This is to certify that the

thesis entitled

MEASUREMENTS OF MIXING DURING
SCAVENGING IN A TWO-STROKE ENGINE

presented by

H. Sean Hilbert

has been accepted towards fulfillment
of the requirements for

Masters degree in Mechanical Engineering

R E Falco
Major professor

Date 21 Feb 91

MEASUREMENTS OF FLOWS DURING SCAVENGING IN A TWO-STROKE ENGINE

By

H. Sean Hilbert

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Mechanical Engineering

1991

ABSTRACT

MEASUREMENTS OF FLOWS DURING

SCAVENGING IN A TWO-STROKE ENGINE

By

H. Sean Hilbert

LIPA (Laser Induced Photochemical Anemometry) was used to measure velocities and velocity gradients over a chosen plane in a motored two-stroke engine during scavenging. The LIPA technique consists of tracking a phosphorescing grid which was created by laser lines directed into the flow. The grid energized a seed chemical that was premixed in the carrier gas. The seed chemical used consists of a mixture of phosphorescent gases with nitrogen as the carrier. In each plane forty-four simultaneous points of data were taken with an approximate grid mesh size of 3mm x 3mm. These measurements were taken over thirty consecutive cycles. By measuring the distance and direction each grid intersection traveled and by knowing the time delay between each photograph, the two velocity components in the grid plane, the turbulence intensities, the Reynolds stress, and the vorticity were calculated.

Images were taken of grids formed in planes parallel to the piston crown in a single cylinder 125cc loop scavenged engine. Averages over the area of interest and over the ensemble of two-dimensional maps were used to look at mixing, cyclic variability, and general flow phenomena.

ACKNOWLEDGEMENTS

I wish to express my thanks to Professor Robert E. Falco for his guidance, advice and patients. His willingness to allow me to take a course of action which most interested me will always be greatly appreciated.

I must also thank the National Science Foundation for their support in the form of an "Award For Creativity In Engineering." Without it, this work on two-stroke engines would continue to be something that I wished I could do someday. The donation of two engines from Kawasaki Motors Corporation is also much appreciated.

Mulled Toast Two

To Sir Douglad Clerk I raise my glass,
his vision places him first in class.
For Alfred Scott let's have your plaudits,
his squirrels drove the four-strokes nuts.

To Motorradwerk Zschopau I doff my cap,
their Walter Kaaden deserves some clap.
Sirs Boyesen and Fox need a mention,
their flair for design got my attention.

The sport of motocross I allege,
initiated my thirst for two-stroke knowlege.
In academe I found my wish
in Blair and Bracco and Carroll Smith.

To the great racers I lift my hat
they make the adrenalin pump pitter-pat
for the Americans at that are always best
like Hannah, Ward, Johnson and the rest.

In case you think, as you peruse this tome
that a computer terminal is my mental home,
I've raced my motorcycle with highest of hopes
and every Winter managed to ski the steepest slopes.

(Adapted from Gordon Blair's poem "The Mulled Toast")

Table of Contents

Table of Contents — Continued

Table of Contents — Continued

List of Figures

List of Figures — Continued

List of Tables

Nomenclature

i — Measurement location index.

j — Index for cycle number.

n — Total number of measurement locations (grid intersections) within each data frame.

m — Total number of data frames.

$< >_S$ — Spatial mean in frame j.

$< >_C$ — Cyclic average at measurement location i.

Γ — Circulation.

ω_z — Vorticity component perpendicular to grid plane.

$\sqrt{\overline{u'^2}}$, $\sqrt{\overline{v'^2}}$ — The RMS turbulence intensities calculated with cyclically averaged velocities.

u'_{rms}, v'_{rms} — Abbreviated versions of the above quantities.

q_{rms} — The RMS turbulence kinetic energies calculated with cyclically averaged velocities.

u'v' — Reynolds stress (turbulent shear stress).

A — Area of a grid box.

C — Perimeter of a grid box.

$\hat{n}$ — Unit normal vector to grid plane.

$\vec{S}$ — Path along which circulation is calculated (circumference of a grid box).

$\vec{V}_{i,j}$ — Velocity vector at a point.

TDC — Top Dead Center (Piston is in the uppermost position).

ATDC — After Top Dead Center

BDC — Bottom Dead Center (Piston is in its lowest position).

SR — Scavenging Ratio (= mass of air supplied/swept volume of cylinder).

SE — Scavenging Efficiency (= mass of air trapped/mass of air supplied).

RPM — Revolutions Per Minute.

CHAPTER 1

INTRODUCTION

It is widely recognized that scavenging of the two-stroke engine is the single largest unknown in its design. From a researcher's or designer's perspective this engine is very complex. In the two-stroke engine the intake and exhaust processes are not separated. Instead of a rising piston doing the work of pushing burnt gases out of the exhaust port, as in a four-stroke engine, the two-stroke engine uses the incoming fresh charge to accomplish this task. This coupling of the scavenging (ridding the engine of burnt gases) and the intake processes makes designing critical engine parameters of a two-stroke very difficult.

The complexity of the gas flows in a loop scavenged two-stroke engine can best be illustrated with a step-by-step description of a single engine cycle; As the piston ascends in its stroke the trapped contents of the cylinder are compressed and ignited. As the piston moves upward it creates a vacuum in the crankcase. Into this vacuum is drawn a fresh charge (or air in the case of a fuel injected engine). As the piston descends after combustion, the crankcase is pressurized and the charge is forced out of the crankcase through ports which transfer it to the combustion chamber. When this fresh charge reaches the cylinder it comes in contact with exhaust gases from the previous engine cycle. In a turbulent clash the incoming charge attempts to push the remaining exhaust out of the cylinder and ready itself for combustion. The piston ascends and the cycle starts over again. For a graphical explanation see Figure 1.1.

As noted, scavenging is accomplished with the use of the incoming charge, and this is a very complicated fluid dynamic process. In theory the incoming gases are aimed in such a way that they "loop" around the cylinder and force the burnt charge out of the exhaust port. A theoretical loop scavenging process is illustrated in Figure 1.2. The output and efficiency of any two-stroke engine are dependent on its scavenging behavior. In the case of the loop scavenged two-stroke cycle spark ignited engine (the subject of this work) the importance of the geometrical arrangement and design of the scavenging ports has long been realized by engine designers and researchers alike. Work by Jante [25], Blair [1,5], and Phatak [26] has substantiated this.

Over the years researchers have used various experimental techniques to study scavenging processes and develop scavenging port systems. The experimental methods used fall into two categories; namely (a) those that use firing engines as their basis, and (b) those which are based on model or rig tests. A third class of technique, not to be included with experimental studies, is prediction of engine performance with the aid of computer models. Examples of class (a) include the determination of overall scavenging performance using cylinder gas pressure and temperature data as reported by Hashimoto et. al. [4], and using exhaust gas analysis to study short circuiting as published by Nuti and Martorand [5]. The general approach in class (b) is to study the scavenging system in isolation from the often variable gas dynamic and thermodynamic effects presented in an actual firing engine. A classic example of this is the Jante method [25], whereby the engine is motored at a constant RPM without the head in place, and "scavenging maps" taken with a rake of pitot tubes placed above the engine parallel with the cylinder axis are evaluated. This is a widely used method and has been shown by Blair and Kenny [1] to be capable of ranking in terms of best performance a group of engine cylinders which differ only in the design of their scavenging port systems. One disadvantage of the Jante method is that the results produced cannot be compared with theoretical isothermal scavenging models such as the pure displacement and perfect mixing models as presented by Hopkinson [6]. Another disadvantage is that, since the head is off, the information gained can only be used for comparative purposes with other similar porting configurations. More recently Ishihara et. al. [12] have tried to extend the Jante measurements to three dimensions without much success. A more contemporary example of class (b) is the single cycle model approach first suggested by Hopkinson [6]. Recent variations of this method presented by Sanborn and Roeder [7] and Blair et. al. [2], have shown that the results can be compared directly against the isothermal pure displacement and perfect mixing curves. Sweeny et. al. [3] concluded that this method provides an accurate and reliable method of assessing the absolute isothermal scavenging efficiency vs. scavenging ratio characteristics of either model or real loop scavenged two-stroke cylinders.

Computer modeling of the scavenging process may be divided into three categories according to Sher [9]: one phase, multi-zone, and hydrodynamics models. One phase models are in effect the upper bounds on the scavenging process. This category includes the perfect displacement and perfect mixing models as mentioned before. As the names imply, the perfect displacement model assumes all incoming fresh charge pushes out the burnt gases without any mixing, momentum transfer, or heat transfer. The perfect mixing model assumes that the fresh charge mixes instantly with the cylinder contents to form a homogeneous mixture, and the excess of these contents escapes through the exhaust port. In multi-zone models, the cylinder is divided into two, three or more zones. Visualization of scavenging also conducted by Sher [10] revealed that the scavenging process may be approximated to proceed in three principal phases; displacement, mixing, and short circuiting (fresh charge exiting the exhaust port). Multi-zone models take into account these phases to produce a higher level of authenticity.

The above thermodynamic models offer a high level of simplicity, however, the best description of the scavenging process would be obtained if the complete set of the differential equations which govern the process could be solved to yield the time variation of the spatial profiles of the temperature, mixture composition, and flow field.

These equations consist of the conservation laws of momentum, mass, energy, and species, and the present state of computational ability requires that models for transport coefficients and boundary conditions be used. Modelers such as Sher [10] produce their own set of equations and models for turbulence and heat transfer, whereas Blair [28] applied the PHOENICS program to a three dimensional simulation of a loop scavenged two-stroke cylinder flow using the k-ε model to describe turbulent transport. A similar, geometrically more correct simulation also using the k-ε model for turbulence was conducted by B. Ahmadi-Befrui et. al. [12].

