich/vision/Video5 1,512,512,512,0,1,1
!
! DATA STORED.
!
!
! Now processing frame number 1
!
RECALL /usr/thor/gendrich/vision/Video1 B 0,0,2,2
AVERAGE B A4 3,5,0,1

        .
        .  model command file is inserted here
        .  processing data from memory A4 and
        .  depositing the intersections in B1.
        .
!
! Calculate and store the intersections
DOTS B,A4 C,/usr/thor/gendrich/vision/Intersect.1
!
!
! Now processing frame number 2
!
RECALL /usr/thor/gendrich/vision/Video2 B 0,0,2,2
AVERAGE B A4 3,5,0,1

        .
        .  model command file is inserted here
        .  processing data from memory A4 and
        .  depositing the intersections in B1.
        .
!
! Calculate and store the intersections
DOTS B,A4 C,/usr/thor/gendrich/vision/Intersect.2
!
!
! Now processing frame number 3
!
RECALL /usr/thor/gendrich/vision/Video3 B 0,0,2,2
AVERAGE B A4 3,5,0,1
```

```
        .
        . model command file is inserted here
        . processing data from memory A4 and
        . depositing the intersections in B1.
        .
!
! Calculate and store the intersections
DOTS B,A4 C,/usr/thor/gendrich/vision/Intersect.3
!
!
! Now processing frame number 4
!
RECALL /usr/thor/gendrich/vision/Video4 B 0,0,2,2
AVERAGE B A4 3,5,0,1

        .
        . model command file is inserted here
        . processing data from memory A4 and
        . depositing the intersections in B1.
        .
!
! Calculate and store the intersections
DOTS B,A4 C,/usr/thor/gendrich/vision/Intersect.4
!
!
! Now processing frame number 5
!
RECALL /usr/thor/gendrich/vision/Video5 B 0,0,2,2
AVERAGE B A4 3,5,0,1

        .
        . model command file is inserted here
        . processing data from memory A4 and
        . depositing the intersections in B1.
        .
!
! Calculate and store the intersections
DOTS B,A4 C,/usr/thor/gendrich/vision/Intersect.5
!
```

```
!
! Image processing command file
!
! Works with all:
!               Ektachrome wing images
!               ASA 1000 wing images
! NOTE: The angles through which the picture is rotated are dependent on
!     the setup of the beam dividers.  These values will change from one
!     setup to the next, requiring modification of the 'ROTATE' angle at
!     the start and end of processing each half frame.
!
! History:
! <cpg>  1 Jan 88 -- comments
! <cpg> 21 Jan 88 -- Instead of subtracting the background
!                    level, differ and HEQ are used.
!
! PRELIMINARIES
!
! First, subtract the background level to get a more even
! grey level across the entire screen.
average   a4    b4 1,87,3,0,1
subtract  b4,a4 c
threshold c     a3 0,143,2,2,2,2
!
! To eliminate the noise at the edges, crop A3.
erase     b3
block     a3 b3 64,64,959,959,64,64
move      b3 a3
!
!
!  START PROCESSING THE FIRST HALF FRAME:
! Rotate the image so the first set of lines is vertical
rotate    a3 b3 -25
!
! Emphasize the vertical lines -- their average within a vertical
! kernel is greater than the surroundings (and the other lines).
average   b3 a2 1,1,130,0,1
threshold a2 b2 0,168,2,2,2,2
not       b2 a1
!
! Fill in gaps from thresholding.
range     a1 b1 1,1,256,100,255,255
!
! Eliminate the thin noise spikes.
minimum   b1 a3 3,5
!
! Fatten up the lines a bit -- it helps for filling in the gaps.
rollor    a3 b3 3
!
! Now fill in the gaps.
range     b3 a2 1,1,256,110,255,255
rollor    a2 b2 5
!
! Eliminate any stray high order bits...
threshold b2 a . .
!
```

```
! Shrink lines to one pixel wide, being careful to eliminate any
! open circles and line stubs.  (ERODEC erodes everything but Contiguous
! objects.)
move      a1 b1
erodec    b1 a3 20
! (This fills in holes for any points that might not want to go away...)
rollor    a3 b3 3
! Now finish up the job...
erodec    b3 a3 36
!
! Put these lines into their original frame of reference.
rotate    a3 b3 25
!
! NOTE -- the origin wasn't moved in our cropping above (although it
!         could have been).  If the origin *is* moved, use ELOCK here
!         to put it back where it belongs.
!                                                    1      2
! Next, fatten up the lines so that intersections     1    2
! won't be missed by one line zigging while            1  2
! the other is zagging.  See illustration ==>           12
!                                                       21
! The intersection between lines "1" and               2  1
! "2" is a <NULL> instead of a point.                 2    1
!
rollor b3 a2 3
! END OF PROCESSING FIRST HALF FRAME.
!
! ... now do it all over with the lines in the other direction
threshold a2    b2 0,150,2,2,2,2
not       b2    a1
subtract  b4,a4 c
move      a1    b4
!                       store the first half frame in B4 when that
!                       memory is available
threshold c     a3 0,128,2,2,2,2
erase     b3
block     a3    b3 64,64,959,959,64,64
move      b3    a3
!
! START OF PROCESSING THE 2nd HALF FRAME
rotate    a3    b3 27
average   b3    a2 1,1,100,0,1
threshold a2    b2 0,166,2,2,2,2
not       b2    a1
_range    a1    b1 1,1,256,110,255,255
minimum   b1    a3 3,5
rollor    a3    b3 3
_range    b3    a2 1,1,256,110,255,255
rollor    a2    b2 7
threshold b2    a1 0,150,2,2,2,2
not       a1    b1
erodec    b1    a3 20
rollor    a3    b3 3
erodec    b3    a3 36
rotate    a3    b3 -27
rollor    b3    a2 3
```

```
! END OF PROCESSING THE 2nd HALF FRAME
!
! Here are the other lines.
threshold a2    b2 0,150,2,2,1,2
not       b2    a1
!
! A logical AND is the intersection of the two sets of lines
! (literally).  :-)
and       a1,b4 c
!
! The system uses C as a working memory.  Put our intersections
! somewnere safer.  :-|
move      c     b1
!
! Let's overlay the points on the original image.
! White objects in the graphics memory (F) are what
! we need to do that.
move      c     f
!
! Look at the original image...
view      a4
!
! ...and turn on the graphics display.
gon
```

```
      program VORTICITY
c
c     Description:
c           This program reads two data files produced by the
c     "object" function of the MegaVision 1024XM.  The labelled and
c     filtered objects are the points within the flow which we are
c     tracking.
c
c           This program performs a number of tasks:
c     1) reads the data
c     2) sorts it by increasing X and Y values
c     3) matches points in frame 1 to points in frame 2
c     4) interpolates in space and time to construct a velocity field
c        throughout the frame
c     5) connects the dots to form polygons,
c     6) uses the velocity information to determine derivatives,
c        stress and strain values, vorticity, etc.
c
c     Author: Chuck Gendrich (Give me unix or ...)
c
c     History:
c     August, 1987 -- 1st version -- written on freyja, an IRIS 3120
c           running unix System V.
c     December, 87 -- 1.1 <cpg> Added graphics display of the
c           calculations (velocities, vorticity, etc.)
c           Added TURB3D-formatted output.
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c$define DEBUG
$ include "vorticity.h"
c
      logical Again, UnDim
      integer NumPts, NumPoly
      external Again, NumPts, NumPoly
c           Again is the error handler -- prints diagnostics and asks
c           if the user wants to quit or re-run the preceding section.
c
c           UnDim comes back TRUE when output will be non-dim'd
c
      open( unit=1, file='/dev/tty')
c           unit1 -- tty input
      open( unit=2, file='/dev/tty')
c           unit2 -- tty output
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c     step 1 -- get the data
c
$ include "graphics.h"
c
10    continue
      call getfrm( frame1, frame2, UnDim, ierr)
      if( ierr.ne.0) then
c           error handle
            if( Again( ierr)) goto 10
      endif
```

```
c
c       step 2 -- sort the points.  this is necessary when using more
c                 sophisticated matching algorithms.
c
        call sortfrm( frame1)
        call sortfrm( frame2)
#ifdef DEBUG
        n = NumPts( frame1, 2)
        write( *, 1000) 'frame 1',n, (i,frame1(i,X),frame1(i,Y),i=1,n)
        n = NumPts( frame2, 2)
        write( *, 1000) 'frame 2',n, (i,frame2(i,X),frame2(i,Y),i=1,n)
1000    format(/' X-Y data for ',a7,' (',i2,' points):'/
     +  (' ',i2,': (',f8.3,',',f8.3,')'))
#endif
c
c       step 3 -- define the correspondence between the two frames (match)
c
20      continue
        call color( GREEN)
        call Matches( frame1, frame2, match, ierr)
        if( ierr.ne.0) then
                if( Again( ierr)) goto 20
        endif
#ifdef DEBUG
        write( *, 1010) match
1010    format(' Here is the match array:'/(i3))
#endif
c
c       step 4 -- interpolate the velocity vectors
c
        call Velocities( frame1, frame2, match, vel)
#ifdef DEBUG
        n = NumPts( vel, 4)
        write( *, 1020) (i,vel(i,X),vel(i,Y),vel(i,Vx),vel(i,Vy),i=1,n)
1020    format(/' Here are the velocity components and their',
     +' locations:'/'          X       Y       Vx      Vy'/
     +  (1x,i2,'.  ',4(1x,f7.2)))
#endif
c
c       step 5 -- define the polygons
c
30      continue
        call Polygons( vel, poly, ierr)
        if( ierr.ne.0) then
                if( Again( ierr)) goto 30
        endif
#ifdef DEBUG
        write( *, 1030)
1030    format(/' Here are the polygons which have been defined:'//
     +' ULH  URH  LRH  LLH')
        n = NumPoly( poly)
        write( *, 1040) (poly(i,1),poly(i,2),poly(i,3),poly(i,4),
     +  i=1,n)
1040    format(4(2x,i2,1x))
#endif
c
```

```
c       step 6 -- calculate fluid kinematic quantities
c
        call fluids( vel, poly, UnDim, ierr)
c
        end
```

```
      include "/usr/include/fgl.h"
      include "/usr/include/fdevice.h"
c
c       data declarations for the vorticity program
c
        integer MaxPts, MaxPoly
        parameter( MaxPts = 100)
c               the maximum number of points per frame
c
        parameter( MaxPoly = MaxPts)
c               the maximum number of polygons
c
        real BIGRAD, SMALLRAD
        parameter( BIGRAD = 3.0)
        parameter( SMALLRAD = 1.5)
c               large and small radii for outer- and inner-circle drawing
c
        real m( 4,4), scale
c               transformation matrix for sizing things correctly
c               and the value to put on the diagonal
c
        integer X, Y
        parameter( X = 1)
        parameter( Y = 2)
c               pointers to XY coords
c
        integer Vx, Vy
        parameter( Vx = 3)
        parameter( Vy = 4)
c               pointers to velocity components
c
        integer BAD
        parameter( BAD = 0)
c               when 'look' returns a BAD point, it didn't find a
c               corner in the indicated direction at the correct
c               distance.
c
        integer ddx, ddy
        parameter( ddx = X)
        parameter( ddy = Y)
c               pointers so that subprograms agree whether the
c               derivative wrt X or wrt Y should be taken
c
        real frame( MaxPts, 2)
        real frame1( MaxPts, 2)
        real frame2( MaxPts, 2)
c               XY coords of each point in frames 1 and 2
c
        integer match( MaxPts)
c               frame1(i) <--> frame2( match(i))
c
        integer poly( MaxPoly, 4)
c          making four-sided polynomials
c               poly( i, 1) is the upper-LH corner of the polygon
c               poly( i, 2) is the upper-RH corner
c               poly( i, 3) lowerLH
```

```
c               poly( i, 4) lowerRH
c
        real vel( MaxPts, 4)
c               velocity field --- Vx & Vy @ X,Y
c
        integer MatchDim
        parameter( MatchDim = 10)
        integer matchs( MatchDim)
c               watch the spelling!!!
c               Hopefully, no more than 10 points will match...
c
        character*30 title
c               for titling output
c
        real UP, DOWN, RIGHT, LEFT
        parameter( UP    =  90.0)
        parameter( DOWN  = 270.0)
        parameter( RIGHT =   0.0)
        parameter( LEFT  = 180.0)
c
        integer Xsize, Ysize
c               size of the window port opened up.
c
        logical GoOn
        external GoOn
c               a logical function which types (<RET> to continue) then
c               returns .TRUE.
c
        integer ierr
c               returns 0 when there is no error, otherwise...
c
        integer i, j, k, n, n1, n2, p
c               counters
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
```

```
        call foregr
c               the job in the textport needs to run in the foreground
        call KeepAs( 1024,767)
c               the window needs to have an aspect ratio of a:b
        i = winope( 'Vorticity Tool',14)
        call wintit('Vorticity Tool',14)
c               get a window and title it
        call getsiz( Xsize, Ysize)
c               find out how large it is, then make the transformation
c               matrix which will scale everything appropriately.
c
        if( Ysize.eq.1024.0) Ysize = 767.
c               mex isn't running.  Ysize is really 767...
        scale = 1024. / Ysize
c               1024. is the largest X or Y value my data will ever have.
c
        do 77777 i=1,4
           do 77777 j=1,4
                if( i.eq.j) then
                        m(i,i) = 1.0
                else
                        m(i,j) = 0.0
                endif
77777   continue
        m(4,4) = scale
c
c*      m(2,2) = -m(2,2)
c*      m(4,2) = Ysize
c*c             this shold flip the image vertically
c
        call multma( m)
        call color( WHITE)
        call clear
        do 77778 i=8,64
                call mapcol( i, i*4, 0, 150-(2*i))
77778   continue
c
c       Here are the predefined colors 1-7 (defined in 'fgl.n')
c
c       BLACK   RED     GREEN   YELLOW
c       BLUE    MAGENT  CYAN    WHITE
c
```

```
# define UPSIDE_DOWN
c# define MFRC
c       <M>ichael <F>ilm <R>eader <C>oords
# define YYMIN 70.0
c       actually -YYMIN .. Y = Y - YYMIN
# define XXMIN 30.0
c
# define YSPLIT 5.217881548
# define XSPLIT 5.217881548
c       The conversion is:
c               Y = (Ymfrc + YYMIN) * YSPLIT    and similarly for X.


        subroutine getfrm( frame1, frame2, UnDim, ierr)
c
c       Description:
c       Read in the data files, storing the points in FRAME1 and FRAME2.
c
c       Return Values:
c               X,Y points in frame1 and frame2
c               ierr = 0 if no error occurred
c
c       History:
c       <cpg> aug 87 --- v1.0
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
# include "vorticity.h"
c
        character*80 filnam1, filnam2
        real Xoffset, Yoffset
        integer NumPts
        external NumPts
        logical UnDim
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c       INITIALIZE both frame1 and frame2
c
        do 1 i=1, MaxPts
                frame1( i,X) = 0.0
                frame1( i,Y) = 0.0

                frame2( i,X) = 0.0
                frame2( i,Y) = 0.0
1       continue
c
c       FRAME1
1000    format(' Please enter the name of the file which contains'/
     + ' the data for frame 1.')
1010  format(' Error opening the data file ',a30/' Please try again.'/)
10      continue
c               return here on error 1
                filnam1 = 'frame1.dat'
                write( *, 1000)
                call getline( filnam1)
```

```
c
                call makefrm( frame1, filnam1, ierr)
                n = NumPts( frame1, 2)
                call color( BLUE)
                do 15 i=1, n
                        call circ( frame1(i,X), frame1(i,Y), BIGRAD)
15              continue
                if( ierr.eq.1) then
c                       try again if error opening the frame data file
                        write( *, 1010) filnam1
                        goto 10
                else if( ierr.ne.0) then
c                       calling routine will have to deal with the error
                        return
                endif
c
c       FRAME2
1020    format(' Please enter the name of the file which contains'/
     + ' the data for frame 2.',
20      continue
                filnam2 = 'frame2.dat'
                write( *, 1020)
                call getline( filnam2)
c
                call makefrm( frame2, filnam2, ierr)
                if( ierr.eq.1) then
                        write( *, 1010) filnam2
                        goto 20
                else if( ierr.eq.0) then
c                       get the X- and Y-offsets for this frame
                        write( *, 1030)
                        read( *, *) Xoffset, Yoffset
                        n = NumPts( frame2, 2)
                        do 30 i=1, n
                                frame2( i,X) = frame2(i,X) - Xoffset
                                frame2( i,Y) = frame2(i,Y) - Yoffset
30                      continue
                        call color( CYAN)
                        do 40 i=1,n
                           call circ( frame2(i,X), frame2(i,Y), BIGRAD)
40                      continue
c
                        call convert( 1.0,1.0,0.0,0.0,UnDim)
                        return
                endif
1030    format(' Please enter the X- and Y-offsets for frame 2.'/
     +' They should be chosen such that (0,0) in frame 1 is the'/
     +' point (Xoffset, Yoffset) in frame 2.')
        end

        subroutine makefrm( frame, filnam, ierr)
c
c       open the frame data file
c       throw away the first (descriptor) line
c       while not EOF
c               find out how many points are in the next set
```