In-cylinder measurements of velocity and turbulence have been made on a limited basis; mainly to formulate better boundary conditions for the above models. J.G. Smyth et. al. [13] made cycle resolved Laser Doppler Anemometry (LDA) measurements of scavenge port exit flow. These results showed that the efflux angles of flow from the ports was substantially different from the designed direction of the port exit. Replacing the slug flow boundary conditions used in Blair's PHOENICS model with these new findings produced scavenging characteristic results very comparable with those which were experimentally obtained. Although in-cylinder velocity measurements taken during scavenging are scarce in the literature, there is much more information available on cylinder flows in ported engines near TDC. Fraser and Bracco [14] measured turbulent length scales in a motored two-stroke engine and reported scales on the order of 3mm at 320^{o} ATDC. This is consistent with other reports by Reddy et. al. [27], Obokata et. al. [15] and Hall and Bracco [16]. All of these measurements were recorded within thirty degrees of TDC. Reddy's results were taken in a motored two-stroke with a hot-wire probe mounted in place of the spark plug. At a motoring speed of 500RPM he found mean velocities during scavenging to be approximately 3m/sec in the region of the spark plug. Turbulence intensities were on the order of 1m/sec.

While important strides have been made in measuring the flow in an engine environment using hot-wires and LDA, a greater need exists for more comprehensive information; both to improve computer modeling and to directly affect engine design. The shortcomings of single point data as a tool for the study of turbulence, combined with the problem of the inability to resolve and separate out cyclic variability have highlighted the need for new methods of measurement. Velocity data obtained at many points simultaneously are very important if the overall fluid flow picture inside an engine is to be found, therefore, the search for an accurate and efficient technique is pertinent. In addition, a time history of these flow patterns as a cycle progresses would be of great value. Methods such as particle tracking [17] and Particle Image Velocimetry (PIV) [18] have been developed to enable researchers to look at instantaneous spatial data. Both have important roles to play in engine diagnostics. However, the use of particles has inherent drawbacks. A technique is sought that can provide adequate spatial information and resolution, high temporal resolution, and simple data reduction. The technique of Laser Induced Photochemical Anemometry (LIPA) has these features. It is used in a gas in this preliminary study of the scavenging motions in a two-stroke engine (schematically illustrated in Figure 1.2). The primary purpose of this work is to call attention to the potential of the LIPA technique as a tool in engine diagnostics and design. It is also used to show the extent of cyclic variability of turbulence quantities and to illustrate a repeatable flow variance relevant to scavenging in a modified production engine.

CHAPTER 2

EXPERIMENTAL SETUP

2.1 Engine

A 1989 Kawasaki model KX-125 single cylinder loop scavenged production engine was chosen for these experiments. In stock form the engine's intake system consisted of a 32mm round bore carburetor mated to a manifold containing a pair of two-petal reeds. The manifold coupled directly to the crankcase where the charge was subsequently fed into the cylinder through five transfer ports (two main ports, two auxiliary ports, and a boost port). Combustion was initiated by a spark. In production form the exhaust port had automatic height adjustment controlled by engine speed, and four small auxiliary exhaust ports aided by a Helmholtz resonating chamber. Products of combustion were carried out of the engine through a tuned exhaust. Complete engine specifications are given in Table 1, and exhaust pipe specifications are shown in Figure 2.1.

Table 1. Engine Specifications

Bore	56.0mm
Stroke	50.6mm
Displacement	124cc
Compression Ratio	8:1
Port Timing: (ATDC)	
Exhaust Port Opens	90.5°
Transfer Ports Open	117°
Boost Port Opens	117°
Exhaust Port Fully Open	155°
Transfers Fully Open	BDC

Several slight modifications were made to the engine. An optical head was fabricated which allowed access for photography with only a slight change in the curvature of the internal geometry of the head, but no change in the compression ratio. The head was a three part design consisting of two aluminum pieces sandwiching a clear acrylic window. See Appendix F for an engineering drawing of the head and Figure 2.2 for a photograph.

One reason that the Kawasaki engine was chosen is that in production form, the combustion chamber was of the small pancake variety coupled with a dished piston. This allowed the transition to a flat optical head without much perturbation to the original design. The clear acrylic head could easily be replaced with quartz for combustion research at a later date.

To allow the laser grid to penetrate into the cylinder the back of one main transfer port and the back of the boost port were replaced with one sixteenth inch thick quartz windows. This alleviated the need to modify the cylinder walls, but at the same time limited the studies to looking at scavenging near bottom dead center (BDC). This, however, is an area of primary interest during scavenging. Further port modifications included epoxying shut the auxiliary exhaust ports and setting the exhaust port height adjustment permanently in the lowest position. This had no adverse affect on the flow patterns in this experiment since at low RPM these ports were in this configuration anyway.

A large flywheel, effectively doubling the rotating inertia of the stock engine, was installed to allow smooth motoring of the test rig. Opposite the flywheel, on the other end of the crankshaft, was bolted a crank angle degree wheel used for setting up each experiment.

Driving the engine was a ten horsepower eddy current motor with a variable clutch drive. Coupled with the flywheel, this allowed effective motoring speeds to range from 50 RPM to 1500 RPM using one to one gearing. The engine was driven with a drive shaft bolted directly to the crank. On this shaft was also a takeoff pulley for the crank angle encoder. A schematic of the complete experimental setup is shown in Figure 2.3.

2.2 Laser and Optics

The optical setup illustrated in Figure 2.4 was used to create the grid of laser lines inside the engine. The beam dividers were developed specifically for the task of transforming one large beam into several small beams. This technique has the advantage of using all of the incident laser energy. The design of the beam divider is a "stairstep" arrangement of mirrors resembling an oversized diffraction grating. Whatever light does not reflect off of the first mirror is passed onto the second mirror and so on until either all of the energy in the beam is depleted or the mirrors come to an end. The mirrors on the beam dividers were coated with aluminum for reflectance and silicon dioxide for durability. The coating was optimized to an angle of incidence of 75 degrees from the vertical and a laser wavelength of 308nm. The bases were made of steel. Each stairstep is 0.100" wide and was machined at a six degree angle. The mirrors were attached to the base with a slow drying silicone RTV adhesive. This type of adhesive allowed each mirror to be individually aimed for optimum performance.

Diffraction effects were investigated using a Helium-Neon laser. Figure 2.5 shows diffraction patterns for three beams. The fringes are fairly weak because the corners of the mirrors are not sharp. Practice has shown that none of the fringes are strong enough to contaminate the grid pattern. This is partly due to the fringes being even weaker under 308nm incident light conditions compared to the 633nm conditions of Figure 2.5. Figure 2.6 is a closeup of these optics.

The remaining optics used to deliver the laser grid consisted of a 50:50 308nm dialectrically coated beam splitter, two 308nm dialectrically coated mirrors, and when applicable, double convex quartz lenses that ranged in focal length from 50mm to 150mm. Note that these are not shown in the illustrations. When needed they were placed between the beam dividers and the engine.

The laser used was a Lambda Physik LPX 220 pulsed excimer laser. The XeCl gas charge produced ultra violet light at 308nm. The initial beam size was 5mm by 20mm, and each pulse carried up to 220mJ of energy over a period of time of 20ns.

CHAPTER 3
EXPERIMENTAL PROCEDURE

3.1 The LIPA Technique

LIPA was previously conducted successfully in media such as water and kerosene [19,20] where phosphorescent chemicals may be dissolved easily. Using this technique in gas, however, posed some different challenges. One problem focused on particles following the flow. If phosphorescing solids were introduced into the flow the problems of flow conformity, static charge, seeding, and high fluid density would be present. These were especially important to avoid in an engine environment where turbulence length scales are small, and moving parts can cause static charge build-up. Another problem was faced when phosphorescent gaseous mixtures were investigated. One drawback common to nearly all of these chemicals is that the phosphorescence was quenched in the presence of oxygen. This was fairly easily addressed in a non-firing engine by using nitrogen as the carrier gas. Therefore, the gas mixture of nitrogen and biacetyl (2,3 Butanedione) was chosen for this experiment. The mixture density compared to air was 1.1 at STP. The nitrogen and biacetyl mixture was created by bubbling nitrogen through liquid biacetyl. Since biacetyl has a low vapor pressure (0.06868 atm) it evaporated very easily. A schematic of this process is shown in Figure 3.1.

During the 20ns that the laser was on, the biacetyl absorbed energy along the undistorted grid lines. After the pulse was completed the phosphorescing grid of fluid deformed with the fluid motions. Grid intersections are the key to this technique. Each intersection is a fluid particle marker, and a temporal sequence of grid images is used to measure velocities and velocity gradients over the plane of the grid. This data is also unbiased by out of plane motions as long as the grid stays in the depth of field of the camera lens, and the lens size is similar to the measurement grid size in order to eliminate any associated parallax errors. At f/1.2 the depth of field of the lens used in this experiment was approximately 1mm — more than enough to capture any out of plane grid motions. The instantaneous velocity data was used to estimate vorticity, and averages over the area of the measurement grid, as well as averages at any given point over many cycles, allowed calculations of turbulence intensities and Reynolds stresses to be made. Low pass spatial filtering of the velocity field could be used to find the swirl. The above grids were recorded using a gated, intensified CID array video camera manufactured by ITT. Its resolution was 512 X 760 pixels. The raw data, stored on 1/2 inch video tape, was then downloaded into a Megavision 1024 XM image processor. The data consisted of two types of grids: undistorted grids and distorted grids. The undistorted grids were captured as the laser was firing, and the distorted grids were captured by the camera a specified time delay after the laser fired. The data must not only be timed with the laser, but also with the engine. This was accomplished with a crank angle encoder and accompanying circuitry. A schematic of this circuitry is located in Appendix F. At the desired crank angle a TTL pulse was sent to a Phillips PM 5712 pulse generator and simultaneously to the laser firing switch. While the laser fired at the presence of this signal, the pulse generator created another TTL pulse that began a specified time delay later and lasted for a specified duration. This second pulse was used to gate the camera. Its delay corresponded with how much grid distortion was desired, and the duration of it controlled how long the camera shutter stayed open. Typical delays and durations were on the order of a fraction of a millisecond. Figure 3.2 is a time line illustrating this process.

The purpose of the image processor was to aid in locating intersection points and subsequently store them on disk. The algorithm used to find the velocities and other quantitative information centered around the "grid box." A four sided grid box is illustrated in Figure 3.3. This figure also illustrates how two consecutive sets of data, separated by a small time interval, were used to calculate the velocity at each corner.

Note that this is an average velocity over the distance that separates the two points. For this reason the delay timing and the grid mesh size should be correlated so that the grid does not deform more than ten percent of the length of any grid box (see Appendix C).

Higher order fluid mechanical quantities, like vorticity, may be calculated by differencing along the length of a grid box. Using the results of this differencing, vorticity may be calculated using the following equation.

$$\omega_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{1}$$

However, a preferable alternative [3] if one is only interested in vorticity, (that is well suited to the type of data presented here) is calculating vorticity using the definition of the circulation.

$$\Gamma = \oint \vec{V} \bullet d\vec{S} \tag{2}$$

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

$$\Gamma = \int \vec{\omega} \bullet \hat{n} \; dA \tag{3}$$

As illustrated in Figure 3.4, the quantity V•dS is estimated by taking the average of the corner velocity components, multiplying that value by the direction cosine of the included angle between the grid box side and the velocity vector and summing in a counterclockwise direction.

Dividing Γ by the area of the grid box results in the average vorticity at the centroid of this grid box. Turbulence intensities and turbulence energy values, traditionally included in hot wire and LDA studies, were calculated using the average velocity at each point across the cyclic ensemble ($<Ui>_C$ and $<Vi>_C$). 'i' will be used to index the range of points within the jth picture, i = 1, n, and 'j' to index the range of cycles covered, j = 1, m.

$$u'_{rms} = \sqrt{\overline{u_i^2}} = \frac{1}{m}\sum_{j=1}^{m}(u_{i,j} - < U_i >_c) \tag{4}$$

$$v'_{rms} = \sqrt{\overline{v_i^2}} = \frac{1}{m}\sum_{j=1}^{m}(v_{i,j} - < V_i >_c) \tag{5}$$

The average turbulence energy field was calculated from the previous quantities:

$$\overline{q_i} = \frac{1}{m}\sum_{i=1}^{m}\sqrt{u_i'^2 + v_i'^2} \tag{6}$$

Reynolds stresses (turbulent shear stresses) were calculated using two different methods. The first method used the spatially averaged velocities in each frame, $<U_j>_s$ and $<V_j>_s$, and the second used the average velocities at each point across the cyclic ensemble, $<U_i>_c$ and $<V_i>_c$.

$$< u'v' >_{s,j} = \frac{1}{n}\sum_{i=1}^{n}(u_{i,j} - < U_j >_s)(v_{i,j} - < V_j >_s); \ \ j = \text{constant} \tag{7}$$

$$< u'v' >_{c,i} = \frac{1}{m}\sum_{j=1}^{m}(u_{i,j} - < U_i >_c)(v_{i,j} - < V_i >_c); \ \ i = \text{constant} \tag{8}$$

Comparing these values provides an excellent test of cyclic variability. Reynolds stress was chosen as the quantity to conduct this test on because it contains the most valuable information in the study of scavenging flows — momentum transport. If cyclic variability was not present, the above values should be of the same order for this finite sample, however, if cyclic variability was present then large differences could appear. Note that the sample size of only thirty cycles is too small to be fully confident in the results, however cyclic variability shown by other means later proves consistent with the results presented using this technique. A grand ensemble average (across space and cycles) was also computed for each flow variable (velocity, vorticity, Reynolds stress), and these values were used to indicate the variation in the spatially averaged quantities.

$$V_{\text{grand ensemble}} = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left|\vec{v}_{i,j}\right| \tag{9}$$

$$\omega_{z\ \text{grand ensemble}} = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(\omega_z)_{i,j} \tag{10}$$

$$u'v'_{\text{grand ensemble}} = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(u'v')_{i,j} \tag{11}$$

where $\left|\vec{V}_{i,j}\right|$ is the velocity magnitude at a point in a cycle. The grand ensembles consisted of (30 frames x 44 points/frame) = 1320 points. Finally, contour maps of the velocity field for each individual frame were investigated, and the ensemble was split up into two different groups: those corresponding to the ensemble average of the information, and a subset containing significantly different flow patterns. This selection of deviant flow patterns is an entirely subjective exercise, but its purpose was to uncover phenomena that would be averaged out by conventional measurement techniques.

3.2 The Experiments

For the data sets presented here an overall measurement area of approximately 19mm x 18mm was used. Contained in this area were thirty grid boxes of average size 3.3mm x 3.3mm. The largest grid box was approximately 3.8mm x 4.0mm and the smallest 2.6mm x 3.0mm. The size differences in the grid box stem from the divergence of the incoming laser beams. In order to cover as much area inside the cylinder as possible while keeping window size small, the grid lines were focused down and allowed to diverge inside the engine. This grid scale proved to be a very good size to work with in this experiment. While the smallest scales of turbulence are not measured (the grid mesh size is on the order of one integral scale [14]) the scales of motion important to studying scavenging, which range in size from the integral scale to scales proportional to the geometric boundaries of the engine were measured easily. Smaller scales would be of more interest only in studies nearer to TDC where combustion is important. A photograph of four raw data frames showing the grid line intersections can be seen in Figure 3.5.

The thickness of the grid plane was of the same order as the width of each beam (on average 0.55mm). The grid plane was located 4mm above the crown of the piston when it was at bottom dead center. This corresponded to 4mm above the lower edge of the transfer ports (the average height of the ports is 11mm), or just below the center line of the port openings. All data were taken when the piston was at bottom dead center. The camera was placed above the engine looking down parallel to the axis of the cylinder. The delay between laser firing and shutter opening was set at 0.14ms, and each image was captured on a single video frame. The upper engine speed for capturing a frame every cycle is 1800RPM (current video framing rate limitation). At higher speeds, circuitry could be developed which would only allow every second or third cycle to be recorded.

The estimated biacetyl concentration in nitrogen was 5%. This mixture was delivered to the engine at the rate of 310cm^3/sec at 150RPM. This translates into a theoretical scavenging ratio of 0.8. The mixture was delivered to the engine passing through the stock carburetor, and the throttle was held 100% open.

CHAPTER 4

RESULTS

4.1 150RPM

Cyclically averaged fields of velocity, vorticity and Reynolds stress, averaged over 30 cycles, are shown in Figure 4.1. The schematic on the right hand side of the page shows the orientation of the measurement grid with respect to engine geometry. The velocity field shows flow from the transfer ports meeting at the center of the measurement region and turning toward and away from the boost port. This sets up a stagnation region just below the center line of the transfer ports. The flow toward the back of the cylinder from the stagnation region could be either returning to the boost port, or it could be flowing back and up the rear wall of the cylinder. These cyclic averaged velocities ranged in magnitude from 0.2 to 3.7 m/sec with an average value of 3.2m/sec, whereas instantaneous velocities ranged from 0.05 to 11m/sec. The vorticity map indicates that most of the large vortical motions and large gradients of vorticity are located around the periphery of the stagnation region. Vorticity magnitudes range from -1008 to +1085/sec, and the majority of the vorticity is grouped into three large vorticies. Examination of the Reynolds stress contours indicates four local regions of high Reynolds stress with values ranging from -3.8 to +4 m^2/sec^2. These values, normalized with the grand ensemble velocity, are an order of magnitude larger than expected from plane shear flows such as turbulent jets [21].

The cyclically averaged turbulence intensity and turbulence energy fields are illustrated in Figure 4.2. The average value of u'_{rms} is 2.35m/sec and it ranges from 0.94m/sec to 4.10m/sec. Note the large concentration of u'_{rms} in the center of the cylinder where the main transfer port jets collide. The v'_{rms} mean is 2.69m/sec and it ranges from 1.47m/sec to 4.62m/sec. Turbulence kinetic energy, q_{rms}, ranges from 2.61m/sec to 4.71m/sec and its average is 3.70m/sec. Cyclic variability is illustrated by the data of both Figure 4.3 and Figure 4.4. Figure 4.3 shows how the average magnitude of each quantity varies with respect to its grand ensemble average. Deviations from the grand ensemble in the total velocity (3.2m/sec) range from -.67 to +1.1 m/sec. Likewise, the vorticity deviated from -400 to +480/sec about its grand ensemble average of 90.8/sec, and the Reynolds stresses deviated from -4 to +3m^2/sec^2 about its grand ensemble average of 0.063m^2/sec^2. Comparing these results to Figure 4.4 clearly illustrates that at BDC this engine's velocity field exhibits large variations from cycle to cycle. Figure 4.4 shows three velocity maps taken from consecutive cycles. The large changes, including complete reversals in direction, graphically illustrate the reasons for the variation indicated in Figure 4.1. The phenomena in Figure 4.4 is investigated further in Figure 4.5 where the data ensemble has been separated into three groups. Figure 4.5a is the entire cyclically averaged velocity field, Figure 4.5b is a subset which contains velocity fields that are similar to the cyclic mean, and 4.5c is an average of velocity fields which deviated in some obvious manner. The ensemble of Figure 4.5c consists of approximately 20% of the data frames, and it shows a distinct looping of the flow back toward one of the transfer ports. This phenomena could possibly be evidence of a wake caused by the transfer port partition.