```
c               read X values
c               read Y values
c               convert X and Y to a std coord system
c       end while
c
c       The following is an example of the data contained in the file:
c ORIGINAL OBJECTS
c
c OBJECT LABEL          □   1     □   2     □   3     □   4     □   5
c X-CENTROID            225.75    662.50    497.44    861.73    705.30
c Y-CENTROID            109.25    115.00    133.78    185.91    234.80
c
c OBJECT LABEL          □  36     □  37     □  38
c X-CENTROID            445.78    570.09    685.25
c Y-CENTROID            916.00    922.36    940.25
c123456789 123456789 123456789 123456789 123456789 123456789 123456789 12345
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
□ include "vorticity.h"
c
        integer set, start
        character*80 filnam
        character*80 line1, line2, line3, junk
        integer HowMany
        external HowMany
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
        open( 3, err = 2000, file=filnam, status = 'old')
        print*, ' Reading data from ',filnam
c
1000    format( a80)
1010    format( 17x, 5(4x, f7.2))
1020    format(' Zero objects found in coordinate set ',i2,'.')
        read( 3, 1000, end=2010) junk
c
        start=0
c               start stores the current number of points that have
c               been read in.
        set = 0
100     continue
            read( 3, 1000, end=2020) junk
c
            read( 3, 1000, end=2050) line1
            read( 3, 1000, end=2050) line2
            read( 3, 1000, end=2050) line3
c
            set = set + 1
            n = HowMany( line1)
c
            if( n.gt.0) then
               read( line2, 1010) (frame(i,X), i=start+1, start+n)
               read( line3, 1010) (frame(i,Y), i=start+1, start+n)
               start = start + n
            else
               write( *, 1020) set
```

```
            endif
c
        goto 100
c
2000    continue
c       error opening the frame data file
        ierr = 1
        return
c
2010    continue
c       empty file
        ierr = 2
        close( 3)
        return
c
2020    continue
c       end of file --- OK
        close( 3)
        ierr = 0
□       ifdef MFRC
        do 2025 i=1,start
                frame(i,Y) = (frame(i,Y) + YYMIN) * YSPLIT
                frame(i,X) = (frame(i,X) + XXMIN) * XSPLIT
2025    continue
□       endif
□       ifdef UPSIDE_DOWN
        do 2030 i=1,start
                frame(i,Y) = -frame(i,Y) + 1024.
2030    continue
□       endif
        return
c
2050    continue
c       wrong number of lines.  three should appear at once --
c       OBJ NUMBER / X-COORDS / Y-COORDS ---> Frame probably incorrect
        ierr = 3
        close( 3)
        return
        end

        integer function HowMany( line)
c
c       Check that the first word is OBJECT.
c       Then count to see how many □ signs there are.
c
c       We're parsing this line:
c OBJECT LABEL          □   36      □   37      □   38
c123456789 123456789 123456789 123456789 123456789 123456789 123456789 12345
c
c       so we'll only look every 11 places for the □.
c
        character*80 line
        integer pos
        character*6 word, obj
        character*1 lb
c
```

```
        parameter( obj='OBJECT')
        parameter( lb =' ')
c
c
1000    format( 1x, a6)
        read( line, 1000) word
        if( word.ne.obj) then
                HowMany = 0
                return
        endif
        n = 0
c       Use n as a counter for the number of objects
        do 100 i= 1,5
c               5 is the maximum number of points possible
                pos = 11*i + 12
                if( line(pos:pos).eq.lb) then
                        n = n+1
                endif
100     continue
        HowMany = n
        return
        end
```

```
        integer function NumPts( points, dim)
c
# include "vorticity.h"
c
        integer dim
        real points( MaxPts, dim)
c
        n = 0
10      continue
c
        if( points(n+1, X).eq.0.0.and.points(n+1, Y).eq.0.0) goto 20
        n = n+1
        if( n.eq.MaxPts) goto 20
        goto 10
c
20      continue
c
c       "n" now contains the number of points
c
        NumPts = n
        return
        end

        integer function NumPoly( poly)
c
# include "vorticity.h"
c
        n = 0
100     continue
        if( poly(n+1,1).eq.0) goto 200
        n=n+1
        if( n.eq.MaxPoly) goto 200
        goto 100
c
200     continue
c
c       "n" now contains the number of polygons
c
        NumPoly = n
        return
        end
```

```
      subroutine sortfrm( frame)
c
c     Description: Uses a simple exchange sort to order FRAME
c     on decreasing Y.  Ignores (0,0) pairs.
c
c     This sort should be efficient enough given the small number
c     of points which can be in a frame.  If MaxPts goes above 200,
c     someone should consider exchanging this for a shell sort or
c     something else more efficient.
c
c     Author: Chuck Gendrich
c
c     History:
c     August, 1987 v1.0
c
c     Procedure:
c             Determine how many points there are
c             For all points except the last
c                     check this point against all subsequent
c                     if X > Xsubsequent,
c                             swap X's and Y's
c                     if X == Xsubsequent
c                             check Y's and swap if appropriate
c             end for
c
c
cccccccccl2ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
      include "vorticity.h"
c
      n = NumPts( frame, 2)
c             find out how many points there are in this frame
c
      do 100 i=1,n-1
c         check all points but the last
          do 75 j=i+1, n
c               compare it against all remaining points
                if( frame(i,Y).lt.frame(j,Y)) then
                      call swap( frame(i,X), frame( j,X))
                      call swap( frame(i,Y), frame( j,Y))
                else if (frame(i,Y).eq.frame(j,Y)) then
c                     compare X's
                      if( frame(i,X).lt.frame(j,X)) then
                              call swap( frame(i,X), frame( j,X))
                              call swap( frame(i,Y), frame( j,Y))
                      endif
                endif
75        continue
100   continue
      return
      end

      subroutine swap( a, b)
c
      real a, b, tmp
c
```

```
        tmp = b
        b = a
        a = tmp
c
        return
        end
```

```
      subroutine Matches( frame1, frame2, match, ierr)
c
c     Description: Establishes a correspondence between points
c     in frame1 and frame2.  Match is setup such that
c
c             frame1( i) <--> frame2( match(i))
c
c     Algorithm:  (What algorithm?  Huh? Who? Where?  :-(  )
c       We'll be looking for points within a certain neighborhood of
c       the original point.  At the moment, we'll assume there's only
c       one point which will fall into the neighborhood, but that may
c       not be the case.
c
c       We'll error-check.  If more than one point matches, this
c       routine will return an error.  If no point matches, a zero
c       will be put in the appropriate place in Match.
c
c     Author: Chuck Gendrich ("I wish I knew how to do this one better.")
c
c     History:
c     August, 1987 v1.0
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
      include "vorticity.h"
c
      real close, close2
      parameter( close  = 23)
      parameter( close2 = close*close)
c             half the distance in pixels between the two closest
c             points in the original image (FENCE1)
c
      real dist
      integer NumPts
      external dist, NumPts
c
      integer found
c             how many matches have we found
c
c     initialize Match
c
      do 10 i=1, MaxPts
             match(i) = 0
10    continue
c
c     OK, we'll do it the hard way.  Compare every point in frame1
c     with every point in frame2.  Tally all the distances less than
c     CLOSE.  If there's only one point, wonderful.  Otherwise, oops.
c
      n1 = NumPts( frame1, 2)
      n2 = NumPts( frame2, 2)
c
      do 100 i=1, n1
         found = 0
         x1 = frame1(i,X)
         y1 = frame1(i,Y)
```

```
            do 75 j=1, n2
                 x2 = frame2(j,X)
                 y2 = frame2(j,Y)
                 dx = x2-x1
                 dy = y2-y1
                 if( (dx*dx + dy*dy).lt.close2) then
                         found = found+1
                         if( found.gt.MatchDim) then
                                 ierr = 6
                                 return
                         endif
                         matchs( found) = j
                 endif
75          continue
            if( found.eq.0) then
                 match(i) = 0
            else if( found.eq.1) then
                 match(i) = matchs( 1)
            else
c                Of course this should be made more sophisticated so that
c                it can conflict-resolve any problems.  But for now, just
c                chalk up an error and return.
                 ierr = 7
                 return
            endif
100       continue
          ierr = 0
          return
          end
```

```
      subroutine Velocities( frame1, frame2, match, vel)
c
c     Description:  vel_maker computes the position and both
c     components of the velocity vector between matching points
c     in frame1 and frame2.
c
c     The position is assumed to be at the midpoint between the two
c     matching points.  If there is no matching point in FRAME2 for
c     some point in FRAME1 (match(i) == 0), the position and velocity
c     components are left 0.
c
c     The initial "velocity" is in dPixels.  A conversion for pixels
c     to real units of measure, and knowledge of dTime between frames
c     is required so that we can calculate the real velocity.  This
c     will be done in a subsequent routine.
c
c     Author: Chuck Gendrich
c
c     History:
c     August, 1987 v1.0
c
cccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
      include "vorticity.h"
c
      character*80 line
      integer NumPts
      real avg, getreal
      external avg, NumPts, getreal
c
c     initialize the velocity descriptor array
c
      do 10 i=1, MaxPts
            vel(i,X) = 0.0
            vel(i,Y) = 0.0
            vel(i,Vx) = 0.0
            vel(i,Vy) = 0.0
10    continue
c
      write( *, 1000)
1000  format(' Please enter how many pixels a point moving at'/
     +' the free stream velocity will move between frames.'/
     +' (Vx) = (deltaX) + baseV')
      line  = ' -36.1855      pixels/frame'
      baseV = getreal( line)
      scale = 4.0 * exp( -abs(baseV) / 28.0)
c
      n = NumPts( frame1, 2)
      i = 0
c
      do 100 j=1, n
c        make a corresponding entry in vel for each point in frame1
c
         if( match(j).ne.0) then
c             we have something to work with
c
```

```
                i = i+1
                vel(i,X)  = avg(frame1( j,X), frame2( match(j),X))
                vel(i,Y)  = avg(frame1( j,Y), frame2( match(j),Y))
                vel(i,Vx) = frame2(match(j),X) - frame1(j,X) + baseV
                vel(i,Vy) = frame2(match(j),Y) - frame1(j,Y)
                call move2( vel( i,X), vel(i,Y))
                call vector( vel( i,Vy), vel( i,Vx), scale)
c
            endif
100     continue
c
c       Now construct the TURB3d output files.
c
        open( unit=3, file='vel.xyz')
c               unit3 -- flow field point locations
        open( unit=4, file='vel.q')
c               unit4 -- flow field information
c
        write( 3, 1010) i
c       write out the X and Y locations
        write( 3, 1020) (vel( j,X),j=1,i)
        write( 3, 1020) (vel( j,Y),j=1,i)
c
        write( 4, 1030) i
c       write out the Vx and Vy velocities
        write( 4, 1020) (vel( j,Vx),j=1,i)
        write( 4, 1020) (vel( j,Vy),j=1,i)
        do 110 j=1,i
c               write out a fictitous pressure value
                write( 4, 1040)
110     continue
        do 120 j=1,i
c               write out a fictitous vorticity value (for now)
                write( 4, 1040)
120     continue
1010    format( i3,' 1')
1020    format( 5(f10.5,1x))
1030    format( i3,' 1'/'1 1 1 1')
1040    format(' 1')
c
        return
        end
```

```
        real function dist( x1, y1, x2, y2)
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
        real x1, y1, x2, y2, sqrd
c
        sqrd = (x1-x2)**2 + (y1-y2)**2
c
        dist = sqrt( sqrd)
c
        return
        end

        real function avg( a, b)
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
        real a, b
c
        avg = (a+b)/2.0
c
        return
        end

        real function tan1( dy, dx)
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c       returns the arctan in degrees given dx and dy
c
        real dx, dy, angle, pi180, small
c
        parameter( small = 0.0001)
c               to keep from dividing by zero
c
        parameter( pi180 = 57.295779513)
c               180/pi for converting to degrees
c
        if( abs( dx).lt.small) then
c               let's not divide by zero, shall we?
                if( dy.gt.0) then
                        tan1 = 90.0
                else if( dy.lt.0) then
                        tan1 = 270.0
                else
                        tan1 = 0.0
c                       ??? punt ???
                endif
        else
                angle = atan( dy/dx) * pi180
                if( dx.gt.0) then
c                       first or fourth quadrant -- atan is OK
                        if( angle.lt.0.0) then
                                angle = angle + 360.0
c                               make sure angle is > 0
                        endif
```

```
                        tanl = angle
                else
c                       second or third quadrant -- add 180 deg.
                        tanl = angle + 180.0
                endif
        endif
        return
        end

        real function interp( xx0, ffx0, xx2, ffx2, xl, ierr)
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c       returns the interpolated value for f(x) given the function
c       value at boundaries x0 and x2.  xl has to lie between x0 and xl.
c
        real x0, fx0, x2, fx2, xl, xx0, ffx0, xx2, ffx2
        integer ierr
c
        real small
        parameter( small = 0.00001)
c
        real dx0, dfx
c
        x0  = xx0
        fx0 = ffx0
        x2  = xx2
        fx2 = ffx2
c
c       since we don't want to change the original values for x0 and x2,
c       etc., we make our own copies which we can switch around if
c       necessary.
c
        interp = fx0
c
c       (junk answer just in case they don't check ierr and interp's
c        value isn't changed because of some error...)
c
        dx0 = x2 - x0
        if( abs( dx0).lt.small) then
c               let's avoid division by zero, eh?
                ierr = 16
                return
        else if( dx0.lt.0) then
c               x0 > x2  ---> We have to switch them around
                call swap( x0, x2)
                call swap( fx0, fx2)
                dx0 = -dx0
c       else
c               they're already in the right order and plenty far apart
        endif
c
        if( xl.gt.x2.or.xl.lt.x0) then
c               out of bounds (oops)
                ierr = 17
                return
```

```
        endif
c
        dx1 = x1 - x0
        dfx = fx2 - fx0
c
        interp = fx0 + ( dfx * dx1 / dx0)
        return
        end

        subroutine centroid( poly, vel, i, x0, y0, ierr)
c
c       Description: The centroid of a four-sided figure is
c       calculated.  This is located at the intersection of the two
c       lines which join the midpoints of opposing sides.
c
c       Author:  Chuck Genarich
c
c       History:
c       August, 1987  v1.0
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c␣ define DEBUG
␣ include "vorticity.h"
c
        real small
        parameter( small = 0.00001)
c
        real avg
        external avg
c
        real x0, y0
c               the coordinates for the centroid          A         B
c                                                         .---------.
c       we'll define the polygon as one consisting of     \         |
c       four corners, A, B, C, and D.  The coordinates      \        |
c       of the corners are Ax, Ay, Bx, By, etc....  a,b,c,  \       |
c       and d point to the appropriate entries into vel.    D.__    |
c                                                              -__.|C
        integer a,b,c,d
        real Ax, Ay, Bx, By, Cx, Cy, Dx, Dy
c
        real m1, m2, b1, b2
c               we'll calculate the slope and intercept of both lines
c               which join the midpoint of opposing sides.  To find the
c               intersections, we let y1 = y2 and solve for x.  Doing
c               this we find  x0 = -(b2 - b1)/(m2 - m1).  Plugging back
c               in, we find   y0 = m1 * x + b1 (= m2 * x0 + b2....)
c
        real deltaY, deltaX, y1, x1, y2, x2
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
        a = poly( i,1)
        b = poly( i,2)
        c = poly( i,3)
```

```
        d = poly( i,4)
c
        Ax = vel( a,X)
        Ay = vel( a,Y)
        Bx = vel( b,X)
        By = vel( b,Y)
        Cx = vel( c,X)
        Cy = vel( c,Y)
        Dx = vel( d,X)
        Dy = vel( d,Y)
c
        ifdef DEBUG
        write( *, 990) a,Ax,Ay,b,Bx,By,c,Cx,Cy,d,Dx,Dy
990     format(/' centroid: finding the centroid with these corners --'/
     + (4x,i2,':  (',f8.3,',',f8.3,')'))
1000    format(' Wierd centroid (no dx or dy) at (',f8.3,',',f8.3,')')
1010    format(' centroid: neither line vertical.  m1: ',f8.3,
     +' m2: ',f8.3/' b1: ',f8.2,' b2: ',f8.2,' --> (',f8.3,',',
     + f8.3,')'/)
1020    format(' centroid: ',a6,' vertical.  m: ',f8.3,' b: ',
     + f8.3/'   ---> (',f8.3,',',f8.3,')'/)
        endif
c
c  ---  initialize x0 and ierr (x0 must be known later)
c
        x0   = -99999.0
        ierr = 0
c
c  ---  line 1 first
c
        y1 = avg( Ay, By)
        x1 = avg( Ax, Bx)
c
        deltaY = avg( Dy, Cy) - y1
        deltaX = avg( Dx, Cx) - x1
c
        if( abs( deltaX).lt.small) then
c               let's see if we'll be dividing by zero
                if( abs( deltaY).lt.small) then
c                       midpt(AB) = midpt(CD) (?) and the centroid is there
                        y0 = y1
                        x0 = x1
                        ifdef DEBUG
                        write( *, 1000) x0, y0
                        endif
                        return
                else if( abs( deltaX/deltaY).lt.small) then
c                       midpt(AB) ---> midpt(CD) is a vertical line
                        m1 = x0
                        x0 = x1
                endif
        else
                m1 = deltaY / deltaX
                b1 = y1 - m1*x1
        endif
c
```

```
c  ---  line 2 next
c
       y2 = avg( Ay, Dy)
       x2 = avg( Ax, Dx)
c
       deltaY = avg( By, Cy) - y2
       deltaX = avg( Bx, Cx) - x2
c
       if( abs( deltaX).lt.small) then
              if( abs( deltaY).lt.small) then
c                    midpt(BC) = midpt(DA) (?) and the centroid is there
                     y0 = y2
                     x0 = x2
D                    ifdef DEBUG
                     write( *, 1000) x0, y0
D                    endif
                     return
              else if( abs( deltaX/deltaY).lt.small) then
c                    midpt(BC) ---> midpt(DA) is a vertical line
                     if( x0.ne.-99999.0) then
c                           the other line was also vertical
c                           Punt!
                            ierr = 4
                            return
                     else
                            m2 = x0
                            x0 = x2
                     endif
              endif
       else
              m2 = deltaY / deltaX
              b2 = y2 - m2*x2
       endif
c
c  ---  now for the intersection (i.e., the centroid)
c
c     -- calculate x0 if we still need to
       if( x0.eq.-99999.0) then
c             neither line was vertical
              x0 = -(b2 - b1) / (m2 - m1)
              y0 = m2 * x0 + b2
D             ifdef DEBUG
              write( *, 1010) m1, m2, b1, b2, x0, y0
D             endif
              return
       endif
c
c     -- calculate y0 using the appropriate slope and intercept.
       if( m1.eq.-99999.0) then
c             line 1 is vertical so use m2 and b2
D             ifdef DEBUG
              write( *, 1020) 'line 1', m2, b2, x0, y0
D             endif
              y0 = m2 * x0 + b2
       else
c             use m1 and b1
```