CHAPTER 5

DISCUSSION

5.1 Engine Investigation

The existence of a stagnation region and of the flow back toward the transfer ports are phenomena that are deleterious to good scavenging. The obvious oscillation of the flow, as evidenced by the switching position of the stagnation region in the ensemble subsets provides the kind of insight that may prove useful in designs for improved scavenging. Also, the use of Reynolds stress data to measure regions of high momentum transport has proven useful. Figure 4.2 explains why looking at a Reynolds stress map is so important. Regions of high intensity and regions of large gradients in 4.2a,b and c all show up as high regions of Reynolds stress in Figure 4.1c. Because Reynolds stress is the best indicator of momentum transport, the plots of u'_{rms}, v'_{rms} and q_{rms} are not as useful for studying scavenging flows. What is not visible by looking at intensities or energy levels alone is the non-connectivity of momentum transport in certain regions of the flow. The fact that there are centralized areas of high transport separated by areas of low transport over the region indicates that there is no correlation between the transport in these regions.

The velocity fluctuation data used to calculate turbulence energy and Reynolds stress data was derived from a classical Reynolds decomposition as presented in Chapter 3. While the periodic nature of engine flows suggests that this may be done, there are still components to the fluctuation that are never filtered out using this technique. In an internal combustion engine it is generally accepted that the bulk velocity changes from cycle to cycle, therefore, the fluctuations automatically have at least two contributing sources; fluctuations in the bulk velocity and the turbulence. This is precisely the reason why an attempt was made to learn more about the nature and size of so called cyclic variations through Reynolds stresses based on spatial as well as cyclic averages. Another consideration should also be studied following this same logic. If enough cycles of data were taken so that certain statistical patterns surfaced, then it might be concluded that these overlaying events should be included in the decomposition of the flow — much like would be done in the flow behind a propeller where an underlying sinusoidal profile must be filtered out so that it is not included in the turbulence intensity measurements.

The size of the grid mesh also is very important to what kind of information is derived from LIPA measurements. The integral scales of motion for an engine near TDC have been universally measured to be on the order of 3mm [14]. Near BDC the integral scale should be somewhat smaller because of the great energy of the issuing scavenging port jets. If the microscales of the turbulence are then another order of magnitude smaller, that would dictate that the grid mesh size be considerably less than 1mm square if all of the details of the flow are desired. However, it is somewhat debatable if that much detail is important for the bulk of scavenging measurements. Imperative in this type of scavenging experiment is the acquisition of data which represents the motions responsible for moving quantities to fluid toward the exhaust port. Scales of this nature range from integral scales to scales on the order of the cylinder size. This is not to say that measurement of smaller scales would not be important. Experiments which looked at detailed mixing along the scavenging front would require an appropriately smaller grid size than an experiment designed to look at bulk scavenging flows.

5.2 Improvements and Future Considerations of This Project

5.2.1 Biacetyl Delivery

The simplistic biacetyl delivery system shown in Figure 3.1 was effective for initial experiments such as these, but it should be improved on. Its main drawbacks were uneven delivery and unknown concentration. Figure 5.1 shows a system that could solve both of these problems.

Biacetyl use could be measured very accurately with a typical automotive fuel delivery system as shown in Figure 5.1. Concentration could also be monitored, and the use of a heated evaporation plate could improve mixing and concentration gradients of the nitrogen - biacetyl combination as it entered the engine.

5.2.2 Laser Grid Generation

The current use of beam dividers to form the laser grid has one main advantage — all of the incident light is used to create the grid. Another method of creating laser lines is through the use of a diffraction grating. This idea was dismissed in the past because a portion of the incident laser beam is completely blocked, but it should be investigated again because of recent improvements in lasers and diffraction grating manufacturing techniques. Today, lasers are powerful enough to afford some losses, and diffraction gratings are much more accurately cut.

Gratings have a large advantage over beam dividers because beam widths and spacing can be adjusted easily, they are relatively cheap, and they take up considerably less space. The possibility also exists for much smaller and more powerful grid lines. Gratings use the properties of light to create a grid rather than forcing the light into a desired shape as with beam dividers. The process is cheaper, easier, and more space efficient.

Another technique that could prove especially useful in engine environments for introducing the laser grid is the use of fiber optics. A group of fibers can be bundled on one end to allow coupling of the laser. After the bundle fibers split off and are routed through the cylinder wall in such a fashion that a grid is created. Lenses on the end of each fiber optic would create collimated beams of light.

The use of fiber optics to deliver the grid would eliminate even the small changes made here in the geometry of the production engine. Actually an attempt to use fibers was initially made, but further development time is needed to refine this approach. The use of fiber optics may in the long run prove to be one of the most valuable aspects of the technique.

CHAPTER 6

CONCLUSIONS

6.1 Engine

- About 20% of the time there appears an unsteady eddy that travels in and out of the field of view. This looping of the flow back toward the transfer ports appears to be created by the transfer port partition, and an unfavorable pressure gradient from the crankcase.

- On this level (very near the piston crown) flow near the boost port is entrained backward toward the port. Since the boost port jet is aimed upward it was assumed that this cylinder flow was being entrained back and up the wall of the cylinder.

- The cyclic variability near BDC in a small two-stroke is very significant as can be seen by Figures 4.3 and 4.4.

- The correlation between spatial averages of Reynolds stress and vorticity plotted from cycle to cycle (Figure (4.3)) suggests the importance of vortical motions in momentum transport and particularly in the mixing that is occurring.

- On average, the velocity vector field picture for the engine looks as one would predict, but the individual frames are very different.

- The regions of high Reynolds stress along the meeting axis of the transfer port jets indicates much momentum transfer (this infers mixing). LIPA could be used here as a design tool to tune the amount of exhaust gas dilution and create better scavenging fronts.

- $\langle u'v' \rangle_C$ was an order of magnitude greater than $\langle u'v' \rangle_S$ on average. This also indicates large cyclic variability.

- It is possible that the very large variations found in this experiment are associated with differences to be expected between motored and fired engines. The lack of high pressure in the cylinder as the exhaust port is exposed will of necessity create significantly different residual flow fields.

6.2 LIPA

- These experiments were conducted in an entirely gaseous environment with properties very near to those of air.

- The data were taken in an engine that was only slightly modified from its production form.

- Analysis of the data is nearly automatic when using image processing command files to locate grid intersections (i.e. large ensembles can be processed very quickly).

- Simple software on an ordinary PC can be used to calculate all fluid mechanical quantities and statistics.

- LIPA can be used simultaneously with LIF (Laser Induced Fluorescence) to study areas such as fuel injection droplet atomization. LIPA also has the capability to work simultaneously with an Exciplex system.

- In the future LIPA may be used with an X-ray laser to determine flow fields within unmodified metal parts.

- LIPA may be expanded to three dimensions using two grids spaced one grid mesh apart. Two cameras must be used to record the images, but the full three dimensional information contained in the region between the parallel grids is obtained with only a factor of two increase in processing time and storage.

APPENDIX

APPENDIX A

REFERENCES

1. G.P.Blair and R.G.Kenny, "Further Developments In Scavenging Analysis for Two-Cycle engines," SAE Paper 800038, 1980.

2. D.S.Sanborn, G.P.Blair, R.G.Kenny and A.H.Kingsbury, "Experimental Assessment of Scavenging Efficiency of Two-Cycle Engines," SAE Paper 800975, 1980.

3. M.E.G. Sweeny, R.G.Kenny, G.B.G.Swann and G.P.Blair, "Single Cycle Gas Testing Method for Two-Stroke Engine Scavenging," SAE Paper 850178, 1985.

4. E.Hashimoto, T.Tottori and S.Terata, "Scavenging Performance Measurements of High Speed Two-Stroke Engines," SAE Paper 850182, 1985.

5. M.Nuti and L. Martorano, "Short Circuit Ratio Evaluation in the Scavenging of Two-Stroke S.I. Engines," SAE Paper 850177, 1985.

6. B. Hopkinson, "The Charging of Two-Cycle Internal Combustion Engines," Trans. NE Coast Instn. Engrs. Shipbuilders, vol. 30, 1914, p.433.

7. D.S.Sanborn and W.M.Roeder, "Single Cycle Simulation Simplifies Scavenging Study," SAE Paper 850175, 1985.

8. M.E.G.Sweeny, G.B.G.Swann, R.G.Kenny and G.P Blair, "Computational Fluid Dynamics Applied to Two-Stroke Engine Scavenging," SAE Paper 851519, 1985.

9. E.Sher, "Modeling the Scavenging Process in the Two-Stroke Engine — An Overview," SAE Paper 890414, 1989.

10. E.Sher, "Investigating the Gas Exchange Process of a Two-Stroke Cycle Engine With a Flow Visualization Rig," Israel J. of Tech., vol. 20, pp. 127-136, 1982.

11. S.Ishihara, Y.Murakami, and K. Ishikawa, "Improvement of Pitot Tube Set for Obtaining Scavenging Pictures of Two-Stroke Cycle Engines," SAE Paper 990171, 1988.

12. B.Ahmadi-Befrui, W.Brandstatter and H.Kratochwill, "Multidimensional Calculation of the Flow Processes in a Loop-Scavenged Two-Stroke Cycle Engine," SAE Paper 890841, 1989.

13. J.G.Smyth, R.G.Kenny and G.P.Blair, "Motored and Steady Flow Boundary Conditions Applied to the Prediction of Scavenging Flow in a Loop Scavenged Two-Stroke Cycle Engine," SAE Paper 900800, 1990.

14.R.A.Fraser, P.G.Felton, F.V.Bracco, and D.A.Santavicca, "Preliminary Turbulence Length Scale Measurements in a Motored IC engine," SAE Paper 860021, 1986.

15. T.Obokata, N.Hanada, and T. Kurabayashi, "Velocity and Turbulence Measurements in a Combustion Chamber of S.I. Engine Under Motored and Firing Operations by L.D.A. with Fiber-Optic Pick-Up," SAE Paper 870166, 1987.

16. M.J.Hall and F.V.Bracco, "A Study of Velocities and Turbulence Intensities Measured in Firing and Motored Engines," SAE Paper 870453, 1987.