```
                y0 = m1 * x0 + b1
#               ifdef DEBUG
                write( *, 1020) 'line 2', m1, b1, x0, y0
#               endif
        endif
c
        return
        end
```

```
      subroutine Polygons( vel, poly, ierr)
c
c     Description:  Establishes a polygon descriptor array.  Each
c       element in poly(i) describes a corner of the polygon according
c       to the following scheme:
c
c             poly( i,1) --> upper LH corner
c             poly( i,2) --> upper RH corner
c             poly( i,3) --> lower LH corner
c             poly( i,4) --> lower RH corner
c
c     Procedure: Try to make roughly rhomboidal-shaped polygons.  Try
c     to use each point in VEL as an upper LH corner.  If it doesn't
c     work, move on to the next point.
c
c     Author: Chuck Gendrich ("SCORPIO:  all scorpio's are assassinated.")
c
c     History:
c     15 aug 87 <cpg> v1.1 -- removed the triangle logic.  look in
c                             older versions to recover it.
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
      include "vorticity.h"
c
      real delta
c             when we look for a corresponding corner, it will be in a
c             particular direction +/- delta deg.
c
      integer ULH, URH, LRH, LLH
c             pointers to the UpperLeftHand, UpperRH, LRH, and LLH
c             vertices of the polygon under construction
c
      logical IncrP
      integer NumPts, look
      external NumPts, look, IncrP
c
c
      n = NumPts( vel, 4)
      delta = 30.0
c             look for point @ direction +/- delta deg.
      p = 0
c             we have currently found zero polygons
c
      do 100 ULH=1, n
c
         URH = look( ULH, vel, 4, RIGHT, delta, matchs, ierr)
c
         if( URH.ne.BAD.and.ierr.eq.0) then
c           found the URH corner
            LRH = look( URH, vel, 4, DOWN, delta, matchs, ierr)
            if( LRH.ne.BAD.and.ierr.eq.0) then
               LLH = look( LRH, vel, 4, LEFT, delta, matchs, ierr)
               if( LLH.ne.BAD.and.ierr.eq.0) then
c                    we have an OK polygon
                     if( IncrP( p)) then
```

```
                                poly( p, 1) = ULH
                                poly( p, 2) = URH
                                poly( p, 3) = LRH
                                poly( p, 4) = LLH
                        else
                                return
c                               doesn't make sense to look for any more
c                               polygons if poly is full....
                        endif
                    endif
                endif
            endif
            if( ierr.ne.0) then
c               there was an error "looking"
                return
            endif
c
c           in every other case, we couldn't find a satisfactory polygon.
c           Forget using this corner as the start of a polygon.
c
100     continue
        return
        end

        logical function IncrP( p)
c
c       simple-minded routine to increment p or quit if there's a
c       problem
c
c       returns: TRUE if p is <= the size of poly
c
        include "vorticity.h"
c
        p = p+1
        IncrP = (p.le.MaxPoly)
c
        return
        end
```

```
      integer function look( p,points,dim,angle,delta, matchs, ierr)
c
c     Description:  "look" uses the location of point 'p' among
c     the given points.  It examines all the other points for ones
c     which are between 100 and 195 pixels away, lying between
c     vectors at angles "angle + delta" and "angle - delta".
c
c
c                          _->> angle + delta
c                       _-=/                                  no good
c                    _-=/                 )              )        .
c                 _-=/                   )              )
c           p  -                        100            195
c           o <-------------------------|--------------|-----> angle
c              -=_                      )              )
c                 \-=_                  )   OK         )
c                     \-=_              )    .         )
c                         \-=_
c                             \->> angle - delta
c
c
c     Return value:  "look" returns BAD if no point falls within
c       the defined search region.
c
c       If only one point matches, "look" returns the pointer to that
c       point.
c
c       If more than one point matches, "look" returns a pointer to the
c       one which is closer to the preferred direction.
c
c     Author: Chuck Gendrich
c
c     History:
c     August, 1987 v1.0
c
cccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
      include "vorticity.h"
c
      integer dim
      real points( MaxPts, dim)
c
      real angle, delta
c
      integer NumPts
      external NumPts
c
      integer found
      real AplusD, AminusD
c             -- to store Angle + Delta and Angle - Delta
c
      real x1, y1, x2, y2, dx, dy
      real d, direction
c             distance and direction from (x1,y1) to (x2,y2)
c
      real MinD, MaxD
```

```
        parameter( MinD =  40.0)
        parameter( MaxD = 180.0)
c               the closest and farthest that a point has to be
c               in order for it to be considered.
c
        real Distnc( MatchDim)
c               difference between the desired and the actual angle
c
        integer best
c               a pointer to the best match (when there's more than one)
c
        logical first, DoDraw
        character*1 char
        save
        data first/.TRUE./
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
        if( FIRST) then
                first = .FALSE.
                print*,'Draw in "look" lines? [n]'
                read(*,'(A1)') char
                DoDraw = (char.ne.'n'.and.char.ne.'N'.and.char.ne.' ')
        endif
c
        ierr = 0
        n = NumPts( points, dim)
        do 1 i=1, MatchDim
                matchs(i) = 0
1       continue
c
        if( n.lt.p) then
c               i.e., are we going to be doing something reasonable?
c               if we get here, the answer is "NO"
                ierr = 11
                return
        endif
c
        found   = 0
        AplusD  = angle + delta
        AminusD = angle - delta
c
c       Get the X and Y coords we're going to be comparing against, then
c       look at every other point...
c
        x1 = points( p, X)
        y1 = points( p, Y)
        if( DoDraw) then
                call color( WHITE)
                call circ( x1, y1, BIGRAD)
                call circ( x1, y1, SMALLRAD)
        endif
c
        do 100 i=1, n
c               ...to avoid excessive indenting and nesting, the tests
c               will be for disqualification.  if a point disqualifies
```

```
c               we'll simply 'continue' on to the next one.
c
                if( i.eq.p) goto 100
c
                x2 = points( i, X)
                y2 = points( i, Y)
                if( DoDraw) then
                        call circ( x2, y2, SMALLRAD)
                endif
c
                dx = x2-x1
                dy = y2-y1
                d  = sqrt( dx*dx + dy*dy)
c
                if( d.lt.MinD.or.d.gt.MaxD) goto 100
c
                direction = tan1( dy, dx)
c
                if( direction.gt.AplusD) direction = direction - 360.0
c                       NB -- tan1 always returns 0 <= angle <= 360
c                       but if AminusD < 0 and the direction really *is*
c                       OK at its position in the fourth quadrant, it'll
c                       never match.  Plus, we have to have A+D > A - D...
c                       Therefore, if direction > A+D, we'll give it a
c                       chance to match with angles in the 4th quad.
c
                if( direction.le.AplusD.and.direction.ge.AminusD) then
                    found = found + 1
                    if( found.gt.MatchDim) then
                       ierr = 12
                       return
                    endif
                    if( DoDraw) then
                        call circ( x2, y2, BIGRAD)
                        call color( RED)
                        call move2( x1, y1)
                        call draw2( x2, y2)
                        call color( WHITE)
                    endif
                    Distnc( found) = dist( x1, y1, x2, y2)
                    matchs( found) = i
                endif
100     continue
c
c       So now we've looked at every point
c
        if( found.eq.0) then
                look = BAD
        else if( found.eq.1) then
                look = matchs(1)
        else
c               look for the match which deviates the least from the
c               desired angle
                best = 1
                do 200 i=2, found
                if( Distnc(i).lt.Distnc(best)) then
```

```
                        best = i
                endif
200             continue
                look = matchs( best)
        endif
        return
        end
```

```
      subroutine fluids( vel, poly, UnDim, ierr)
c
c     Description:  calculates the fluid mechanical properties of the
c     flow.  The object is to produce plot(s) of
c        (1) du/dy
c        (2) du/dx
c        (3) dv/dy
c        (4) dv/dx
c        (5)  uv
c        (6)  Wz (vorticity)
c
c     Output: all output will be written to stdout.  If you want it in
c     a file, put a tee on the process when you run it.
c         e.g.:   freyja>  vorticity | tee output_file
c
c     Procedure: First the conversion factor for pixel <--> mm is
c     obtained, along with the time between each frame and the
c     absolute Y offset for both frames.
c
c     Author: Chuck Gendrich
c
c     history:
c     August, 1987 v1.0 -- lots of linear interpolation.  ugh.
c     15 aug 87 <cpg> v1.1 -- removed triangle logic.  see older
c                             versions to recover it.
c     16 aug 87 <cpg> v1.2 -- added the non-dim title stuff
c     22 jan 88 <cpg> v1.3 -- graphics to draw the polygons
c      7 apr 88 <cpg> v1.4 -- defines a colormap based on Wz[min-max]
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c# define DEBUG
# include "vorticity.h"
c
      real dd, d, avg, dist
      integer NumPts, NumPoly
      external NumPts, NumPoly, d, avg, dist
c
      integer a, b
c             a and b point to the vertices of the side along which
c             the velocity is to be integrated next
c
      real suba, subi
      real area, intgrl, GAMMA
c             area of the polygon which is integrated
c             integral of V * ds around the polygon
c             GAMMA is the total circulation throughout the frame
c
      logical UnDim
c             from convert... TRUE if results are being UnDim'd
c
      character*2 num
      character*5 XYunits, result
      character*7 Runits
c             for titling the output (depending on whether
c             the results are being non-dimensionalized or not...
```

```
c
      real Parray( 2, 4)
c             parray - points array, XY locations of polygon vertices
c             iarray - intensity array.. vorticity at the corners.
c
      real Wzs( MaxPoly), WZmax, WZmin
c             Stores the values of Wz, their max, and min.
      integer Csplit, shade
      parameter( Csplit= 56)
c             the total number of colors which we can draw
c
c
1000  format('       X',a5,'       Y',a5,3x,a5,a7)
c
      p = NumPoly( poly)
      n = NumPts( vel, 4)
      if( UnDim) then
            XYunits = '(noD)'
      else
            XYunits = '(mm)'
      endif
c
c   === vorticity
c
      if( GoOn()) then
            title  = 'vorticity'
      endif
      result = '  Wz '
      call CutHere( title)
      if( UnDim) then
            Runits = '(noD)'
      else
            Runits = '(1/sec)'
      endif
      write( *, 1000) XYunits, XYunits, result, Runits
c
      GAMMA = 0.0
      call color( WHITE)
      call clear()
      WZmin=+999999.
      WZmax=-999999.
      do 100 i=1, p
         call color( RED)
         area = 0.0
         intgrl = 0.0
c
         do 50 j=1,3
              a = poly(i,j)
              b = poly(i,j+1)
c                 a and b are the endpoints of the side along which
c                 V * ds is to be integrated.
              call integrt( a, b, vel, suba, subi)
c
              xl = vel( a, X)
              x2 = vel( b, X)
              yl = vel( a, Y)
```

```
                y2 = vel( b, Y)
c
                call move2( x1, y1)
                call draw2( x2, y2)
                area   = area + suba
                intgrl = intgrl + subi
c
50          continue

c           now for the last side (4 --> 1)
            a = poly(i,4)
            b = poly(i,1)
            call integrt( a, b, vel, suba, subi)
            x1 = vel( a, X)
            x2 = vel( b, X)
            y1 = vel( a, Y)
            y2 = vel( b, Y)
c
            call move2( x1, y1)
            call draw2( x2, y2)
c
            area   = area + suba
            intgrl = intgrl + subi
            Wz = intgrl / area
            Wzs(i) = Wz
            if( Wz.lt.WZmin) WZmin=Wz
            if( Wz.gt.WZmax) WZmax=Wz
c
            call centroid( poly, vel, i, x0, y0, ierr)
            if( ierr.ne.0) goto 100
c
            intgrl = intgrl * 50.0 * .029520809 * .029520809 / 1.0
            GAMMA  = GAMMA + intgrl
# ifdef DEBUG
            print*,' Area:',area,'  intgrl:',intgrl,' GAMMA now.',GAMMA
# endif
            call convert( x0, y0, Wz, 0.0, UnDim)
            call color( CYAN)
            call cmov2( x0, y0)
            write( num,'(i2)') i
            call charst( num, 2)
c
100     continue
        GAMMA = GAMMA
c               circ = (pix**2 / frame) * (frame/sec) * (mm/pix)**2
        print*,'Total circulation is ',GAMMA,' (mm^2/sec).'
        if( GoOn()) then
           do 150 i=1,p
c               draw shaded polygons
                shade = (Wzs(i) - WZmin)/(WZmax-WZmin)*(Csplit)+8
# ifdef DEBUG
                print*,'Vorticity ',i,' is ',Wzs(i),'. Shade is ',shade
# endif
                call color( shade)
                do 125 j=1,4
                   Parray( X, j) = vel( poly(i,j),X)
```

```
                  Parray( Y, j) = vel( poly(i,j),Y)
125             continue
                call polf2( 4, Parray)
150         continue
        endif
        if( GoOn()) then
                continue
        endif
c
c   === du/dy
c
        title  = 'du/dy'
        result = 'du/dy'
        call CutHere( title)
        write( *, 1000) XYunits, XYunits, result, Runits
c
        do 200 i=1, n
                dd = d( Vx, ddy, i, vel, ierr)
c               ignore dd and ierr -- either something was found and
c               printed or it wasn't...  dd produces its own output...
c
200     continue
c
c   === du/dx
c
        title  = 'du/dx'
        result = 'du/dx'
        call CutHere( title)
        write( *, 1000) XYunits, XYunits, result, Runits
c
        do 300 i=1, n
                dd = d( Vx, ddx, i, vel, ierr)
300     continue
c
c   === dv/dy
c
        title  = 'dv/dy'
        result = 'dv/dy'
        call CutHere( title)
        write( *, 1000) XYunits, XYunits, result, Runits
c
        do 400 i=1, n
                dd = d( Vy, ddy, i, vel, ierr)
400     continue
c
c   === dv/dx
c
        title  = 'dv/dx'
        result = 'dv/dx'
        call CutHere( title)
        write( *, 1000) XYunits, XYunits, result, Runits
c
        do 500 i=1, n
                dd = d( Vy, ddx, i, vel, ierr)
500     continue
c
```

```
c   === Reynold's stress
c
        title = 'instantaneous Reynolds stress'
        if( UnDim) then
                result = '  uv '
        else
                result = 'uv (m'
                Runits = 'm/s)^2 '
        endif
        call CutHere( title)
        write( *, 1000) XYunits, XYunits, result, Runits
c
        do 600 i=1,n
           call convert( vel(i,X), vel(i,Y), 0, vel(i,Vx)*vel(i,Vy),
     +     UnDim)
600     continue
c
        return
        end
```

```
      subroutine getline( line)
c
c     Prompting needs to be done before calling getline.
c
c     Input:
c           line initially contains the default value
c     Return value:
c           line contains the output
c
      character*80 line, inline, blank
      character*1 comment
c
      data blank/ ' '/
      data comment/'□'/
c
1000  format(' Default: ',a80)
1010  format( a80)
c
      write( *, 1000) line
c
100   continue
c     return here if the line is a comment
      read( *, 1010, end=200) inline
c
□ ifdef DEBUG
      write( *, 1020) inline
1020  format(' getline: ',a80)
□ endif
c
      if( inline(1:1).eq.comment) goto 100
      if( inline.ne.blank) then
            line=inline
c     else
c           don't change the line....(default was accepted)
      endif
      return
c
200   continue
c
c     EOF was detected
c
      line = blank
      return
      end

      real function getreal( prompt)
c
      character*80 prompt
      real value
c
      call getline( prompt)
      read( prompt, '(f10.3)') value
c
      getreal = value
      return
      end
```

```
        logical function Again( ierr)
c
c       Description:  Print out a diagnostic and figure
c       out whether the user wishes to stop the program
c       or rerun the section which just generated the error.
c
c       Author: Chuck Gendrich
c
c       History:
c       August, 1987, v1.0
c
        character*1 YorN
c
# include "vorticity.e"
c
1000    format(a80/' Do you wish to repeat the section where ',
     + 'the error occurred? [y]')
1010    format( a1)
1020    format(' Please input either ''y'' or ''n''.')
c
10      continue
        write( *, 1000) err( ierr)
        read( *, 1010,end=20) YorN
        if( YorN.ne.'y'.and.YorN.ne.'Y'.and.YorN.ne.' '.and.
     +      YorN.ne.'n'.and.YorN.ne.'N') then
c               the user goofed, can't type or something :-)
                write( *, 1020)
                goto 10
        else
                Again = ( YorN.eq.'y'.or.YorN.eq.'Y'.or.YorN.eq.' ')
                return
        endif
c
20      continue
c
c       EOF encountered
        Again = .FALSE.
        return
        end
```

```
      subroutine CutHere( title)
c
# include "vorticity.h"
c
      write( *, 1000) title
c
1000  format(/
     +' ========   0_  _'/
     +' ========    _><_ ------- cut here for: ',a40/
     +'           0'/)
c
      return
      end
```

```
      real function d( VxVy, dd, p, vel, ierr)
c
c     Description:  Take the derivative of U or V with respect to
c     x or y (as indicated by dd).  The derivative is to be taken
c     at point p in velocity descriptor array vel.
c
c     du/dy Procedure: Look down.  If a "down" point exists, see
c     what angle it's at, interpolate to find u at a point directly
c     below the original point.  This makes dx == 0.
c
c     Then calculate du/dy = (u1-u2) / (y1-y2) ◇ the point (x0, avg(y))
c
c     du/dx Procedure: Similar to du/dy, but swap X's and Y's
c
c     dv/dy Procedure: Identical to du/dy, but use vel( Vy) instead.
c     This should be taken care of by the routine being called with
c     VxVy set to either Vx or Vy (as appropriate).
c
c     Author: Chuck Genarich
c
c     History:
c     August, 1987 v1.0
c
cccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c# define DEBUG
c# define UGHBUG  <--- only if you want the really gruesome diagnostics
# define STRICT
c        only use values corrected for dx/dy (only use interpolated
c        values)
# include "vorticity.h"
c
      integer VxVy, dd
      real delta
      parameter( delta = 30.0)
c
      real x0, x2, dx
      real y0, y2, dy
      real angle
c
      integer look
      real interp
      external look, interp
c
      logical UnDim
c             -- from convert... not used here ---
# ifdef UGHBUG
      n = NumPts( vel, 4)
      write( *, 1000) (i,vel(i,X),vel(i,Y),vel(i,Vx),vel(i,Vy),i=1,n)
1000  format(/' Here are the velocity components and their',
     +' locations:'/'          X       Y       Vx      Vy'/
     + (1x,i2,':  ',4(1x,f7.2)))
# endif
c
cccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
```

```
      ierr = 0
      if( dd.eq.ddy) then
         j = look( p,vel,4, DOWN, delta, matchs, ierr)
#        ifdef DEBUG
         write( *, 1010) 'down', p, j
c*       write( *, 1010) 'down', p, j, matchs
#        endif
      else
         j = look( p,vel,4, RIGHT, delta, matchs, ierr)
#        ifdef DEBUG
         write( *, 1010) 'right', p, j
c*       write( *, 1010) 'right', p, j, matchs
#        endif
      endif
1010  format(' ''look''d ',a5,' from ',i2,' and found ',i2'.')
c*   + /' Here''s the match array:',20(1x,i2))
c
c     error check point j
c
      if( j.eq.BAD.or.ierr.ne.0) then
            d = BAD
            if( ierr.ne.0) then
                  return
            else
                  ierr = 21
                  return
            endif
      endif
c
      x0 = vel( p,X)
      x2 = vel( j,X)
      dx = x2 - x0
c
      y0 = vel( p,Y)
      y2 = vel( j,Y)
      dy = y2 - y0
c
      angle = tanl( dy, dx)
      if( dd.eq.ddy) then
         if( angle.gt.DOWN) then
               k = look( j,vel,4,LEFT, delta, matchs,ierr)
         else
               k = look( j,vel,4,RIGHT, delta, matchs,ierr)
         endif
      else
         if( angle.gt.delta) then
c              it's in the 4th quadrant.
               k = look( j,vel,4,UP, delta, matchs,ierr)
         else
               k = look( j,vel,4,DOWN, delta, matchs,ierr)
         endif
      endif
c
c     interpolate if we found a point to interpolate with...
c
      if( k.ne.BAD.and.ierr.eq.0) then
```

```
c           there is a point to interpolate to
c           ifdef DEBUG
            write( *, 1020) j,k
1020        format(' Going to interpolate between points ',i2,
     +      ' and ',i2,'.')
c           endif
            if( dd.eq.ddy) then
               u=interp( x2, vel(j,VxVy), vel(k,X), vel(k,VxVy), x0,ierr)
            else
               u=interp( y2, vel(j,VxVy), vel(k,Y), vel(k,VxVy), y0,ierr)
            endif
         else
c           use the lower/right velocity and ignore dx/dy... eeek!
            u  = vel( j,VxVy)
         endif
c
c        ifdef STRICT
         if( k.eq.BAD.or.ierr.ne.0) then
                return
         endif
c        endif
c
         du = u - vel( p,VxVy)
         if( dd.eq.ddy) then
            d = du/dy
            call convert( x0, avg( y0,y2), du/dy, 0, UnDim)
         else
            d = du/dx
            call convert( avg( x0, x2), y0, du/dx, 0, UnDim)
         endif
         return
1040     format( 3(2x,f8.3))
         end
```

```
      subroutine convert( xpix, ypix, ddpix, uvpix, UnDim)
c
c     Description:  print out the conversion to "real units"
c     of the above values.  Only print out the ones which
c     aren't zero (generally either ddpix or uvpix)...
c
c     Output: all output will be written to stdout.  If you want it in
c     a file, put a tee on the process when you run it.
c        e.g.:   lori 38>  vorticity | tee output_file
c
c     Procedure: First get conversion factors and the time between
c     each frame.  Since this subroutine is "saved", these values
c     only have to be obtained once, then they're applied to every
c     subsequent value as appropriate...
c
c     Caveats: No values are printed out the first time through.
c
c     Author: Chuck Gendrich
c
c     History:
c     August, 1987 v1.0
c
cccccccccIccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c  define DEBUG
c
      logical UnDim
c             true if values should be non-dimensionalized
c
      real xpix, ypix, ddpix, uvpix
c             xpix and ypix are (X,Y) in pixel values
c             ddpix is a spatial derivative of some velocity
c                     (e.g. du/dx)
c             uvpix is the product of two velocities
c                     (e.g. Vx*Vy)
      real xmm, ymm, ddmm, uvmm
c             x, y, dd, and uv converted to "real units" or non-
c             dimensionalized (local copies of the numbers so that
c             we don't accidentally try to set some constant equal
c             to something else....)
c
      real MmPixel, dt, Mmdt, Mmdt2, xoff, yoff
c
      real Uw, nu, t, xyfact, ddfact, uvfact
c             Uw -- wall velocity
c             nu -- kinematic viscosity
c              t -- total elapsed time
c         xyfact -- non-dimensionalizing factor for x's and y's
c         ddfact -- non-dimensionalizing factor for dd's
c         uvfact -- non-dimensionalizing factor for uv's
c
      real getreal
      external getreal
c
      character*80 line
c
```

```
      save
c
      data MmPixel, dt/ 2*0.0/
c
1000  format(' Please enter the multiplication factor to convert'/
     +' pixels to mm.  (pixel_value * factor) = (mm_value)')
1020  format(' Please enter the time between frames (dt).')
1030  format(' Please enter the Y offset for frame 1.'/
     +' ( Y(frame1) - Yoffset) = Y (abs dist to the wall)')
1040  format(' Would you like the values to be non-dimensionalized?',
     +' [n]')
1050  format(a80)
1060  format(' Please enter the wall velocity, Uw.')
1070  format(' Please enter the kinematic viscosity, nu.')
1080  format(' Please enter the total elapsed time, t.')
1090  format(/' convert: MmPixel: ',f10.9,' dt: ',f10.9,'  Yoff: ',
     +  f10.7)
1100  format(' convert: non-dimensionalizing.  xyfact: ',e9.4,
     +'  udfact: ',e9.4,'  uvfact: ',e9.4/)
1110  format(' convert: printing values with real units.'/)
1120  format(/' convert: Pixel values --  (',f8.2,',',f8.2,')'/
     +' spat. deriv: ',f12.3,' and  uv value: ',f12.3/)
1130  format(4(2x,f11.3))
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c     Get the conversion to mm's and the time between frames if necessary
c
      if(MmPixel.ne.0.and.dt.ne.0) goto 100
c     neither of these values is permitted to be zero...
c
10    continue
c
      write( *, 1000)
      line    = '0.029520809 mm/pixel'
c        0.0277 mm/pixel when .50" = 458.13 pixels
      MmPixel = getreal( line)
c
      write( *, 1020)
      line = '0.02000000 sec'
      dt   = getreal( line)
c
      Mmdt = MmPixel/dt
      Mmdt2 = Mmdt * Mmdt
c
      write( *, 1030)
      line = '560.0      Pixels'
      Yoff = getreal( line)
c
      Xoff = 0.0
c
20    continue
c
c     Get non-dimensionalizing constants (or set them to 1.0)
c
      write( *, 1040)
```

```
      read( *, 1050) line
      if( line(1:1).eq.'Y'.or.line(1:1).eq.'y') then
            UnDim = .TRUE.
      else if( line(1:1).eq.'N'.or.line(1:1).eq.'n'.or.
     +  line(1:1).eq.' ') then
            UnDim = .FALSE.
      else
            goto 20
      endif
c
      if( UnDim) then
            write( *, 1060)
            line = '5.0         in./sec'
            Uw   = getreal( line)
c
            write( *, 1070)
            line = '0.00310460 in^2/sec.'
            nu   = getreal( line)
c
            write( *, 1080)
            line = '5.25        sec.'
            t    = getreal( line)
c
            xyfact = 1.0/( 2.0 * 25.4 * sqrt( nu * t))
            ddfact = 4.0 * sqrt( nu * t) / Uw
            uvfact = 1.0
c                   don't know how to non-dimensionalize this one
      else
            xyfact = 1.0
            ddfact = 1.0
            uvfact = 1.0
      endif
c
#     ifdef DEBUG
      write( *, 1090) MmPixel, dt, Yoff
      if( UnDim) then
            write( *, 1100) xyfact, ddfact, uvfact
      else
            write( *, 1110)
      endif
#     endif
c
      if(MmPixel.eq.0.or.dt.eq.0) goto 10
      return
c           don't print anything out the first time through
c
100   continue

#     ifdef DEBUG
      write( *, 1120) xpix, ypix, ddpix, uvpix
#     endif
c
      xmm  = (xpix - Xoff) * MmPixel * xyfact
      ymm  = (ypix - Yoff) * MmPixel * xyfact
c
      ddmm = ddpix / dt * ddfact
```

```
        uvmm = uvpix * Mmdt2 * uvfact
c
        if(      ddmm.ne.0.and.uvmm.ne.0) then
                write( *, 1130) xmm, ymm, ddmm, uvmm
        else if( ddmm.ne.0.and.uvmm.eq.0) then
                write( *, 1130) xmm, ymm, ddmm
        else if( ddmm.eq.0.and.uvmm.ne.0) then
                write( *, 1130) xmm, ymm, uvmm
        else
                write( *, 1130) xmm, ymm
        endif
c
        return
        end
```

```
      subroutine integrt( p1, p2, vel, area, intgrl)
c
c     Description: Integrating from point p1 to point p2 (whose X- and
c     Y-coordinates are described in vel), return the area under the
c     curve and the value of V * ds.  Simple trigonometric and
c     calculus identities are used;  e.g., the area between a line and
c     the x-axis is 1/2 (y1 + y2) (x2 - x1) and
c
c     _  _
c     V*ds = |V|*|s|*cos( the angle between the two vectors)
c
c     Temporary variables are used to store intermediate results so
c     that the steps are clear.
c
c     Author: Chuck Gendrich
c
c     History:
c     August, 1987 v1.0
c
c23456789012345678901234567890123456789012345678901234567890123456789012
c
#define DEBUG
#include "vorticity.h"
c
      integer p1, p2
      real area, intgrl
c
      real avg, dist, tanl
      external avg, dist, tanl
c
      real Ubar, Vbar, V, s, thetaV, thetaS
c           Ubar is the average Vx for a side
c           Vbar is the average Vy for a side
c             V  is the magnitude of the velocity vector
c             s  is the length of a side
c        thetaV is the angle V makes with the +x axis
c        thetaS is the angle S makes with the +x axis
c
      real pi180
      parameter( pi180 = 0.017453293)
c           pi/180 for converting degrees to radians
c
      if( p1.eq.0.or.p2.eq.0) then
c           we're integrating to or from a non-existent point
            area  = 0.0
            intgrl = 0.0
            return
      endif

      x1 = vel( p1, X)
      y1 = vel( p1, Y)
c
      x2 = vel( p2, X)
      y2 = vel( p2, Y)
c
      dx = x2 - x1
      dy = y2 - y1
```

```
c
        Ubar = vel( p1, Vx)
        Vbar = vel( p1, Vy)
c       Ubar = avg( vel(p1,Vx), vel(p2,Vx))
c       Vbar = avg( vel(p1,Vy), vel(p2,Vy))
c
        V = sqrt( Ubar*Ubar + Vbar*Vbar)
        s = sqrt( dx*dx + dy*dy)
c
        thetaV = tan1( Vbar, Ubar)
        thetaS = tan1( dy, dx)
c
        intgrl = V * s * cos( (thetaV - thetaS) * pi180)
        area   = 0.5 * (y1 + y2) * dx
c
#ifdef DEBUG
        write( *, 1000) p1, p2, dx, dy, Ubar, Vbar, V, s, thetaV, thetaS
1000    format(/' intgrt: from ',i2,' to ',i2,'.  dx: ',f8.3,' dy: ',
     +  f8.3/' Ubar: ',f8.3,' Vbar: ',f8.3,'  -> V: ',f8.3/
     +  's: ',f8.3,' thetaV: ',f9.3,' and thetaS: ',f9.3)
        write(*, 1010) intgrl, area
1010    format('    line intgrl: ',f12.3,'   area: ',e14.3/)
#endif
c
        return
        end
```

```
        subroutine vector( dy, dx, scale)
c
c       Description:  Draw a 2D vector with arrow head from the current
c       point through the final point.  The length of the vector will be
c       equal to its original length times "scale."
c
c       The arrow head will always be the same size (more or less).
c
c       Current values for linewidth, color, etc. are used.
c
c       Author:  Chuck Gendrich <cpg>
c                ("You mean I can't just find this in some library?!◊□")
c
c       History:
c               12 dec 87 -- original inception and coding
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
        real dy, dx, scale
c
        real SideLen, dAng
        parameter( SideLen = 8.0)
        parameter( dAng    = 20.0)
c               the length of each side of the arrow (in pixels)
c               the angle that each side makes with the vector
c
        real x1, y1, x2, y2, x3, y3, a2, a3
c               destination points for the far end of the vector and the
c               two ends of the arrow head...
c
        real angle
c               the angle that the vector makes with the X-axis
c
        real tan1
        external tan1
        real pi
        parameter( pi = 3.141592653589)
c
c       Figure the angles that the lines will be at WRT the X-axis
c
        angle = tan1( dy, dx)
        a2    = (angle - dAng) * pi / 180.0
        a3    = (angle + dAng) * pi / 180.0
c
c       Figure the relative moves
c
        x1 = scale * dx
        y1 = scale * dy
        x2 = -SideLen * cos( a2)
        y2 = -SideLen * sin( a2)
        x3 = -SideLen * cos( a3)
        y3 = -SideLen * sin( a3)
c
c       Draw it
c
        call rpr2( x1, y1)
```

```
        call rdr2( x2, y2)
        call rmv2(-x2,-y2)
        call rdr2( x3, y3)
c
c       Put the cursor back where we started
c
        call rmv2(-x3,-y3)
        call rmv2(-x1,-y1)
c
        return
        end
```

```
        logical function GoOn()

        character*1 char
        logical first, slow
        save

        data first/.TRUE./

        if( first) then
                first = .FALSE.
                print*,'Slow mode? [y]'
                read( *,'(A1)') char
                if( char.eq.'n'.or.char.eq.'N') then
                        slow = .FALSE.
                else
                        slow = .TRUE.
                endif
        endif
        if( SLOW) then
                print*,'<RETURN> to continue...'
                read( *,'(A1)') char
        endif
        GoOn = .TRUE.
        return
        end
```

```
#
#       makefile for the vorticity determination program (vorticity)
#       and other miscellaneous test and error-analysis programs
#
LIBS=
DESTDIR=/usr/thor/gendrich/bin/iris/

OBJS1=  getline.o getframe.o errors.o sort.o match_points.o count_points.o \
        velocities.o make_poly.o geometry.o look.o main.o fluids.o \
        cut_here.o ddx.o convert.o integrate.o vector.o GoOn.o
OBJS2=  TestLook.o getline.o getframe.o count_points.o geometry.o sort.o \
        convert.o GoOn.o
OBJS3=  TestMatch.o getline.o getframe.o errors.o look.o count_points.o \
        geometry.o sort.o convert.o vector.o GoOn.o
OBJS4=  TestDdx.o getline.o getframe.o errors.o sort.o match_points.o \
        count_points.o velocities.o geometry.o look.o ddx.o convert.o \
        vector.o GoOn.o
OBJS5=  getline.o getframe.o errors.o sort.o match_points.o count_points.o \
        velocities.o one_poly.o geometry.o look.o TestPoly.o fluids.o \
        cut_here.o ddx.o convert.o integrate.o vector.o GoOn.o
OBJS6=  Errors.o stats.o vector.o sort.o count_points.o match_points.o \
        getframe.o getline.o geometry.o look.o errors.o convert.o
OBJS=   $(OBJS1) $(OBJS2) $(OBJS3) $(OBJS4) one_poly.o $(OBJS6)

SRCS1=  main.f getframe.f count_points.f sort.f match_points.f velocities.f \
        geometry.f make_poly.f look.f fluids.f getline.f errors.f cut_here.f \
        ddx.f convert.f integrate.f vector.f GoOn.f
SRCS2=  TestLook.f getline.f getframe.f count_points.f geometry.f sort.f
SRCS3=  TestMatch.f look.f getframe.f errors.f getline.f count_points.f
SRCS4=  TestDdx.f ddx.f getframe.f count_points.f sort.f match_points.f \
        velocities.f geometry.f look.f convert.f getline.f errors.f
SRCS6=  Errors.f stats.f

ALL=    vorticity TestLook TestMatch TestDdx TestPoly Errors

O=
# can't optimize FORTRAN code.  bummer.
DBG=
#DBG= -g -DDEBUG
COPTS=  -Zf -Zg
CFLAGS= $O $(COPTS) $(CCONFIG) $(DBG)