17. J.C.Kent, A.Mikulec,Proceedings of the Conference on Applications of Flow Visualization and Measurement,Ford Motor Co.,1989.

18. David L. Reuss, Ronald J. Adrian, Christopher C. Landreth, Donald T. French, Todd D. Fansler, "Instantaneous Planar Measurements of Velocity and Large-Scale Vorticity and Strain Rate in an Engine Using Particle-Image Velocimetry," SAE Paper 890616, 1989.

19. R.E.Falco, C.C.Chu, "Measurement of Two-Dimensional Fluid Dynamic Quantities

Using A Photochromic Grid Tracing Technique," SPIE vol.814 Photomechanics and Speckle Metrology,1987, pp. 706-710.

20. R.E. Falco,C.C. Chu, M.H. Heatherington, and C.P. Gendrich, "The Circulation of An Airfoil Starting Vortex Obtained From Instantaneous Vorticity Measurements Over An Area," AIAA-88-3620-CP, 1988.

21. J. O. Hinze, Turbulence, McGraw-Hill, New York, 1975.

22. R.A. Fraser, P.G. Felton, F.V. Bracco, D.A. Santavicca, "Preliminary Turbulence Length Scale Measurements In A Motored IC Engine," SAE Paper 860021, 1986.

23. John B. Heywood, "Fluid Motion Within the Cylinder of Internal Combustion Engines -- The 1986 Freeman Scholar Lecture," Journal of Fluids Engineering, vol. 109, 1987, pp. 3-35.

24. Gordon P. Blair, The Basic Design of Two Stroke Engines, SAE, 1990.

25. Alfred Jante, "Scavenging and Other Problems of Two-Stroke Cycle Spark Ignition Engines," SAE Paper 680468, 1968.

26. Ramkrishna G. Phatak, "A New Method of Analyzing Two-Stroke Cycle Engine Gas Flow Patterns," SAE Paper 790487, 1979.

27. K. V. Reddy, V. Ganesan, K. V. Gopalakrishnan, "Under the Roof of the Cylinder Head -- An Experimental Study of the In-Cylinder Air Movement in a Two-Stroke Spark Ignition Engine," SAE Paper 860166, 1986.

28. M.E.G.Sweeney, R.G.Kenny, G.B.G.Swann, and G.P.Blair, "Computational Fluid Dynamics Applied to Two-Stroke Engine Scavenging," SAE Paper 851519, 1985.

29.Charles F. Taylor, The Internal Combustion Engine in Theory and Practice, MIT

Press, Cambridge, Mass, 1977, vols. 1-2.

30. Gordon P. Blair (Ed.), Advances in Two-Stroke Cycle Engine Technology, SAE Publication PT-33, 1988.

31. C.F. Beaton, G.F Hewitt, Physical Property Data For The Design Engineer, Hemisphere Publishing Corporation, New York, 1989.

APPENDIX B

DELIVERY RATE CALCULATIONS

APPENDIX B

DELIVERY RATE CALCULATIONS

Because the engine was fed from a compressed bottle of nitrogen and not from the atmosphere, a delivery rate scheme had to be developed. The following equation was used to calculate total swept volume at any engine speed.

$$\frac{X \ \text{Revolutions}}{\text{Minute}} \times \frac{1 \ \text{Minute}}{60 \ \text{Seconds}} \times \frac{124\,cc}{\text{Revolution}} = {}^{cc}\!/_{\text{sec}} \ \text{swept volume} \qquad \text{(B1)}$$

If the engine is assumed to have a uniform scavenging ratio (SR) of 0.8 (analogous to volumetric efficiency in four-stroke engines) then the delivery rate vs. RPM curve shown in Figure B1 results.

At 150RPM used in the experiment a delivery rate of 310cc/sec was used. This corresponds with the above graph, but it was later realized that at this low speed the scavenging ratio would be much lower than 0.8. Because of the lack of exhaust tuning and cylinder blowdown effects at this low RPM, the scavenging ratio should have been between 0.4 and 0.5 [24]. Figure B2 shows the calibration curve for the nitrogen tank/regulator combination used.

APPENDIX C

LIPA DISCUSSION

APPENDIX C

LIPA DISCUSSION

Taking data at realistic engine speeds is very important if LIPA is to become a valuable tool in engine research and design. However, there are a few hurdles to overcome before this can become a reality. Just as with any new measurement technique, present technology plays an important role in how well it can be implemented. This section will discuss several areas of importance in improving LIPA for future uses in or out of engine environments.

LIPA requires an image of a distorted grid as well as an undistorted grid to comprise a data set. Photography of the undistorted grid is never a problem (it can be done with no fluid motions in the cylinder), however, capturing the distorted grid on film or video tape is much more difficult. Several factors either increase of decrease the chances of successfully photographing a distorted grid. These include engine speed, flow speed, camera delay, camera duration (= exposure time), phosphorescence vs. time characteristic of the seed chemical, quantum efficiency of the seed chemical, grid size, laser power, and how sensitive the recording device is. Note that several of the above are closely correlated.

C.1 Camera Delay

The delay between the start of the laser pulse and the camera shutter opening is only a function of flow speed: This small amount of time, which is used to calculate the absolute velocity at each grid intersection, is adjusted so that grid distortion is kept to near ten percent of the average grid mesh size. This rule of thumb helps to insure a level of linearity in the distorted grid.

C.2 Camera Duration

The duration, or exposure time, can be the most important factor governing the accuracy and even feasibility of LIPA. The shorter the duration the better. Ideally, an instantaneous snapshot of a distorted grid would produce the most accurate results, but time is required to obtain a usable image. This amount of time, however, cannot be so long that the image of the distorted grid is extremely blurred (a "time exposure" effect). The duration, like the delay, then is largely a function of flow speed. Below is a chart showing some representative durations calculated using two rules developed through experience. The first, mentioned before, is ten percent grid distortion. In this case a grid size of 5mm X 5mm was used as an example. The second rule is to keep the length of the duration to within twenty percent of the delay. This keeps the image of the grid sharp. Note how small the durations become for even moderate speed flows.

Table C1. Comparison of flow speed with delay and exposure times

Flow speed	Delay to ensure 0.5mm (max) movement of grid	Exposure time (1/5 of delay)
5m/sec	0.1ms	0.02ms
10m/sec	0.05ms	0.01ms
20m/sec	0.025ms	0.005ms
30m/sec	0.0167ms	0.003ms
40m/sec	0.0125ms	0.0025ms

C.3 Dynamic Range

In a mainly unidirectional flow, such as pipe or jet flow, high velocities are not a problem because the entire grid can shift with the bulk flow. However, in an engine there is no preferred flow direction. This means that very high gradients can be present. In a two-stroke engine, velocities may vary from 0m/sec (a stagnation region) to as high as 100m/sec. High regions of shear are common in the compilation of jet flows that make up the total scavenging picture. The challenge lies in successfully capturing the full range of velocities with a single measurement. Should the delay correspond to the higher speed portions of the flow and leave the grid in the region of lower speed flow nearly undistorted? Should the delay correspond to the lower speed portions of the flow and leave the grid in the regions of higher speed flow grossly distorted? What happens to the accuracy of the measurements when the range of velocities is great? These are a few of the questions that will be answered in the near future and are actually best answered in an engine environment. These experiments have shown that LIPA in its present form can support the dynamic range of a two-stroke engine motored at 150RPM. This is very encouraging considering the present state of development of the phosphorescent seed chemicals and the optics.

C.4 Image Recording

While the duration ideally is only a function of flow speed, it is also a function of several other factors. The duration must be of sufficient length to allow enough photons to pass into the recording device to make an image. Therefore, in practice, duration is adjusted to the shortest time possible to allow a distorted grid image to be captured. If this happens to be shorter than twenty percent of the delay, that is acceptable, but (as with the data presented here), that is most often not the case with technology at its present level. The grids in Figure 3.5 are slightly blurred because the delay and duration used were of the same order of magnitude.

Several performance oriented issues about the laser, the optics, and the phosphorescent seed chemical are important concerning this issue. The laser should be powerful enough to fully energize the grid lines in the seed chemical, and the optics should be high quality to minimize losses. However the most important factor at this point is the seed chemical. Each chemical has two key parameters regarding phosphorescence that are important to LIPA. The first is quantum efficiency. This is simply the ratio of incident laser energy absorbed to how much energy is released in the form of photons.

$$\phi = \frac{\text{Laser Energy Absorbed}}{\text{Photon Energy Released}} \tag{C1}$$

Because this energy is released over time it is also important to be aware of the phosphorescence vs. time characteristic as illustrated in Figure C1.

Figure C1 illustrates why this characteristic is important. The shaded area represents the slice of energy available for image recording. If this curve falls off quickly, and large delays are required (i.e. low speed flows), then it is conceivable that not enough energy would be left for image recording. High speed flows actually have an advantage in that they must always be recorded in the "fatter" part of the characteristic.

C.5 Piston Motion and Measurement

In an engine environment data are often required at specific crank angles. It is then important to know how much piston motion occurs while a distorted grid is being recorded. The amount that the piston moves is dependent on engine dimensions, RPM, crank angle, and camera duration. If a linear relationship between RPM and flow speed (and thus camera duration) is assumed for any specific crank angle then the curve in Figure C2 results.

The bore to stroke ratio of the engine has a direct effect on piston speed. Engines with a relatively long stroke generate higher piston velocities. The piston reaches its maximum velocity at mid-stroke and comes to a complete stop at TDC and BDC. To simplify calculations an average piston speed was used. Since all of the LIPA measurements were taken at BDC, a slight over estimation in piston movement was calculated. Engine dimensions were taken from Table 1.