CC=     f77
INSTALL=mv -f
CHMOD=  chmod
OBJMODE=755

.f.o:
        $(CC) $(CFLAGS) -c -o $*.o $*.f

vorticity: $(OBJS1)
        $(CC) $(COPTS) -o vorticity $(OBJS1) $(LIBS)
        $(CHMOD) $(OBJMODE) vorticity

TestLook:       $(OBJS2)
        $(CC) $(COPTS) -o TestLook $(OBJS2) $(LIBS)
```

```
	$(CHMOD) $(OBJMODE) TestLook

TestMatch:	$(OBJS3)
	$(CC) $(COPTS) -o TestMatch $(OBJS3) $(LIBS)
	$(CHMOD) $(OBJMODE) TestMatch

TestDdx:	$(OBJS4)
	$(CC) $(COPTS) -o TestDdx $(OBJS4) $(LIBS)
	$(CHMOD) $(OBJMODE) TestDdx

TestPoly:	$(OBJS5)
	$(CC) $(COPTS) -o TestPoly $(OBJS5) $(LIBS)
	$(CHMOD) $(OBJMODE) TestPoly

Errors:		$(OBJS6)
	$(CC) $(COPTS) -o Errors $(OBJS6) $(LIBS)
	$(CHMOD) $(OBJMODE) Errors

install: vorticity TestPoly Errors
	$(INSTALL) vorticity $(DESTDIR)
	$(INSTALL) TestPoly $(DESTDIR)
	$(INSTALL) Errors $(DESTDIR)

all: $(ALL)