For example, at 400RPM the engine undergoes 6.67 revolutions per second. This translates into an average piston speed of 607.2 mm/sec. For a camera duration of 0.03ms the piston then moves a total of 0.018mm. This equals 0.065° of crank rotation, Therefore at 400RPM the crank angle resolution is within one tenth of a degree — sufficient resolution for this type of study.

Because of the nearly linear relationship between RPM and scavenging velocities [27] this scale of resolution should stay roughly constant throughout the RPM range. However, while these values are an over estimation near BDC and TDC, they will be gross under estimations near the middle of the stroke.

C.6 Other Considerations

C.6.1 Small Grid Size

The smallest important length scales measured in a turbulent engine flow are the Kolmogoroff scales. Previous measurements have measured them to be on the order of 0.05mm [18]. It is possible, with the correct optics, to produce a grid with a mesh size this small. Decreasing the size of the grid, however, increases the difficulty of recording it on film or video tape because camera duration must be decreased for two reasons.

The primary reason duration must be decreased is because less grid motion is acceptable. Distortion should still be kept to ten percent of a grid box length. Secondly, the lines making up the grid must be thinner. Both of these reasons mean fewer photons will be available to record an image, therefore, even more emphasis must be placed on developing better phosphorescent chemicals and more sensitive recording equipment.

C.6.2 Laser Pulse Width

The laser energizes the seed chemical over a time interval of about 20ns. The question is: how important is it that this process does not take place instantaneously? The answer is straight forward. The shortest realistic camera delay in an engine would be on the order of 0.1μs. This would be for capturing velocities of near 50m/sec with a grid mesh size of 1mm. 0.0001ms is two orders of magnitude larger than the laser pulse width, therefore, the laser will not affect the accuracy of LIPA even under extreme conditions.

APPENDIX D

COMPUTER PROGRAMS

APPENDIX D

COMPUTER PROGRAMS

D.1 Documentation

The following are descriptions of the computer programs used in reducing the raw data from that which is pictured in Figure 3.5 to hard-copies like Figures 4.1 through 4.5. Before any of these programs are used, however, the raw grids must be reduced to data files that contain only grid intersection points. These are referred to as *.pts files here.

Reducing the raw data to point files may be accomplished in one of two ways. Since automation of LIPA is of primary importance when using large ensembles, a command file which automatically locates grid intersections should be used. However, if the grids do not have enough contrast with the background, or if there is a considerable amount of grid distortion, then the grid intersections must be found manually. In these experiments intersections were located manually using the Megavision 1024 XM routine SAMPLE.

A short description showing what order the following programs should be used in order to achieve different types of results is given. Also a diagram showing the structure of the main program VORTICITY is shown in Figure D1. Note that * is used as a wildcard filename in all of the program descriptions.

The following software falls into two categories — initial processing and post processing. The initial processing software centers around the program VORTICITY. It uses undistorted and distorted grid intersection data (*.pts files) to calculate velocities, vorticity, and Reynolds stress based on spatially averaged velocities. The post processing software is then used to calculate averages of the above, Reynolds stress based on cyclically averaged velocities, velocity fluctuations and turbulence energy, and hardcopies of velocity vector fields.

Below is a step-by-step set of instructions for the usage of these programs beginning with initial processing. It assumes that the user has already compiled VORTICITY and that the raw data is in the *.pts format. It also assumes that a polygon layout has been defined for each data frame so that polygon descriptor files may be written (see program VORTICITY).

(1) Execute VORTICITY and answer all questions that the program asks.

(2) When asked about what polygon descriptor file is to be used, either enter the data as prompted or enter the name of a previously defined polygon descriptor file.

(3) General output filenames must now be changed to specific names for each grid. For example:

vel.q ---> grid*.vel

vort.out---> grid*.vort

uvave.out ---> grid*.uvave

UV.out ---> grid*.UV

(4) Run the GNUPLOT* graphing utility and get an initial plot of each velocity vector field. The following commands will produce a screen plot of the vector field.

> load 'gnuplot.aro'

> plot 'gnuplot.pts1' with dots

Both of these files are output from VORTICITY. Other commands within GNUPLOT allow customization of output and hardcopy generation.

*GNUPLOT is a shareware graphing utility available for nearly all operating systems.

(5) Repeat each step until all of the data is reduced. Note that with very large data sets that a batch file could be created to automate this process.

The post processing software is well enough explained in the program description section with the exception of the vector field generation programs. These programs are used to create publication quality velocity vector field hardcopies by allowing scaling of the vectors. Two sets of programs are presented. The first set is used for creating hardcopies of instantaneous vector fields. These plots will be scaled versions of those created in step four above. The second set is used for creating vector plots of average ensembles. Below is a description of the use of these programs.

Instantaneous vector plots:

$$\text{Megavision SAMPLE output} \xrightarrow{(*.pts)} \text{Program VECTOR}$$

(gnuplot.aro)

GNUPLOT

Instantaneous vector field hardcopy

Averaged vector plots:

Megavision SAMPLE output

(*.pts)

Program AVERAGE

(avg.stats)

Program VELPLOT

(*.aro)

Program VECTOR2

(gnuplot.aro)

GNUPLOT

Averaged velocity vector field hardcopy

D.2 Program Descriptions

D.2.1 Program VORTICITY

Description:

This program reads two data files produced by the "SAMPLE" function of the Megavision 1024 XM. The first frame must always be an undistorted reference grid, and the second frame is the same grid photographed some time delay later. The user must input the data points in the same order from each data frame. Unreadable data points must be entered as 0,0 and the program will dismiss the data automatically.

This program performs a number of tasks:

1) reads the data
2) throws out bad data points
3) interpolates in space and time to construct a velocity field throughout the frame
4) uses the velocity information coupled with user supplied polygon descriptor files (see *.pfl) to determine vorticity.
5) calculates instantaneous Reynolds stresses at each point.

Input files: (undistorted grid point file (*.pts), distorted grid point file (*.pts), polygon descriptor file (*.pfl))

The *.pts files must be in the Megavision "sample" format.

Output files: (#######, gnuplot.*, see Program velocities)

Polygon descriptor files:

Because this series of programs requires the user to put the point files in the same order for each data frame automatic polygon generation is not present. Since there is an option to throw out bad data points, there will necessarily be different polygon formations for each frame which has a unique point format. The user must construct the best polygon layout by hand (each having four vertices), and enter the four corner points which make up each tetrahedral into the *.pfl file. Each polygon should be entered on a line. For example, if points 1, 2, 8, 9 make up the first polygon and points 2, 3, 9, 10 make up the second polygon, then the first two lines of the file should read as follows:

1,2,8,9

2,3,9,10

The following programs that use this information to calculate circulation and vorticity will then process each polygon as they are listed in this file.

Authors: Hilbert and Gendrich

D.2.2 Program AVERAGE

Description:

Read in a number of Megavision OBJECT files and calculate the average location for each point.

"prior information" -- The first file must contain as many points as there are. If a point shows up in any subsequent file which has been chosen as bad by the user (see Program vorticity), it is discarded.

Input file: (avg.dat)

An input file is required in which operating parameters are specified. It should contain the following:

# comment	
# comment	
Average velocity output filename	-- for the average frame
stats output filename	-- for stats
"prior information" frame name	-- for undistorted points
data frame 1 name	-- measurement 1
....	-- measurements 2,3,...
data frame n name	-- final measurement frame

Output files: (user chosen in the above file)

The average velocity output can be used later to calculate Reynolds stresses etc. The statistics output file can be used to check data and make sure it isn't out of control.

Authors: Gendrich and Hilbert

D.2.3 Program VORTAVE

Description:

Read in a number of vorticity.F output files (grid*.vort) and calculate average centroid locations and average vorticies.

Input file: (vortave.dat)

An input file is required in which operating parameters are specified. It should contain the following:

```
# comment
# comment
output filename       -- filename of your choice
data frame 1 name     -- measurement 1
      ....            -- measurement 2, 3, ...........
data frame n name     -- final measurement frame
end                   -- end of file marker
```

Output file: (user chosen in above file)

The output file will contain averaged centroid locations and averaged vorticies for those input files which contain full data sets (no bad points). Frames without all grid boxes present must not be included in the input file.

Author: Hilbert

D.2.4 Program STRESS

Description:

Read in a number of vorticity.F output files (grid*.vel) and calculate u'v' and u'v'bar for each intersection.

Input file: (stress.dat)

An input file is required in which operating parameters are specified. It should contain the following:

```
# comment
# comment
output filename        -- filename of your choice
avgvel.dat             -- average values for all frames
data frame 1 name      -- measurement 1
     ......            -- measurement 2,3,......
data frame n name      -- final measurement frame
end                    -- end of file marker
```

Output file: (user chosen in above file)

This output file will contain u'v' for each frame as well as UV which is the average Reynolds stress for each point taken over all of the frames in the data frame file.

Author: Hilbert

D.2.5 Program UVPRIME

Description:

This program reads in a number of Vorticity output files and calculates u', v' and q=sqrt(u'**2 + v'**2) for each grid intersection. The output file contains u', v' and q for each point averaged over all of the frames in the data frame files.

Input file: (uvprime.dat)

An input file is required in which operating parameters are specified. It should contain the following:

```
# comment
# comment
output filename       -- filename of your choice
avgvel.dat            -- average values for all frames
data frame 1 name     -- measurement 1 (*.vel)
   ......             -- measurement 2,3,...... (*.vel)
data frame n name     -- final measurement frame (*.vel)
end                   -- end of file marker
```

Output file: (user chosen in above file)

Author: Hilbert

D.2.6 Program AveAve

Description:

This program averages a list of data. The format statement must be modified for different input file styles. It will also output an average in a different set of units.

Input file: (The program prompts the user for this)

Output file: (The output is directed to the screen)

Author: Hilbert

D.2.7 Program RMS

Description:

This program takes cyclically averaged vorticity data and calculates a normalized vorticity using a spatially averaged RMS value.