print:
	pr $(SRCS1) Makefile | rsh freyja lp

clean:
	rm -f core junk $(ALL) $(OBJS)
	mv vor.* Results

TestLook.o:	TestLook.f vorticity.h graphics.h
TestMatch.o:	TestMatch.f vorticity.h graphics.h
TestDdx.o:	TestDdx.f vorticity.h graphics.h
TestPoly.o:	TestPoly.f vorticity.h graphics.h Makefile
main.o:		main.f vorticity.h graphics.h Makefile
Errors.o:	Errors.f stats.h
stats.o:	stats.f stats.h
convert.o:	convert.f Makefile
count_points.o: count_points.f vorticity.h
cut_here.o:	cut_here.f vorticity.h
ddx.o:		ddx.f vorticity.h Makefile
errors.o:	errors.f vorticity.e
fluids.o:	fluids.f vorticity.h
geometry.o:	geometry.f vorticity.h Makefile
getline.o:	getline.f Makefile
getframe.o:	getframe.f vorticity.h
integrate.o:	integrate.f vorticity.h Makefile
look.o:		look.f vorticity.h
make_poly.o:	make_poly.f vorticity.h
one_poly.o:	one_poly.f vorticity.h
match_points.o: match_points.f vorticity.h
sort.o:		sort.f vorticity.h
velocities.o:	velocities.f vorticity.h
```

APPENDIX B

A Reprint from the

PROCEEDINGS

Of SPIE-The International Society for Optical Engineering

Volume 814

International Conference on
Photomechanics and Speckle Metrology

17-20 August 1987
San Diego, California

Measurement of two-dimensional fluid dynamic quantities using a photochromic grid tracing technique

R. E. Falco, C. C. Chu
Turbulence Structure Laboratory, Department of Mechanical Engineering
Michigan State University

MEASUREMENT OF TWO-DIMENSIONAL FLUID DYNAMIC QUANTITIES USING A PHOTOCHROMIC GRID TRACING TECHNIQUE

R. E. Falco and C. C. Chu

Turbulence Structure Laboratory, Department of Mechanical Engineering, Michigan State University

ABSTRACT

A non-intrusive photochemical phenomena called photochromism is used to obtain important fluid quantities such as velocity, and vorticity over a two-dimensional domain of a fluid flow. Using a uv laser, and a novel beam splitting and steering technique, a grid of lines is impressed in the prepared fluid. Two successive pictures are needed to calculate the fluid dynamic quantities. Measurements in a Stokes' layer are compared against exact solutions, and measurement of the circulation of a vortex ring is presented. The technique will be useful in turbulent and unsteady flows.

1. INTRODUCTION

There is an increasing need to obtain flow field information over a region of flow simultaneously. Examples include the study of mixing in internal combustion engines, and the understanding of the importance of coherent motions in many turbulent and unsteady flows. Furthermore, there is a need to obtain more detailed information about these and many simpler flows, including vorticity and strain-rate measurements. Existing probe techniques are at best able to provide single point estimates of these quantities in flows with steady mean velocities.

Hummel and his co-workers first exploited the phenomena of photochromism[1,2]. Using a single line of uv radiation for excitation, he and his co-workers were able to measure one component of velocity in flows with predominately one direction of motion. With the advent of high power uv lasers and new chemicals it is increasingly possible to apply the technique to more complicated flows. We have generalized the technique to enable it to uniquely tag specific points in the flow on a specified grid. As a result we have a non-intrusive photo-optical technique which can be used to obtain two velocity components and the spatial gradients of these components, at a large number of points in a flow field.

2. USE OF PHOTOCHROMISM TO MEASURE FLUID DYNAMIC QUANTITIES

The phenomena of photochromism has been defined by Brown[3] as "a reversible change of a single chemical species between two states having distinguishably different absorption spectra, such change being induced in at least one direction by the action of electromagnetic radiation." The chemical used in this study was Kodak 1,3-Trimethyl-8-nitrospiro[2-H-1-benzopyran-2,2'-indoline]. We excited it at 351nm using pulses of 20ns produced by a Tachisto Excimer laser running on XeF (approximately 100mj output). Dissolved in the working fluid, which was kerosene, we found that the chemical, thus excited, reflected blue light for the order of tens of seconds. Thus, the fluid was tagged long enough for it to move appreciably. In practice, for the velocities of our experiments, we needed to follow the lines only a fraction of a second to obtain the information required.

2.1 Use of a grid to tag fluid particles

To enable us to measure two components of the velocity in a flow using the visualization, we must tag and follow specific fluid particles. The simplest way to do this is have two lines intersect. That intersection defines a point in the flow which, because it is colored, we can follow. To measure a spatial gradient, we must simultaneously mark two points in the flow. To enable us to measure a spatial gradient in a prescribed direction, we need several points and must use interpolation. Thus, creation of a grid of intersecting marked fluid lines would enable both velocities and gradients to be measured.

2.2 The beam divider

We sought a method of dividing a beam of laser light into 'n' beams where 'n' would be 5-25, using a static device. In doing so, we sought to lose as little light as possible. Thus, we designed a divider that appears as an oversized blazed reflection grating; the steps size 'a' is approximately 2mm. The incident laser energy was directed at a shallow angle to the plane of the grating. Because the facets are so much larger than a wavelength of light, we expected to primarily see specular reflection. The separation of the specularly reflected portions of the beam were determined by the difference of the angle of incidence and the blaze angle, and the grating step width and spacing as shown in Fig. 1. A photo of the actual beam divider is shown in Fig. 2. It was fabricated by individually fixing Aluminum coated mirrors (with the coating optimized for shallow incidence angles) to a machined base. Apart from diffraction effects, no incidence laser energy would be lost.

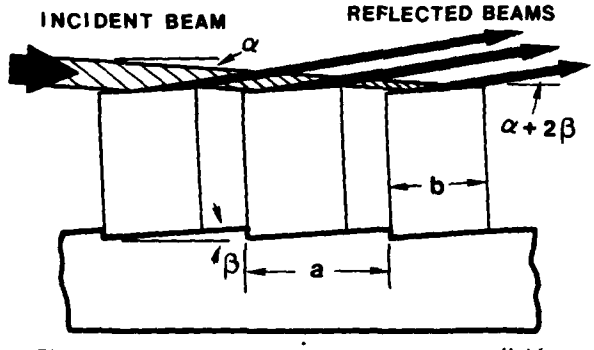


Fig. 1. The specular reflection from the beam divider.

As expected, however, diffraction effects were exhibited. A photo of the output of the beam divider using a Helium Neon laser is shown in Fig. 3. We see the specularly reflected beams, and the superimposed diffraction pattern. The diffraction bands are spaced as expected from the grating equation[4]. The appearance of the first and higher order bands are a source of 'noise' in the present grid making scheme. When experiments were performed with the Excimer laser at

351 nm, the diffraction pattern was, for reasons we do not understand, not observed. If it was present, results showed that it was too weak to excite the photochromic chemical (see Fig. 5a and 5b).

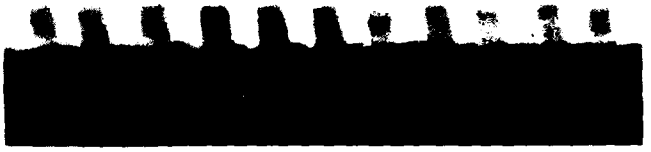


Fig. 2. The beam divider

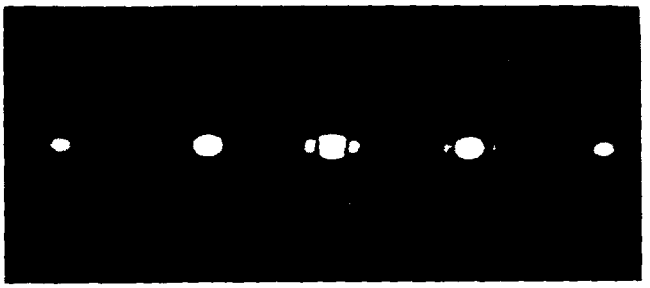


Fig. 3. A photo of the output of the beam divider.

Fig. 4 shows a sketch of the optical set-up for making the grid shown in Fig. 5a. Note that the lines in Fig. 5a remain uniform and show little divergence. The mesh size is the order of 1mm. The two photos in Fig. 5 are 0.4 seconds apart, taken by a still camera with a motor drive. From the distortion of the grid in Fig. 5b with respect to that in Fig. 5a, (one laser pulse, two photos Δt apart) we can calculate all the kinematic quantities of the flow in the plane of the film. (Note, that in practice we would not use a second image which has distorted as much as Fig. 5b, but we present it to illustrate the ideas involved.)

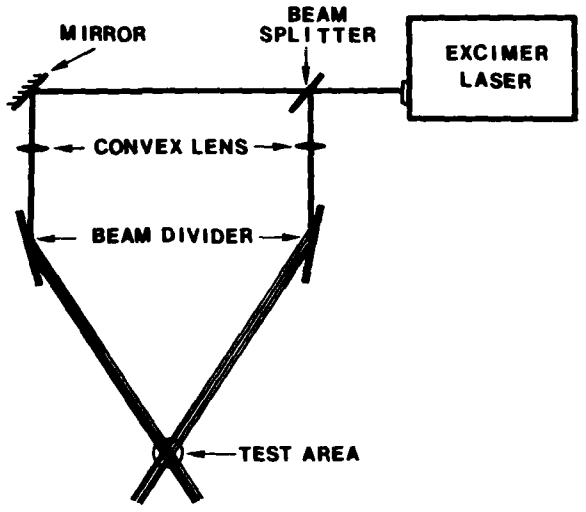


Fig. 4. The optical setup used to produce a grid of laser lines.

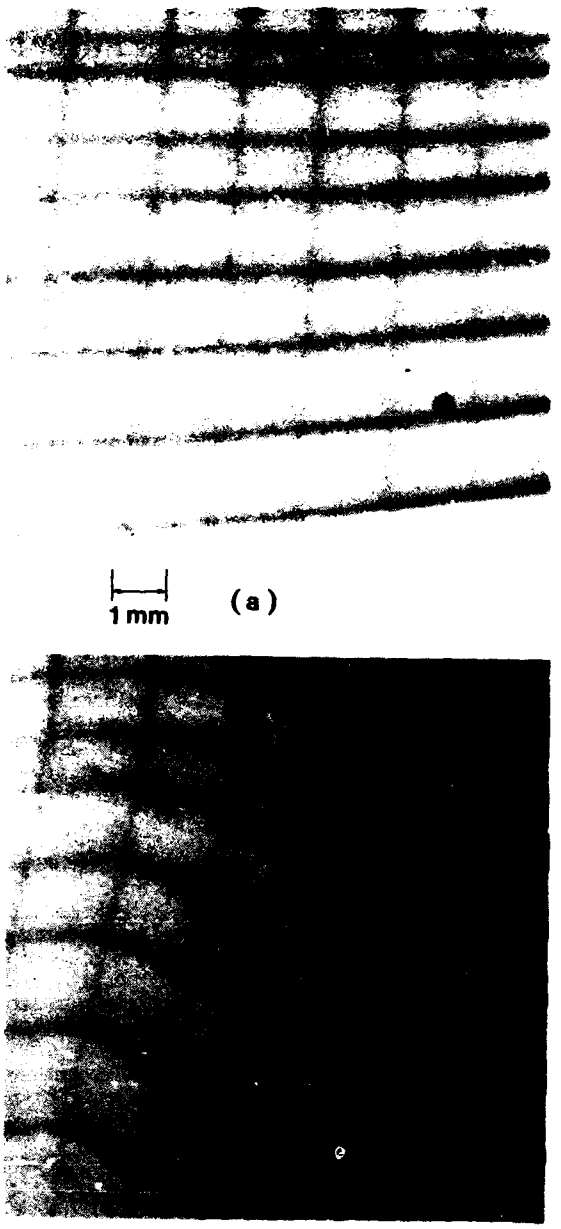


Fig. 5. An example of the grid. a) shortly after the laser pulse, b) .4sec later.

2.3 Algorithms to obtain fluid dynamic quantities

For convenience, we pick up one 'grid box' in Fig. 5a which, if the mesh size is small enough, can be thought of as a text book fluid 'particle' in flow at time t = 0. This fluid particle is shown using solid lines in Fig. 6a. As this particle moves with the flow, it may undergo several motions such as translation, rotation and deformation. The dashed lines in Fig. 6b show the same fluid particle after a short time interval, Δt. Because we know the history of this specific fluid particle, we can compute the velocity of each corner by taking the time derivative of displacement ds associated with each point, that is, $ds/\Delta t$. The deduction procedures are shown in Fig. 6b.

Note that V is designated as the velocity at point 1 which is then positioned at the midpoint between point 1' and point 1". Fig. 7 shows the velocity vectors deduced from Fig. 6. Actually, a fluid particle moving in a general three-dimensional flow field may have motions about all three coordinate axes, however, we limit the discussion to the projection of this motion onto the plane of the photos which is parallel to the initial plane of the grid.

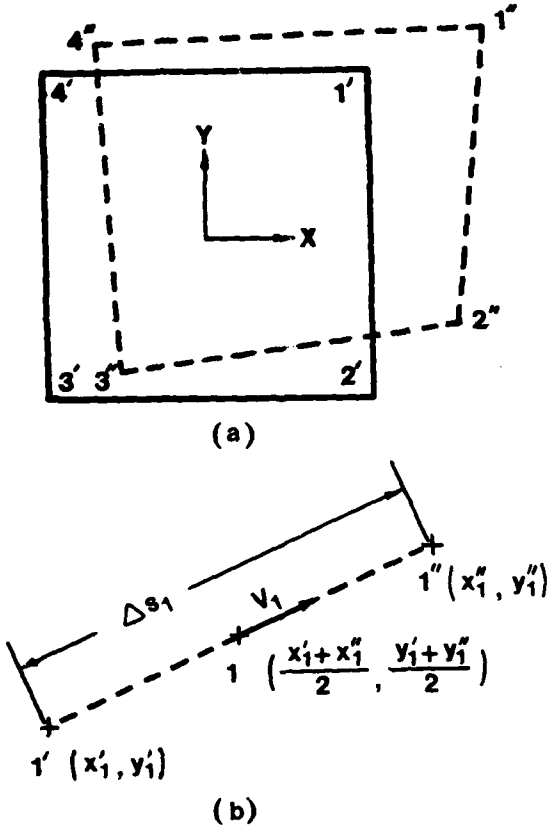


Fig. 6. Procedure for conversion of the displacement of a grid box to velocities.

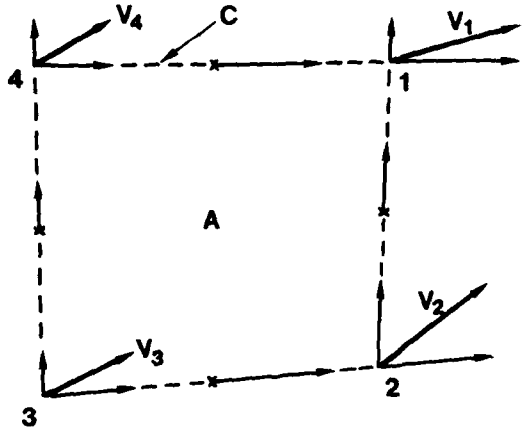


Fig. 7. Decomposition of the velocities used to calculate the circulation around a grid box.

To obtain velocity gradients, vorticity, and strain rates from the velocities, we must difference across the distance of a grid box. Thus, for a given flow the grid must be small enough to resolve the smallest scales of motion.

An alternative method can be used if we are only interested in the vorticity. Using the definition of the circulation Γ,

$$\Gamma = \int_C \mathbf{V}\cdot d\mathbf{s} \qquad (1)$$

and Gauss' theorem to relate the surface integral to an area integral,

$$\Gamma = \int_A \omega\cdot \mathbf{n} dA$$

we can obtain the vorticity component normal to the film plane, ω_z, from a grid box of arbitrary shape, with circumference C and area A.

The velocity component $\mathbf{V}\cdot d\mathbf{s}$ in Eq. (1) is estimated by forming the average of the corner velocity components (see Fig. 7). Dividing Γ by the area of the fluid particle results in the average vorticity at the centroid of this fluid particle. This technique has the advantages of avoiding a second differencing of the experimental data.

Thus, by following the distortions of a single grid box we can obtain the velocities, circulation, vorticity and even the Reynolds stresses and strain rates if desired, over that small area. By doing this for the other grid mesh points, we can obtain the important fluid dynamic quantities at many locations over a two-dimensional field in a fluid flow simultaneously.

2.4 Error analysis

Consider the grid box in Fig. 6. We will assume that the grid mesh is spaced ϵ and that a worse case line movement between images is 1/2 the mesh width. After time Δt standard analysis[5] shows that the error in estimating the vorticity ω_z is

$$\delta\omega_z = (1/\Delta t)(\delta\epsilon/\epsilon)$$

to first order in ϵ.

As an example of the designed accuracy of our experiments, if we produce 100μm lines on a grid mesh of 1mm, and our error in reading the center of the lines is 10%, we obtain $\delta\epsilon/\epsilon = .01$. For $\Delta t = .01$sec, the error in the measurement of vorticity is 1/sec. This is better than has been possible with hot-wire probes, and is comparable with the most sophisticated single point laser doppler measurements.[6]

3. EXPERIMENTAL RESULTS

Two experiments were performed to demonstrate the technique. Both used deodorized kerosene with 10 ppm of the photochromic chemical dissolved in it. The data were recorded on Kodak Tmax 400 ASA film through a Photo-Sonics 35mm high speed movie camera, framing between 150 and 200 frames per second. The films were analyzed on a NEC film analyzer which is accurate to .05mm.

3.1 Stokes' layer flow

We can test the accuracy of the technique in a flow for which an exact solution of the Navier-Stokes equations is obtainable. The flow is that of an impulsively started infinite plate in an initially still infinite ambient. The problem involves the two-dimensional diffusion of vorticity generated at the plate into the ambient. The thickness of the boundary layer, δ, is a function of the kinematic viscosity, ν, and the time after the start of the plate's motion. We have approximated this motion using a belt in a tank which can be rapidly started. Fig. 8 shows the experimental setup. The width of the belt is 17.8 cm, and the distance between the rollers is 152.4 cm, with the test position 90 cm downstream of the lead roller. All experimental data was obtained before the leading edge effects reached the test position. Furthermore, the aspect ratio is sufficient to prevent the disturbances generated in the corners from reaching the center of the belt at the test section in the test time interval. Therefore, the wall layer flow measured was two-dimensional. The speed of the belt was adjustable from 2.5 to 22.9 cm/sec. It reaches a constant speed within a second. This short acceleration reasonably approximates an impulsive start. A grid mesh size of $.15\delta$ was used, with the laser lines intersecting as in Fig. 8.

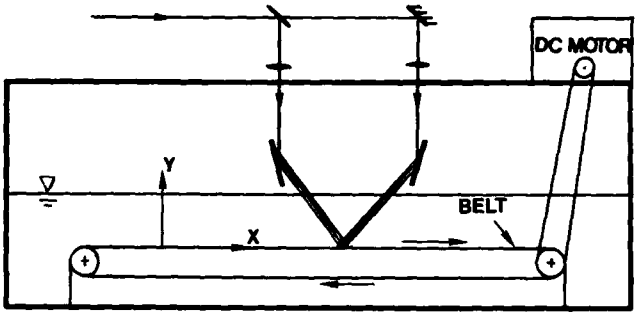


Fig. 8. Setup used to create and measure the Stokes' layer flow.

Results of the velocity gradient obtained from double differentiation in the Stokes layer are shown in Fig. 9, where they are compared with the exact solution. Everything is non-dimensionalized by the similarity variables of the exact solution. Because the flow is two-dimensional, incompressible, and doesn't change with the streamwise coordinate, x, the continuity equation reduces to $\partial v/\partial y = 0$, and a check on this is also shown in Fig. 9 (v is the velocity component normal to the wall, and y is the coordinate normal to the wall). We see that both the error in the estimates of the continuity equation, and in the velocity gradient distribution are of similar magnitude. The absolute value of the error is ± 2/sec, consistent with the design error analysis. The deviation of the experimental profile in the near wall region is a consequence of the grid mesh being too large to adequately resolve the large value of the gradient $\partial u/\partial y$. On the whole the results are very encouraging.

3.2 Vortex ring flow

The real value of the technique is its ability to measure flows that are not measurable, analytically solvable or numerically calculable. The vortex ring is a common place example. It represents a class of flow fields that are of particular importance in turbulent and unsteady flows.

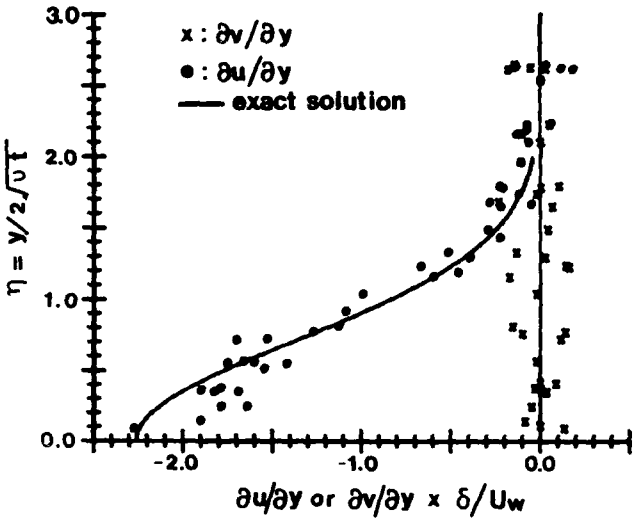


Fig. 9. Experimental measurements of the velocity gradient in a Stokes' layer compared with the theoretical solution. A check of the continuity equation $\partial v/\partial y = 0$ is also shown.

The experimental setup used to generate the vortex ring is shown in Fig. 10. A small amount of fluid from a constant head reservoir was emitted from a 1.9 cm orifice to create the rings. A solenoid valve whose opening time could be adjusted controlled the outflow. The kerosene in the reservoir was dyed with chlorophyll to partially mark the ring. The convection velocity of the rings was measured to be 3.8 cm/sec at the test section which is located 7.6 cm downstream of the orifice. The Reynolds number of the ring is approximately 700, based on the ring's core diameter and its convection velocity. The flow is axisymmetric, so that we only needed to align our grid through an axis of symmetry and measure the flow in half of the ring to measure its circulation.

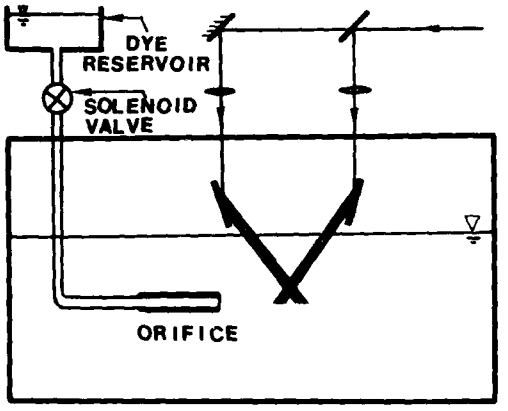


Fig. 10. Setup used to create and measure the circulation of a vortex ring.

The instantaneous circulation distribution of the ring is shown in Fig. 11. Although no solution exists for the experimentally created ring, we show a comparison with Hill's[7] exact solution for a ring in which the vorticity occupies a sphere. The core of our ring is smaller than that of the hypothetical Hill's ring, the overall shape is an oblate spheroid, and it is unlikely that the vorticity distribution is similar, but, as the data shows, the circulation of the ring (Reynolds number = 700) is not very different.

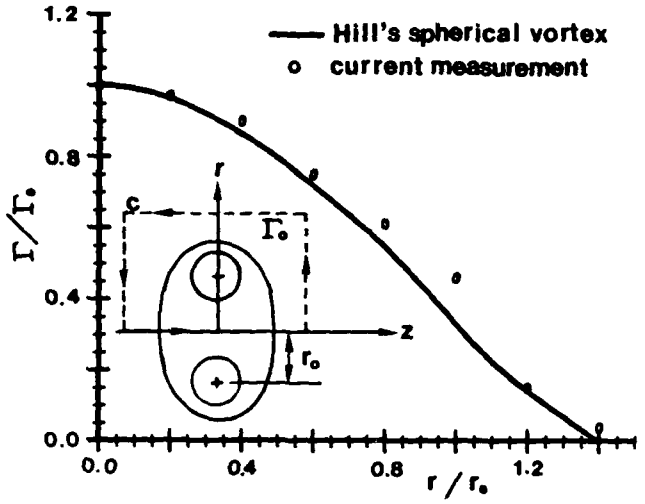


Fig. 11. The circulation distribution of a laminar vortex ring (Reynolds number = 700), compared with Hill's solution for a spherical vortex.

4. DISCUSSION

The demonstrated accuracy of the technique is more than adequate for many fluid dynamic problems. Since, in principle, the calibration depends only on a time and a length scale, there is further room for improvement. In practice, determining the intersection points of the grid lines is limited by the grain size of the film used, and the distortion of the developing process. Improvements in the grain limitation can be made by using a large format still camera with a shuttering mechanism that will enable a double exposure of the grid to be made on a plate. Furthermore, image processing techniques can be used to increase the accuracy and repeatability of the results. We are currently pursuing these avenues.

The technique is not restricted to slowly varying flows as are so many flow visualization techniques. Furthermore it is not restricted to Newtonian fluids. We feel that the whole field approach can supplement LDA measurements where velocity fields are required, as well as going beyond to make Reynolds stress, vorticity, and strain rate measurements where needed.

Since the photochromic grid following technique requires only a single pulse of uv light to create the grid in a suitably mixed solution, the technique can be used in many labs which possess single pulse or low pulse rate lasers.

Current limitations include to need to work in organic liquids, required to dissolve the chemicals, and available penetration of the laser beam into the fluid. This problem, addressed in Ref. 2 is a function of both the laser power and the chemical concentration. It can be alleviated somewhat by scanning the laser beam across the 'steps' in the beam divider, thereby putting the full energy of the laser in every line. Since the excitation takes place in picoseconds, during a 20 nanosecond pulse width, the scanning (through minutes of a degree) can be accomplished with off the self equipment, and each line in the fluid will be proportionally more intense, or the chemical concentration reduced to enable laser penetration further into the fluid.

5. CONCLUSIONS

A new technique for obtaining velocity, and velocity gradient data over a two-dimensional domain of a fluid flow is presented. Calibration depends only upon a time and a length scale. Comparison of measurements with the exact solution in a Stokes' flow indicates that its accuracy can be predicted by classical analysis, and is comparable or better than that achievable with the best single point probe techniques. The circulation of a vortex ring has been measured.

6. ACKNOWLEDGMENTS

We greatfully acknowledge support of the Air Force Office of Scientific Research under Grant No. AFOSR-87-0047. The contract monitor was Dr. Jim McMichael. We are also pleased to acknowledge the assistance of Mr. Sean Hilbert, especially for his role in the fabrication of the beam divider.

7. REFERENCES

1. Popvich, A.T. and Hummel, R.L., A.I.Ch.E.J., Vol.14, pp. 21-25 (1967).

2. Seeley, L.E., Hummei, R.L. and Smith, J.W., J. Fluid Mech., Vol. 68, pp. 591-608 (1975).

3. Brown, G.H., ed., Photochromism, Wiley-Interscience, New York (1971).

4. Hecht, E. and Zajac, A. Optics, Addison-Wesley, pg 357 (1974).

5. Taylor, J.R. An Introduction to Error Analysis Univ. Sci. Books, Calif. (1982)

6. Lang, B.L. Laser Doppler Velocity and Vorticity Measurements in Turbulent Shear Layers. PhD Thesis GALCIT, Cal. Tech. (1985).

7. Hill, M.J.M. Phil. Trans. Roy. Soc. A Vol. 185 (1894).

APPENDIX C

THE CIRCULATION OF AN AIRFOIL STARTING VORTEX OBTAINED FROM INSTANTANEOUS VORTICITY MEASUREMENTS OVER AN AREA

R. E. Falco, C. C. Chu, M. H. Hetherington and C. P. Gendrich
Turbulence Structure Laboratory
Department of Mechanical Engineering
Michigan State University

ABSTRACT

By dissolving a photochromic chemical in an organic liquid and exciting it with ultraviolet light, a colored line can be created anywhere in the liquid. A device has been constructed which divides a laser beam into 'n' lines which are made to intersect to form a grid. Introducing two sets of these lines into the flowing fluid produces a grid of 100μm colored fluid lines with mesh spacing of approximately 1mm. The color change persists long enough so that the grid lines are distorted by the flow before disappearing. The process is completely non-intrusive. The lines are analyzed by an image processing algorithm to allow the grid intersection points to be located quickly and with high precision. The movement of the intersection points in a unit of time results in temporal gradients of position, which in conjunction with estimates of the circulation around each grid box can be used to obtain the vorticity. Additionally, differencing again across a grid box results in spatial gradients of velocity. Experiments in a Stokes' flow indicate that the vorticity can be measured to 1/sec. Application to the measurement of the formation of lift on an impulsively started airfoil has been made and compared with known results. It appears that the separating streamline is fixed to the trailing edge significantly before the Kutta condition is established, resulting in a starting vortex with less circulation than expected.

1. INTRODUCTION

The generation of lift has important technological consequences for both military and civilian aircraft. Whenever a change in velocity, attitude, or wing conformation is made, lift changes. For helicopters, each blade on the rotor continually experiences velocity changes. Batchelor[2] states: "It is a remarkable fact that in practice a circulation is generated around an airfoil, owing to the convection of a non-zero amount of vorticity from the rear edge of the airfoil at an initial stage of motion, and that when the airfoil is in steady motion the circulation is established with just this special value." The subject of this paper is to present measurements using a new experimental tool that can provide the detailed data needed to critique our understanding of this problem. It enables the simultaneous measurement of fluid velocity and vorticity at a large number of points in a flow field. We furthermore present preliminary measurements that suggest this problem is not as well understood as it appears, even in the simplest example of an impulsively started two-dimensional wing (called an airfoil).

The predominate view of the mechanism for the generation of lift on an airfoil with a sharp trailing edge, is that lift occurs because the zero-lift irrotational flow field which is set up immediately after an impulsive start would lead to the physically unrealistic condition of fluid passing around the trailing edge which has zero curvature at infinite velocity. If considered allowable at the initial instant, when the viscous layer around the airfoil is infinitely thin, this fluid would move into a strong adverse pressure gradient in the upper surface of the airfoil and separate because the retarded fluid in the boundary layer on the lower surface does not have the energy to negotiate the pressure rise from the infinite suction at the trailing edge to the rear stagnation point (see Batchelor[2] and Lighthill[11]). A vortex forms and is shed from the body. When this happens, the rear stagnation streamline moves to the trailing edge. Shortly after, a smooth flow is established, and we have the Kutta condition, which states that for the conditions of this problem, the separation streamline leaves from the trailing edge, and there is no net shear across this streamline. Figure 1 summarizes this classical view. The shed vortex is the so called 'starting vortex', which many authors suggest contains the full circulation. Calculations of the roll up of a semi-infinite vortex sheet such as those of Pullin[14] support this point of view.

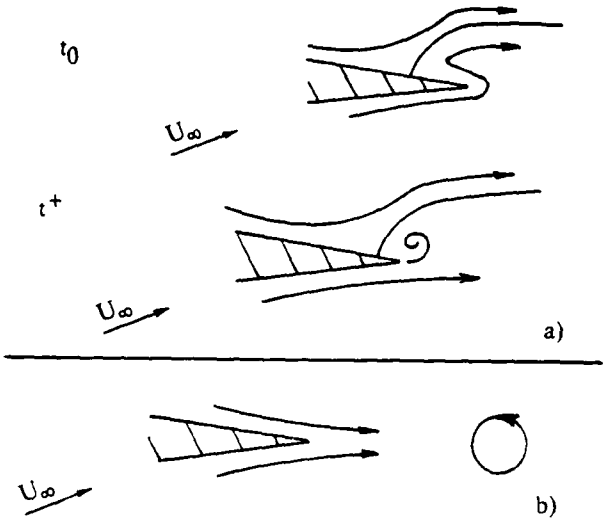


Figure 1. a) Evolution of the starting vortex from the irrotational solution. b) The starting vortex after the Kutta condition has been satisfied.

Many authors have not been entirely comfortable with this view. Lamb[9], states "It is still not altogether easy, in spite of the attempts that have been made, to trace out deductively the stages by which the result is established when the relative flow is started." Goldstein[6], after presenting it, states: "In this way the establishment of such a circulation may be held to be at least partly explained". Prandtl[13] states that "when the motion begins, a surface of discontinuity forms at the rear edge of the airfoil, and the fluid originally flows past it as..." irrotational flow around the airfoil plus the initial

rolled up sheet of discontinuity. This avoids the difficulty of infinite velocities, but he does not describe how the sheet of discontinuity forms. The Russian authors Kochin, Kibel and Roze[8] do not allude to the details indicated above, but only say that "the process of shedding of vortices with prevalence of vortices of one sign will continue until the circulation Γ reaches the value for which the velocity at the trailing edge of the airfoil will be finite". These differing descriptions are consistent with the lack of direct experimental evidence about the origin of lift. Up to now, it has been impossible to measure the instantaneous vorticity over the starting vortex, and at other relevant places. To our knowledge, only one set of measurements of the circulation in the fluid downstream of an impulsively started airfoil have been made. Walker[17] used a dense concentration of particles to determine the growth of the total circulation in the fluid aft of an airfoil that was started from rest. He found that more than six chord lengths were required before the circulation reached 90% of the steady flow value.

2. EXPERIMENTAL TECHNIQUES

2.1. Review of the measurement technique

The technique of Laser Induced Photochemical Anemometry (LIPA) is reviewed. The first use of excited photochromic chemicals to measure fluid properties was made by Popvich and Hummel[12]. They used the reaction to produce time lines, from which they obtained velocity fields for some quasi one-dimensional flows. The extension described by Falco and Chu[5] -- which is reviewed below -- enables time resolved measurement of planar velocity vectors, instantaneous Reynolds stress components, the in-plane strain-rate components and the component of vorticity perpendicular to the plane of interest in fluctuating three-dimensional velocity fields.

2.1.1. The photochromic dye

Use is made of the phenomena of 'photochromism' to measure both velocity and vorticity in the starting airfoil problem. Brown[3] has defined it as "a reversible change of a single chemical species between two states having distinguishably different absorption spectra, such change being induced in at least one direction by the action of electromagnetic radiation." The chemical used in this study was Kodak 1,3-Trimethyl-8-nitrospiro[2-H-1-benzopyran-2,2'-indoline]. We excited it at 351nm using pulses of 20ns produced by a Tachisto Excimer laser running on XeF (approximately 100mj output). The chemical, which was dissolved in kerosene, reflected blue light for the order of tens of seconds. Thus, the fluid was tagged long enough for it to move appreciably. In practice, for the velocities of our experiments, we needed to follow the lines only a fraction of a second to obtain the information required.

2.1.2. Use of a grid to tag fluid particles

To enable us to measure two components of the velocity in a flow using the visual technique, we must tag and follow specific fluid particles. The simplest way to do this is to have two lines intersect. That intersection defines a point in the flow which, because it is colored, we can follow. To measure a spatial gradient, we must simultaneously mark two points in the flow. To enable us to measure a spatial gradient in a prescribed direction, we need several points and must use interpolation. Thus, the creation of a grid of intersecting marked fluid lines enables both velocities and their gradients to be measured.

We required a method of dividing a beam of laser light into 'n' beams where 'n' would be 5-25. In doing so, we sought to lose as little light as possible. Thus, we designed a divider that appears as an oversized blazed reflection grating with step size approximately 2mm (see Figure 2). The incident laser energy was directed at a shallow angle to the plane of the grating. Because the facets are so much larger than a wavelength of light, we expected to see primarily specular reflection. The separation of the specularly reflected portions of the beam were determined by the difference of the angle of incidence and the blaze angle, and the grating step width and spacing. It was fabricated by individually fixing aluminum coated mirrors (with the coating optimized for shallow incidence angles) to a machined base. Apart from diffraction effects, no incident laser energy would be lost.

As expected, however, diffraction effects were exhibited. We determined that as long as the grid was formed closer than approximately 1 m from the divider, we were in the Fraunhofer regime, and we obtained no noticeable diffraction effects.

2.1.3. Algorithms to obtain fluid dynamic quantities

For convenience, we pick up one 'grid box' which, if the mesh size is small enough, can be thought of as a text book fluid 'particle' in the flow at time $t=0$. As this particle moves with the flow, it may undergo translation, rotation and deformation. Because we know the history of this specific fluid particle, we can compute the velocity of each corner by taking the time derivative of displacement $d\underline{s}$ associated with each point, that is, $d\underline{s}/\Delta t$. The deduction procedures are described in Falco and Chu[5]. In a three-dimensional flow field a fluid particle may have motions about all three coordinate axes; however, we will obtain the projections of this motion onto the plane of the photos which is parallel to the initial plane of the grid.

To obtain velocity gradients, vorticity, and strain rates from the velocities, we must difference across the distance of a grid box. Thus, for a given flow the grid must be small enough to resolve the smallest scales of motion.

An alternative method can be used if we are only interested in the vorticity. Using the definition of the circulation Γ.

$$\Gamma \equiv \int_C \underline{V} \cdot d\underline{s} \qquad (1)$$

and Gauss' theorem to relate the surface integral to an area integral,

$$\Gamma = \int_A \underline{\omega} \cdot \underline{n} dA \qquad (2)$$

we can obtain the vorticity component normal to the film plane, ω_z, from a grid box of arbitrary shape, with circumference C and area A.

The velocity component $\underline{V} \cdot d\underline{s}$ along any side of a grid box in Eq. (1) is estimated by forming the average of the velocity components obtained at two of the corners of the box. Dividing Γ by the area of the box results in the average vorticity at the centroid of this area. This technique has the advantage of avoiding a second differencing of the experimental data.

Thus, by following the distortions of a single grid box, we can obtain the velocities, circulation, vorticity and even the instantaneous Reynolds stresses and strain rates if desired, over that small area. By doing this for the other mesh elements, we can obtain the important fluid dynamic quantities at many locations over a two-dimensional field in a fluid flow simultaneously.

The photographs were digitized both manually using a NAC motion analyzer, and automatically using a Westinghouse ETV-2000 camera which was controlled by a MegaVision 1024XM image processor. The scanner and processor both are capable of dealing with images as large as 2048 x 2048 pixels. The analysis algorithm used the fact that relatively straight lines were present in the flow, and the intersections of those lines were the points to be followed. By using spatially biased filtering kernels, the centerline of each set of lines was extracted from the image, and the logical intersection "AND" was obtained.

The intersection information was processed into fluid kinematic quantities on an IRIS 3120. The IRIS provided not only a fast CPU but also a graphics environment in which the individual steps of the algorithm could be checked for consistency.

2.1.4. Error analysis

Consider a single grid box. We will assume that the grid mesh is spaced ϵ and that a worse case line movement between images is 1/2 the mesh width. After time Δt standard analysis (see Taylor 1982) shows that the error in estimating the vorticity ω_z is

$$\delta\omega_z = (1/\Delta t)(\delta\epsilon/\epsilon) \qquad (3)$$

to first order in ϵ.

As an example of the designed accuracy of our experiments, if we produce 100μm lines on a grid mesh of 1mm, and our error in reading the center of the lines is 10%, we obtain $\delta\epsilon/\epsilon$ = .01. For Δt = .01sec, the error in the measurement of vorticity is 1/sec. This is better than has been possible with hot-wire probes, and is comparable with the most sophisticated single point laser doppler measurements (see Lang[10]).

2.2. Experiments performed

Two experiments are described. The first may be considered as a calibration of the LIPA technique. Its objective was to measure the boundary layer that develops on an impulsively started moving wall. Since this has an exact solution of the Navier-Stokes equations, we can use it to determine the accuracy of our vorticity measurements. The second experiment concerned measurement of the lift formation process on an impulsively started airfoil.

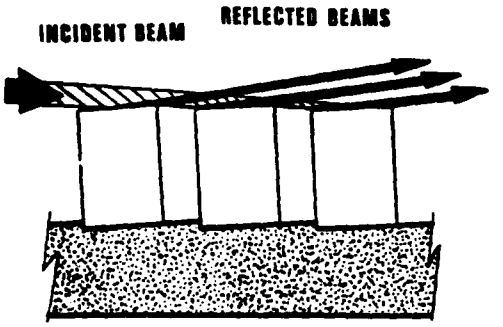


Figure 2. The optical setup used to produce a grid of laser lines.

For both experiments the same laser, chemicals and similar optical arrangements were used. A Tachisto excimer laser outputting 100mj and 351nm was used to excite the chemical. Figure 2 shows a sketch of the optical setup used to produce a grid of laser lines. Figure 3 shows an example of the grid produced after it has been distorted by the fluid motion. For the Stokes' layer problem the grid extended across the entire layer. For the airfoil problem Figure 4 shows the size of the measurement volume relative to the starting vortex diameter as indicated by the roll up of photochromic dye generated on the airfoil for the case when the airfoil had traveled 1.3 chords. This is the largest it grew to during our measurements. Thus we anticipate capturing essentially all of the vorticity in the starting vortex. Figure 5 shows the setup used to create the impulsively started flows. The mylar belt was chain driven by a 1/4 HP Dayton DC motor regulated by a Dayton SRC controller. In the starting airfoil experiment this combination accelerated the belt to a final velocity of 61mm/s in .06 sec (.057 of a chord length). The airfoil, which approximated a NACA 0012, had a chord length of 63.5mm. The chord Reynolds number was 1550, and it was fixed at a 4.7 degree angle of attack.

A 35mm high speed Photosonics 35ML camera with a quartz regulated motor was used to record data. A lens arrangement providing approximately 1:1 magnification was used. For the Stokes' flow experiments, we framed at 150 frames/second, while for the airfoil experiment 50 frames/second were used. For the airfoil problem the following timing sequence was used. The camera was started approximately 1 second before the belt to bring it up to speed, then the belt motor was started along with the delay timer. The delay timer triggered the laser at preselected delay times corresponding to .3, .8 and 1.3 chord lengths of travel of the airfoil. Only the first two frames after the laser fired were used, in each case, for the calculation of the vorticity.

3. RESULTS

3.1. Stokes' layer flow

We can test the accuracy of the technique in a flow for which an exact solution of the Navier-Stokes equations is obtainable. The flow is that of an impulsively started infinite plate in an initially still infinite ambient. The problem involves the two-dimensional diffusion of vorticity generated at the plate into the ambient. The thickness of the boundary layer, δ, is a function of the kinematic viscosity, ν, and the time after the start of the plate's motion. We have approximated this motion in a tank using a belt which can be rapidly started (see Figure 5). The width of the belt is 17.8 cm, and the distance between the rollers is 152.4 cm, with the test position 90 cm downstream of the lead roller. All experimental data was obtained before the leading edge effects reached the test position. Furthermore, the aspect ratio is sufficient to prevent the disturbances generated in the upstream corners from reaching the center of the belt at the test section in the test time interval. Therefore, the wall layer flow measured was two-dimensional. The speed of the belt was adjustable from 2.5 to 22.9 cm/sec. It reaches a constant speed within .06 second. This short acceleration reasonably approximates an impulsive start. A grid mesh size of .15δ was used, with the laser lines intersecting as in Figure 5.

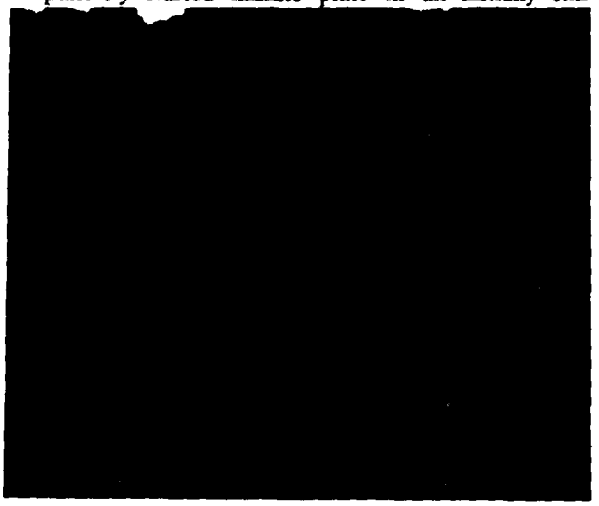


Figure 3. A photo of the grid after it has been distorted by fluid motion. The wing is moving left to right.

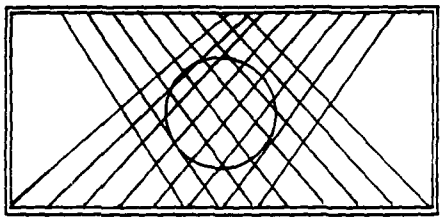


Figure 4. The size of the measurement region with respect to that of the starting vortex at .3, .8 and 1.3 chords behind the airfoil. The wing is moving left to right.

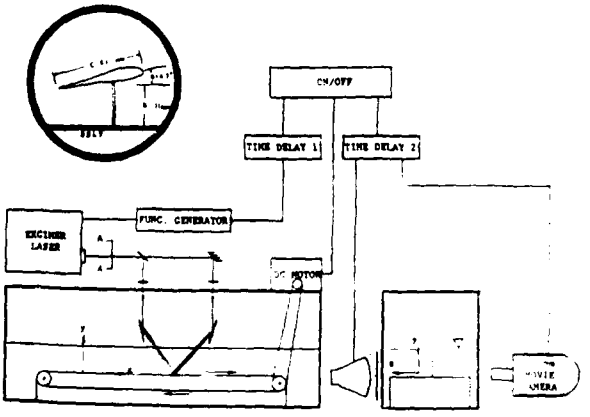


Figure 5. Setup used to create and measure the Stokes' flow. This same device was modified to support an airfoil (see inset).

Results of the velocity gradient obtained from double differentiation in the Stokes' layer are shown in Figure 6, where they are compared with the exact solution. Both data reduction by hand and using the image processing equipment are shown. All quantities are non-dimensionalized by the similarity variables of the exact solution. Because the flow is two-dimensional, incompressible, and doesn't change with the streamwise coordinate, x, the continuity equation reduces to $\partial v/\partial y = 0$, and a check on this is also shown in Figure 6 (v is the velocity component normal to the wall). We see that both the error in the estimates of the continuity equation, and in the velocity gradient distribution are of similar magnitude.

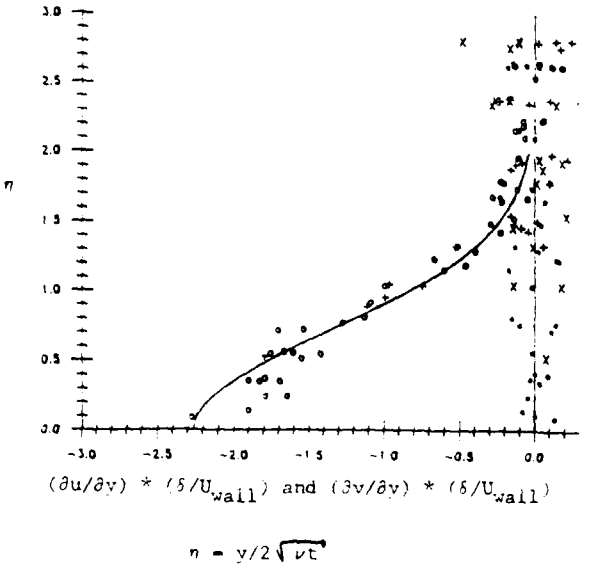


Figure 6. Comparison of experimental measurements of the velocity gradient in a Stokes' layer compared with the theoretical solution. (+) the algorithm, (o) manually, (-) the exact solution. A check of the continuity equation $\partial v/\partial y = 0$ is also shown. (x) the algorithm, (*) manually.

We can make use of all of the samples in the continuity equation to get a statistical estimate of the error in measurement of velocity gradients. The rms error equals .9/sec, consistent with the design error analysis. The deviation of the experimental profile in the near wall region is a consequence of the grid mesh being too large to adequately resolve the large value of the gradient $\partial u/\partial y$. On the whole the results indicate that we should be able to measure vorticity over an area more accurately than instruments using multiple hot-wires or multiple LDA beams can at a point. Furthermore, the resolution of each vorticity measurement can potentially be higher, since we are essentially constrained only by the minimum thickness of an ultraviolet laser beam.