Input file: (vortave.out)

Output file: (normvort.out)

Author: Hilbert

D.2.8 Program VELPLOT

Description:

This program makes Gnuplot "load" files which will plot the average velocity vector field.

Input files: (datum.pts (undistorted point data), avg.stats (average distorted point data))

Output file: (gnuplot.aro)

Author: Hilbert

D.2.9 Program PLOTIT

Description:

This program makes data files that are useful for creating surface or contour plots of spatially dependent data (i.e. vorticity, Reynolds stress etc.). It reads in location data in the form of pixels, converts this data to the proper units, and combines it with any other type of data in an output file. The output file is arranged in a x,y,z format where x and y are location and z are fluid mechanical quantities.

Input files: (Example: Datum.pts and Avg.vort)

Output file: (Example: Avgvort.surf)

Author: Hilbert

D.3 <u>Program Listings</u>

The following listings contain all of the programs and supporting subroutines used to reduce raw LIPA data. All programs were written in FORTRAN.

Program AveAve

```
      program AveAve
c
c
c  This program averages a list of data. The format statement must be
c    modified for different output styles.
c
c  Author:
c    H. Sean Hilbert
c
c
      character*80    avefile,junk
      integer         N,L,i
      real            AA,BB,tota,totb,avea,aveb
c
c  get input filename
c
      write(*,1000)
      read(*,1100)avefile
      open(unit=1,file=avefile)
c
c  get # of data points to average
c
      write(*,1200)
      read(*,1300)N
c
c  get # of header lines
c
      write(*,1400)
      read(*,1300)L
c
      do 100 i=1,L
              read(1,1100)junk
100   continue
c
      tot = 0.0
      do 200 i=1,N
              read(1,1500)AA
              print*,AA
              tot = tot + AA
200   continue
      ave = tot/N
      print*,tot,N
      write(*,1450)ave
      cave = ave*0.0342/0.00014
      write(*,1475)cave
c
c
1000  format(' What is the name of the input file : ')
1100  format(a80)
1200  format(' How many data points to average : ')
1300  format(i3)
1400  format(' How many header lines are there : ')
1450  format(' Average: ',f10.5)
1475  format(' Converted average: ',f12.5)
c
c  change line 1500 for different output format types
c
1500  format(23x,f10.5)
      stop
      end
```

Program AVERAGE

```
        program average
c
c Description:
c       Read in a number of MegaVision OBJECT output files and
c       calculate an average location for each place.
c
c       "Prior Information"--  The first file read in must contain as
c       many points as there are.  If a point shows up in any subsequent
c       file which is not in the neighborhood of our "prior information"
c       points, it is discarded.
c
c Input file:
c       An input file is required in which operating parameters are
c       specified.  This file must be named 'Avg.dat'.  It should
c       contain the following:
c               # comment
c               # comment
c               velocity output filename        -- for the Average frame
c               stats output file name   -- for stats on the averages
c               "prior information frame" name  -- base info
c               data frame 1 name               -- measurement 1
c                       ...             -- measurements 2, 3, ...
c               data frame n name               -- final measurement frame
c
c Author:
c       CHuck Gendrich <cpg>
c
c History:
c  <cpg> 22 may 89 -- v1.0 for crunching the 2nd batch of airfoil data
c       Based on: TestLook v.4aug87, getframe v.aug87
c
c  <hsh> 9 oct 90 -- v2.0 rewrote matching section similar to
c       main.F ver1.5 data entry style
c
# include "vorticity.h"                 /* global variable defs */
# include "stats.h"                     /* stats information */
c
        character*30    infil
        parameter( infil='Avg.dat')     /* input data file */
c
        character*80    line    /* input line */
        character*80    basfil  /* data file containing frame information */
        character*80    datfil  /* data file containing frame information */
        character*80    outfil1 /* output file containing average locations */
        character*80    outfil2 /* output file containing stats on locs */
        integer         datfrm  /* number of data frames read */
        integer         saved   /* number of points saved for each frame */
        integer         mtch    /* good/bad data flag*/
        real            q       /* number of measurements used */
c
        data datfrm/ 0/
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c
c Initialization stuff
        do 10 i=1, MaxPts
                frame1( i,X) = 0.0      /* frame 1 contains the base points */
                frame1( i,Y) = 0.0      /* ...our 'prior information'. */
10      continue
        call init( counter)     /* initialize our statistical counters */
c
c Get operating parameters
        open( 1, file=infil, status='old', err=2000)
```

Program AVERAGE

```
20      continue                        /* Comments are permitted only at */
        read( 1, 1000, end=2010) line   /* the beginning of the file.  */
        if( line(1:1).eq.'#') goto 20   /* Skip them -- '# comment...' */
        read( line, 1000)       outfil1
        read( 1, 1000, end=2060) outfil2
        read( 1, 1000, end=2040) basfil
c
c               note: unit 3 is used for data frames...
        open( 2, file=outfil1, status='unknown', err=2020)
        open( 4, file=outfil2, status='unknown', err=2030)
c
c Check the base data frame ("prior information" about where the
c       intersections should be located.
c       MAKEFRM return values:
c       ierr = 1 -> error opening the frame data file
c       ierr = 2 -> empty file
c       ierr = 3 -> wrong number of lines... three should appear at once --
c       OBJ NUMBER / X-COORDS / Y-COORDS ---> Frame probably incorrect
c
        call makefrm( frame1, basfil, ierr)
        if( ierr.gt.0) goto 2050        /* quit if it's a bad file */
        n = NumPts( frame1, 2)          /* how many points are there? */
c
c Save these points for the start of our stats summation
        do 30 i=1, n
                counter( i, VAL) = frame1( i, X)
                counter( i+n, VAL) = frame1( i, Y)
                call save( counter, i)          /* X value */
                call save( counter, i+n)        /* Y value */
30      continue
c
c Now check out all the other frames mentioned in the input file
40      continue        /* main processing loop begins here */
                do 50 i=1, MaxPts
                   frame2( i,X) = 0.0   /* frame 2 contains points from our */
                   frame2( i,Y) = 0.0   /* data frames... */
50              continue
                close(3)
                read( 1, 1000, end=800) datfil
                call makefrm( frame2, datfil, ierr)
                if( ierr.gt.0) goto 40
                datfrm = datfrm + 1
                saved = 0
c
                do 100 i=1, n   /* for each point in the first frame */
                        if(frame2(i,Y).ge.1019.)then
                                mtch=0
                        else
                                mtch=i
                        endif
                        if( mtch.ne.0) then     /* we have a good hit */
c                               save the data....
                                saved = saved+1
                                counter( i, VAL) = frame2( mtch, X)
                                counter( i+n, VAL) = frame2( mtch, Y)
                                call save( counter, i)          /* X value */
                                call save( counter, i+n)        /* Y value */
                        endif
100             continue
                print*,'Saved ', saved,' points.'
                if( GoOn()) continue
        goto 40         /* and we'll read until the input file is empty */
c
c Done reading data... time to process it
```

Program AVERAGE

```
800     continue        /* calculate the average frame and output stats */
c       Did we read any frames at all?
        if( datfrm.le.0) goto 2070
c
c       Save the comment on this data frame
        open( 3, file=basfil)
        read( 3, 1000) line     /* note that line has to be saved for prntfrm
        close(3)
        write( 4, 1010) line, datfrm
c
        do 810 i=1, n   /* for all points in our base frame */
c
c  HEY!! We really should check UPSIDE_DOWN before doing this!!!
c
c          get X mean and stats
           call stats( counter, i, Xmean, Xvar, Xdev, Xkew, Xkurt, q)
           vel(i,Vx) = Xmean - frame1(i,X) /* output velocities */
           write(2,1030)i,vel(i,Vx)
           frame1( i, X) = Xmean        /* store the mean */
c          get Y mean and stats
           call stats( counter, i+n, Ymean, Yvar, Ydev, Ykew, Ykurt, q)
           vel(i,Vy) = Ymean - frame1(i,Y) /* output velocities */
           write(2,1040)i,vel(i,Vy)
           frame1( i, Y) = Ymean        /* store the mean */
           write( 4, 1020) i, q, Xmean, Xdev, Xkew, Xkurt, Ymean,
    +            Ydev, Ykew, Ykurt
810     continue
1000    format( a80)
1010    format( 'Comment: ',a60//
     +  i2, ' data frames were compared to the base frame.')
1020    format( 'Point ', i2,': ', f4.0,' measurements'/
     +  '   X: ',f6.1,' mean, ',f7.3,' sdev, ',f7.1,' skew, ',f9.1,
     +  ' kurt'/
     +  '   Y: ',f6.1,' mean, ',f7.3,' sdev, ',f7.1,' skew, ',f9.1,
     +  ' kurt')
1030    format(' At point ',i2,' Vxbar is ',f10.5)
1040    format(' At point ',i2,' Vybar is ',f10.5)
c
        call prntfrm( frame1, line, outfil1, ierr)
        close(2)
        close(4)
        stop '%Avg-A-OK: Avgerage normal termination.'
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c       ERROR HANDLING IS DONE HERE
c
2000    continue
        stop '%Avg-F-OPENFAIL:  Error opening input data file.'
c
2010    continue
        write( *, '(a80)') infil
        stop '%Avg-F-ONLY#:  Input file contains only comments'
c
2020    continue
        write( *, '(a80)') outfil1
        stop '%Avg-F-OPENERR:  Error opening MV output file'
c
2030    continue
        write( *, '(a80)') outfil2
        stop '%Avg-F-OPENERR:  Error opening stats output file'
c
2040    continue
        stop '%Avg-F-EOF:  Error reading base data frame file'
```

Program AVERAGE

```
c
2050    continue
        write( *, '(a80)') basfil
        stop '%Avg-F-BADFRM:  Base data frame is bad (makfrm)'
c
2060    continue
        stop '%Avg-F-EOF:  Error reading stats output file name'
c
2070    continue
        stop '%Avg-F-NODATA:  No data frames were read'
        end
```