3.2. Impulsively started airfoil

Using the photochromic dye, we could observe the direction of motion of fluid particles, and measure the time required for a point in the fluid to move along the upper surface to the trailing edge. We could observe motion in increments of $Ut/c = .02$. The motion of marked fluid was always observed to be downstream with respect to the airfoil. By positioning the airfoil under the laser generated grid, and firing it at a repetition rate high enough, we could successively mark selected "fluid particles" on the airfoil's surface and follow their movements. Doing this, we obtained a definite indication that the separating streamline was fixed to the trailing edge by $Ut/c = .06$. We could resolve movement of .006 chord. During this time a vortex formed on the LOWER surface of the airfoil, very close to the trailing edge, but beneath the fixed separation point. It should be noted that although the trailing edge was sharp when viewed with respect to the airfoil as a whole, it had a radius of curvature of approximately .42mm.

Having established that the process of movement of the separation point was completed in $Ut/c = .06$, we examined the starting vortex. Using the grid technique described above, we measured the vorticity and circulation in the starting vortex after it had convected .3, .8, and 1.3 chord lengths downstream. Γ/Γ_∞ was .392, .378 and .387. Thus, the magnitude of the circulation has not changed within our experimental error. The steady state circulation for our NACA 0012 airfoil was estimated to be 1024mm^2/sec. The estimate at the .3 chord included all of the circulation aft of the airfoil. We see that the estimates indicate the starting vortex has developed to full strength by .3 chord. Thus, the circulation in the starting vortex is approximately 1/3 of the total circulation around the airfoil in steady motion.

The fact that Γ of the starting vortex had not reached the theoretical value of $\pi c U_\infty \sin\alpha$ by this position downstream was expected from results of Walker[17], and theoretical results of Wagner[16], and numerical calculations of Basu and Hancock[1] and Chow and Huang[4], among others. Explanations have alluded to the fact that when the starting vortex was at these chord positions, the separation streamline had not yet reached the trailing edge, so that additional vorticity was being shed -- which might rearrange itself into additional vortices. However, we find that the separating streamline has moved to the trailing edge and remains fixed there by .06 chords, so the classical mechanism for the supply of vorticity is no longer operative.

Figure 7 compares our results with those of experiments of Walker[17] and theoretical estimates of Wagner[16]. It is interesting to note that the value of the circulation we measured at .3 chord is (within experimental error) the same as that measured by Walker and calculated by Wagner. They are at all times considering the total circulation of the fluid downstream of the airfoils. At .3 chord we are also measuring the total circulation of our airfoil. This correspondence is expected, and adds weight to the accuracy of our measurements. The discrepancy found at .8 and 1.3 chords thus highlights the fact that processes other than the classical physical picture are taking place. It is interesting to point out that for impulsively started airfoils, only a single vortex is expected if similarity is assumed (Pullin[14]). Thus, it appears that the process of roll up of the vorticity shed from the back of the airfoil does not conform to this hypothesis, since the starting vortex contains only a third of the vorticity responsible for the lift of the airfoil.

Further experiments were performed to determine whether additional vorticity was being shed off the airfoil after the circulation in the starting vortex

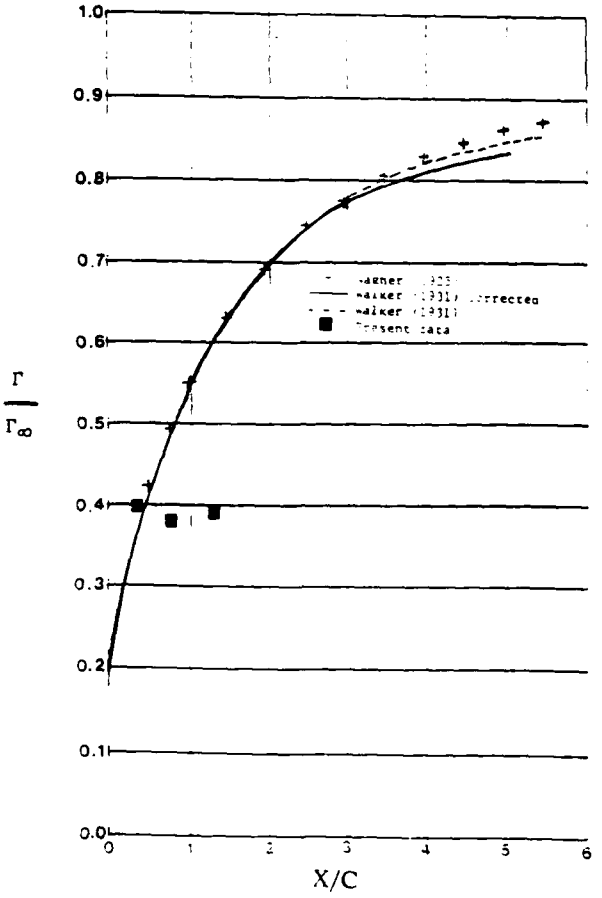


Figure 7. The increase in circulation with distance from the airfoil, compared with experimental and theoretical data.

had reached its constant value. A single photochromic line was created .07 chord lengths behind the airfoil after it had traveled .5, .75, 1 and 2 chord lengths. We examined the movement of this line, both above and below the airfoil to determine whether a shear existed across the wake. Indeed a shear existed in all the measurements. Figure 8 shows the distortion of a single photochromic line created approximately .06 chord lengths behind the airfoil, and photographed after approximately .22 chord lengths of travel, for the case where the airfoil has traveled 1/2 chord, i.e. after the starting vortex has attained its final value (see Figure 7). The sign of this vorticity was the same as that in the starting vortex. This magnitude decreased as the airfoil had traveled further, but measurable shear existed in the experiment even after two chords of travel. Thus, there was continued generation of vorticity long after the separation streamline was fixed to the trailing edge, and the Kutta condition was therefore not satisfied after at least 2 chords of travel in our measurements.

Figure 8. The shear that exists across the wake when the starting vortex is 0.5 chord downstream. The wing is moving right to left.

4. DISCUSSION

The demonstrated accuracy of the technique is sufficiently adequate to enable us to reexamine many unsteady fluid dynamic problems. Since, in principle, the calibration depends only on a time and a length scale, there is further room for improvement. In practice, determining the intersection points of the grid lines is limited by the grain size of the film used (or the resolution of a photo optical detector), the distortion of the developing process and the accuracy of either hand or automated measurements in defining the lines. Improvements in the grain limitation can be made by using a larger format camera or larger aperture lens/finer grain film combination.

For these preliminary low Reynolds number experiments, the average magnitude of the vorticity in each of our grid boxes was 3 sec^{-1}. Since the accuracy of our technique is $\pm$ .9 sec^{-1}, we could only expect 66% accuracy per individual estimate. However, estimates of the circulation for each case involved integrating over an average of 23 circulation cells, thus significantly reducing the error in estimating Γ. Furthermore, in a direct test of the reproducibility of our estimates, at .8 chords two independent runs were analyzed. These results indicated that the circulation varied by 4%. Another possible source of error was the possibility of not including all of the vorticity in the starting vortex for the case at 1.3 chords. However, as Figure 4 indicates, the dye suggests that most of the vorticity has been measured.

4.1. The generation of lift

If the circulation is no longer determined by the fact that the separating streamline has moved to the trailing edge, then what determines the magnitude of the circulation responsible for steady lift? Where is the additional circulation coming from? In the theoretical and numerical accounts, vorticity is continuously coming off in a sheet that rolls up into the starting vortex at some position downstream of the airfoil; the vorticity generation comes to an end when the Kutta condition is established. Graham[7] critiques analysis which suggest that a sheet is shed approximately parallel to the bisector of the trailing edge and then rolls up. He notes that all evidence suggests that the surface of discontinuity rolls up immediately as it forms and comes off the airfoil. Using this hypothesis he arrives at a picture of the initial lift being very high, decreasing and then increasing as the Wagner[16] function after Ut/c = .1. All of this work suggests that as the circulation is increasing the rear stagnation point is moving towards the trailing edge. However, it takes several chord lengths to have all of the vorticity shed.

The fact that we measured only 2/3 of the circulation expected in the starting vortex after 1.3 chords, and that there does not appear to be any change in the strength over the previous chord of motion, suggests that additional vorticity is continuing to be shed after the vortex rolls up. A contradiction arises. The classical theory suggests that the upper stagnation point will have moved to the trailing edge when the starting vortex has been shed, and thus there is no longer a mechanism for the generation of additional circulation. The unique irrotational solution for the circulation is established. However, in the irrotational solution the separating streamline is a stagnation streamline; in the real fluid it is not.

In our experiments the separation point has moved to the trailing edge after .06 of a chord of travel. Thus, it appears that the movement of the separation point to the trailing edge does not in itself establish the Kutta condition. It must be cautioned that we are not able to resolve the velocity field with sufficient resolution to determine the stagnation point, but our determination is being made by measuring the movement of lines of marked fluid that are created at large angles to the surface, and selected marked 'fluid particles' (intersections of the marked fluid lines). We see that initially the lines and marked points move downstream with respect to the airfoil, and that after .06

chord of airfoil travel they are fixed to the trailing edge. Thus, it is not at all certain that the initial position of the stagnation point is on the upper wing surface (we have no evidence for or against this possibility). After movement to the trailing edge (if this happens at all), one still has the presence of a strong shear layer. Accordingly, we feel that the shear layer still exists for more than 6 chords, i.e. the circulation is changing over airfoil travel of more than 50 times as long as indicated by the classical theory.

Finally, we note that this is a very low Reynolds number experiment. It could be argued that the lift on this airfoil will not be the same as that for a NACA 0012 at higher Reynolds numbers. However, it is interesting to note that the value for the total circulation which both Walker and we have measured after .3 chords is remarkably close. It is certainly possible to perform experiments of this type at considerably higher Reynolds numbers. The limitation is not the LIPA technique, but instead the mechanical mechanisms for rapid acceleration of the airfoil.

5. CONCLUSIONS

The generation of lift on an airfoil is re-examined. Using a new quantitative optical measuring technique, we determined that the separation streamline moves to the trailing edge of an airfoil long before the Kutta condition (unsheared flow from the trailing edge) is established, i.e. long before the change in circulation necessary to reach the steady flow lift has occurred. Preliminary results show that the vorticity in the starting vortex which has convected approximately 1.24 chords after the airfoil separation streamline has moved to the trailing edge, is much less than that predicted.

This suggests that the classical hypothesis relating the formation of lift and the establishment of the Kutta condition needs to be re-examined.

6. ACKNOWLEDGMENTS

We gratefully acknowledge support of the Air Force Office of Scientific Research under Grant No. AFOSR-87-0047. The contract monitor was Dr. Jim McMichael. We are also pleased to acknowledge the assistance of Mr. Sean Hilbert, especially for his role in the fabrication of the beam divider.

7. REFERENCES

1) Basu, B. C. and Hancock, G. J. 1978 The unsteady motion of a two-dimensional aerofoil in incompressible inviscid flow. *J. Fluid Mech.* 87, pp. 159-178.

2) Batchelor, G. K. 1970 An Introduction to Fluid Dynamics Cambridge University Press.

3) Brown, G. H. (ed) 1971 Photochromism Wiley-Interscience, New York.

4) Chow, C-Y. Huang M-K., 1982 The initial lift and drag of an impulsively started airfoil of finite thickness *J. Fluid Mech.* 118, pp 393-409.

5) Falco, R. E. and Chu, C. C. 1987 Measurement of two-dimensional fluid dynamic quantities using a photochromic grid tracing technique. Proc SPIE 31st Annual International Technical Symposium on Optical and Optoelectronic Applied Science and Engineering, 16-21 August.

6) Goldstein, S. (ed) 1938 Modern Developments in Fluid Mechanics Oxford University Press.

7) Graham, J. M. R. 1983 The lift on an aerofoil in starting flow. *J. Fluid Mech.* 133, pp 413-425.

8) Kochin, N. E., Kibel, I. A. and Roze, N. V. 1964 Theoretical Hydrodynamics Interscience, New York

9) Lamb, H. 1932 Hydrodynamics, 6th ed., Cambridge University Press.

10) Lang, B.L. Laser Doppler Velocity and Vorticity Measurements in Turbulent Shear Layers. PhD Thesis GALCIT, Cal. Tech. (1985).

11) Lighthill, M. J. 1986 An Informal Introduction to Theoretical Fluid Mechanics Clarendon Press, Oxford.

12) Popvich, A.T. and Hummel, R.L., 1967 Experimental study of the viscous sublayer in turbulent pipe flow. *A.I.Ch.E.J.*, Vol.14, pp. 21-25.

13) Prandtl, L. 1952 Essentials of Fluid Dynamics Hafner, New York.

14) Pullin, D. I. 1978 The large-scale structure of unsteady self-similar rolled-up vortex sheets. *J. Fluid Mech.* 88, pp 401-430.

15) Taylor, J.R. 1982 An Introduction to Error Analysis Univ. Sci. Books, Calif.

16) Wagner, H. 1925 Uber die Entstehung des dynamischen Auftriebes von Tragflugeln. *Z. angew. Math. Mech.* 5, pp. 17-35. (Also NACA TM 1139 (1948)).

17) Walker, P. B. 1931 Experiments on the growth of circulation about a wing. British Aeronautical Research Council R & M 1402.

APPENDIX D

Appendix D contains the original list of equipment requested under AFOSR-86-0242. It also contains the list of equipment which was ultimately purchased, as well as a copy of the letter explaining our reasons for requesting equipment substitutions.

FINAL EQUIPMENT LIST
AFOSR-86-0242

COMPANY/ITEM	71-2309	MSU EQPT. COST-SHARING	MSU OTHER COST-SHARING
CAM Corp. (P.O. 36154) Seiko color copier with video multiplexer;maintenance	------	$ 9,741.00	$ 1,593.00
Helix Ltd. (P.O. 37409) Hasselblad 135MM Macro lens for Westinghouse Camera System	$ 1,039.00	------	------
Helix Ltd. (P.O. 52681) Hasselblad lens mounting ring; gelatin filter holder; lens mount adapter; auto bellows extension; shipping	------	1,053.00	------
Megavision (P.O. 36153) Megavision Image Processing System, 1024XM;Dual output channel;2-4Mbyte image memory;Pixel Data Flow Processor;2 1 Mbyte boards, Color monitor;maintenance	53,489.03	3,008.41	2,397.30
Optimem (P.O. 36155) Optimem 1000 Disc Driver; 4 singlesided media; ext. warranty	13,600.00	1,646.00	------
Silicon Graphics (P.O. 36156) Iris 3120 workstation; tape drive; accelerator; processor;compiler; extra display memory; software;ext. warranty	44,525.02	------	5,590.01
Sun Microsystems (P.O. 36152) Sun 3/260HM-P2 Workstation; 481A Multiplexer;License, Software;Disc Driver;DNA/ 2-01 Software;maintenance	44,919.22	2,900.00	5,437.50
Vanguard (P.O. 36417) M-35-C Projection head	------	8,863.29	------
Westinghouse (P.O. 36416) ETV-2000 Camera System; maintenance	19,800.00	1,000.00	------
Travel	------	------	3,646.62
Freight	------	------	1,044.50
Other	------	------	2,704.93

Overhead (42% MTDC thru 6/30/88; 44% 7/1 thru 7/31/88)	------	------	9,444.29
TOTALS	$ 177,372.27	$ 28,211.70	$ 31,858.15

ORIGINAL EQUIPMENT LIST

EQUIPMENT

DIPIX image processing system	
Aries III image processor	31,500
3 megabytes image memory	15,600
pixel processor	9,950
vector graphics package	10,000
extended software package	15,000
Hitachi 19" color monitor	5,200
16-32 bit image converter	3,000
fortran 77	2,500
vgt100 graphics monitor/printer	3,500
installation	3,000
Training	2,000
support 1 year at 1%/mo	11,550
Subtotal	112,800
EIKONIC DIGITIZER	
EC78/99 Digital camera	16,500
L-01 Condensor light source	5,500
C-10 Workstation	4,800
E-12 interface to Microvax II	2,500
Driver	1,500
EO-2 Data Normalizer	1,000
I-01 Color filter wheel	2,500
previewer	1,800
Disc application software	1,000
105 mm Nikon macro lens	900
	38,000
Less 6% educational discount	-2,280
	35,720
Factory installation	1,800
Support 1 year at 1%/mo	4,344
Subtotal	41,864
DEC MICRO VAX II HOST	
Micro Vax II	32,686
Includes:	
MVII SBB:1MB/FP BA123	
RD/RX DISC CONTROLLER	
RX50 W/CABLES	
RD53 W/CABLES	
MICROVMS SOFTWARE	
NETWORKING	
MANUELS	
FORTRAN 77	

DEC RA81 456 mb disc/controller	27,000
DEC 9-track tape drive	9,990
	69,676
Less 14% educational discount	-9,755
	59,921
8mb Chryslin memory	3,380
installation	2,000
support 1 year at 1%/mo	7,190
Subtotal	72,491
Total Equipment	202,071
Total Direct Costs	227,155
Indirect Costs (41% of Direct Costs less equipment)	10,284
Total Cost	237,439
MSU Cost Sharing (25.3%)	-60,000
Total Cost to DOD	$177,439

MICHIGAN STATE UNIVERSITY

OFFICE OF ASSOCIATE DEAN FOR
GRADUATE STUDIES AND RESEARCH
COLLEGE OF ENGINEERING

EAST LANSING • MICHIGAN • 48824-1226

February 2, 1987

Dr. Jim McMichael
AFOSR/NA
Bolling AFB
Washington, D. C. 20332-6448

Dear Jim:

Request for Equipment Substitutions in DOD Equipment Grant
AFOSR-86-0242 DEF

I have made a thorough re-evaluation of the image processing equipment in the grant, and have determined that a better system can be put together for the same costs. This system will enable us to more accurately measure the vorticity and strain rates, and to better understand the data. It will also enable us to interface with the Direct Numerical Simulations that will be coming from NASA Ames and other places.

The major new components of the system are:

a. Westinghouse 2" video camera that gives 2048 x 2500 pixels;
b. Megavision 1024M processor that will process 2048 x 2048 images and can be upgraded to 4096 x 4096 images which will be necessitated by later higher speed experiments;
c. Silicon Graphics IRIS processor, which will allow us to view the data from any angle and to reconstruct the three-dimensional fields from the two-dimensional slices of vorticity data we are obtaining, and compare them with numerical computations.
d. Sun 3/260 with optical disk, to be the host computer for the network of digitizer, image processor and graphics processor, mass storage, and communications.
e. A hard copy device that will enable permanent records at the resolution of the screen images.

A detailed list of the new equipment is attached. In each case I have bargained for some special deal with the vendors. The net result is that I have achieved this significant enhancement of system capabilities at no additional cost.

Permission is requested to make these substitutions.

Sincerely,

Bob Falco

Robert Falco
Professor and Director
Turbulence Structure Laboratory

Approved:

John R. Lloyd, Chairperson
Mechanical Engineering Dept.

Howard G. Glider, Director
Contract & Grant Administration

William C. Taylor, Acting Assoc. Dean
Graduate Studies and Research

RF:kk
enc.

PROPOSED REVISED EQUIPMENT COST BREAKDOWN
FOR AFOSR-86-0242, R. E. FALCO
1/26/87

	TOTAL COST	GRANT	MSU EQ	MSU S&S
[illegible] Image Processing system,1024XM: Dual output channel:2-4Mbyte Image memory: Pixel Data Flow Processor:2 1 Mbyte boards,Color Mon.	57710 +frt	53360	3000 +frt	1350 (ext.warrant.+ integration fee)
Sun 3/260HM-P2 Workstation:481A Multiplexer: License,Software: Disc Driver:DNA/2-01 Software	53350 +frt	44879	2861 +frt	5610 (1yr maint.)
West ETV-2000 Camera System	21800 +frt	19800	+frt	2000 (3yr maint.)
Hasselblad 135 Lens for West.	1400	1400		
Iris 3120 worksta: Tape Drive: Accel: Extra display mem. Software	49865 +frt	44400	+frt	5465 (ext.warrant.)
Vanguard M-35-C Proj. head,lens	9040 +frt		9040 +frt	
[illegible] media	16700 +frt	13600	+frt	3100 (4media+ext. warrantee)
[illegible] opier [illegible] plexer	10470 +frt		9561 +frt	909 (1yr maint.)
[illegible] topier	360			360
[illegible] for platform				

[illegible]ntegration	3000			3000
	1051			1051
[illegible]t)	2400			2400
[illegible]Cost	227146	177439	24462	25245
[illegible]	10350			10350
Total Cost	237496	177439	24462	35595

Cost by source AFOSR - 177439 MSU - 60057