Program RMS

```
      program RMS
c
c  This program takes cyclicly averaged vorticity data from the output
c    file 'vortave.out' and calculates a normalized vorticity using
c    a spatially averaged RMS value.
c
c
c  Author:
c    H. Sean Hilbert
c  History:
c     Oct 14, 1990 -- ver1.0 used for crunching two-stroke engine data
c
c
      integer N                /* N = # of polygons */
      parameter(N=31)
      real AvVort(N)           /* read in from vortave.out */
      real Norm(N)             /* normalized vorticity */
      real tot,SQAV,RMS
      character*80 junk
c
      tot = 0.0                /* set up */
c
      open(unit=1,file='vortave.out')
      open(unit=2,file='normvort.out')
c
      read(1,1000)junk         /* read header line and discard */
      write(2,1300)            /* write header of output file */
c
      do 100 i=1,N
              read(1,1100) AvVort(i)
              print*,AvVort(i)
              SQAV = AvVort(i)**2.0
              tot = tot + SQAV
100   continue
c
      print*,tot
c
      RMS = SQRT(tot/N)        /* get RMS value */
      print*,RMS
c
      do 200 i=1,N
              Norm(i) = AvVort(i)/RMS
              write(2,1200) i,Norm(i)
              print*,i,Norm(i),AvVort(i)
200   continue
c
      write(2,1400) RMS
c
1000  format(a80)
1100  format(60x,f12.5)
1200  format(' Normalized vorticity at ',i2,' is ',f12.5)
1300  format('    Average vorticity normalized with spatial RMS')
1400  format('       Spatial RMS of vorticity: ',f10.5)
      stop
      end
```

Program VECTOR

```
        program vector
c
c  Description:
c      This program creates scalable gnuplot vector command files to
c      be used with the gnuplot 'load' command. It also creates a key
c      for the input bottom of the plot indicating a relative speed.
c      The input files to this program are point files created by
c      the Megavision "sample" command.
c
c  Author:
c      H. Sean Hilbert
c
c  Version: ver(1.0) -- used for two-stroke data
c
c  Variables:
c
        real scale                      /* scaling factor */
        real x1,x2,y1,y2                /* points */
        real x1k,x2k,x3k,yk             /* label positions */
        real len                        /* length of key vector */
        real Z,Znew,a,b                 /* hypotenuse */
        real conv                       /* mm/pixel */
        real delay                      /* delay before photo */
        real key                        /* speed to make key */
        integer i,N
        character*80 frmnam,junk
c
c  set constants
c
        pi = 3.14159
        N = 44                          /* # of points/frame */
        delay = 0.00014                 /* in seconds */
        key = 2000                      /* key velocity in mm/sec */
        conv = 0.0342                   /* mm/pixel */
c
c  get scale factor
c
        print*, 'Input scale factor: '
        read(*,*) scale
        print*,scale
c
c  open files
c
         print*, 'What grid do you wish to scale? (type
     + out full path and filename): '
        read(*,'(a80)') frmnam
c
        open(1,file='/usr2/hilbert/bin/datum.pts')
        open(2,file=frmnam)
        open(3,file='gnuplot.aro')
        open(4,file='gnuplot.pts1')
c
c       read in data and do calculations
c
        read(1,1000) junk
        read(2,1000) junk
c
        do 100 i=1,N
                read(1,1100) x1, y1
                read(2,1100) x2, y2
c
c
c  turn upside down and convert to mm
c
```

Program VECTOR

```
                y1 = (1024 - y1) * conv
                y2 = (1024 - y2) * conv
                x1 = x1 * conv
                x2 = x2 * conv
                write(4,1400) x1,y1
c
c            get deltax and deltay
c
                a = x2 - x1
                b = y2 - y1
c
c            find length
c
                Z = sqrt(a**2 + b**2)
c
c            scale
c
                Znew = Z * scale
c
c            decompose
c
                a = Znew * a/Z              /* a/Z = cos(theta) */
                b = Znew * b/Z              /* b/Z = sin(theta) */
c
c            get new x2 and y2
c
                x2 = x1 + a
                y2 = y1 + b
c
c            make gnuplot file
c
                write(3,1200) x1,y1,x2,y2
c
100       continue
c
c  calculate length and placement for key
c
          len = key * delay * scale
          x1k = 15                          /* start of key arrow */
          x2k = x1k + len                   /* end of key arrow */
          x3k = x2k + 0.2                   /* for label */
          yk = 2                            /* y position of key */
c
          write(3,1200) x1k,yk,x2k,yk       /* key arrow */
          write(3,1300) x3k,yk              /* label placement */

c
1000      format(a80)
1100      format(8x,f4.0,7x,f4.0)      /* for *.pts files */
c1100      format(15x,f5.1,1x,f5.1,4x,f5.1,1x,f5.1)  /* for old *.aro */
1200      format('set arrow from ',f5.1,',',f5.1,' to ',f5.1,',',f5.1)
1300      format('set label "2 m/sec" at ',f5.1,',',f5.1)
1400      format(f7.2,f7.2)
          stop
          end
```

Program VECTOR2

```
      program vector2
c
c  Description:
c      This program creates scalable gnuplot vector command files to
c      be used with the gnuplot 'load' command. It also creates a key
c      for the input bottom of the plot indicating a relative speed.
c      The input files to this program is a gnuplot.aro file created
c      after averaging or other operations have been done to the point
c      file data. It was designed to take output files from velplot.F.
c
c  Author:
c      H. Sean Hilbert
c
c  Version: ver(1.0) -- used for two-stroke data
c
c  Variables:
c
      real scale                        /* scaling factor */
      real x1,x2,y1,y2                  /* points */
      real x1k,x2k,x3k,yk               /* label positions */
      real len                          /* length of key vector */
      real Z,Znew,a,b                   /* hypotenuse */
      real conv                         /* mm/pixel */
      real delay                        /* delay before photo */
      real key                          /* speed to make key */
      integer i,N
c
c  set constants
c
      pi = 3.14159
      N = 44                            /* # of points/frame */
      delay = 0.00014                   /* in seconds */
      key = 2000                        /* key velocity in mm/sec */
      conv = 0.0342                     /* mm/pixel */
c
c  get scale factor
c
      print*, 'Input scale factor: '
      read(*,*) scale
c
c  open files
c
      open(1,file='gnuplot.aro')
      open(2,file='gnuplot.pts1')
      open(3,file='nc_vel.aro')
c
c     read in data and do calculations
c
      do 100 i=1,N
              read(1,1100) x1, y1, x2, y2
c
c
c  turn upside down and convert to mm (some data may not need it)
c
c             y1 = (1024 - y1) * conv
c             y2 = (1024 - y2) * conv
c             x1 = x1 * conv
c             x2 = x2 * conv
              write(4,1400) x1,y1
c
c           get deltax and deltay
c
              a = x2 - x1
              b = y2 - y1
```

Program VECTOR2

```
c
c           find length
c
               Z = sqrt(a**2 + b**2)
c
c           check for no movement
c
               if(Z.eq.0.0) then
                       goto 999
               endif
c
c           scale
c
               Znew = Z * scale
c
c           decompose
c
               a = Znew * a/Z              /* a/Z = cos(theta) */
               b = Znew * b/Z              /* b/Z = sin(theta) */
c
c           get new x2 and y2
c
               x2 = x1 + a
               y2 = y1 + b
c
c           make gnuplot file
c
999            continue
c
               print*, x1,y1,x2,y2
               write(3,1200) x1,y1,x2,y2
c
100       continue
c
c  calculate length and placement for key
c
          len = key * delay * scale
          x1k = 15                         /* start of key arrow */
          x2k = x1k + len                  /* end of key arrow */
          x3k = x2k + 0.2                  /* for label */
          yk = 2                           /* y position of key */
c
          write(3,1200) x1k,yk,x2k,yk      /* key arrow */
          write(3,1300) x3k,yk             /* label placement */

c
1000      format(a80)
1100      format(15x,f5.1,1x,f5.1,4x,f5.1,1x,f5.1)
1200      format('set arrow from ',f5.1,',',f5.1,' to ',f5.1,',',f5.1)
1300      format('set label "2 m/sec" at ',f5.1,',',f5.1)
1400      format(f7.2,f7.2)
          stop
          end
```

Subroutine CONVERT

```
        subroutine convert( xpix, ypix, ddpix, uvpix, UnDim,i)
c
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: The first time through, no values are printed, and the
c       following values are returned in the appropriate argument.
c               MmPixel -> ddpix
c               dt      -> uvpix
c               Xoff    -> xpix
c               Yoff    -> ypix
c
c Author: CHuck Gendrich <cpg>
c
c History:  <cpg> August, 1987 v1.0
c  <cpg> 31 may 89 v1.1 -- return conversions the first time through
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
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
        open(unit=9,file='vort.out')
        open(unit=10,file='uv.out')
```

Subroutine CONVERT

```
        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,
     +'  ddfact: ',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    = '0.034200000 mm/pixel'
c
        MmPixel = getreal( line)
c
        write( *, 1020)
        line = '0.00014000 sec'
        dt   = getreal( line)
c
        Mmdt = MmPixel/dt
        Mmdt2 = Mmdt * Mmdt
c
        write( *, 1030)
        line = '442.0      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
```

Subroutine CONVERT

```
        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
c
c       return these values the first time through
        xpix  = Xoff
        ypix  = Yoff
        ddpix = MmPixel
        uvpix = dt

        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
                write(9,1350)i,xmm,ymm,ddmm
        else if( ddmm.eq.0.and.uvmm.ne.0) then
                write( *, 1130) xmm, ymm, uvmm
```

Subroutine CONVERT

```
                write(10,1400)i,xmm,ymm,uvmm
        else
                write( *, 1130) xmm, ymm
        endif
c
1350    format('vorticity ',i2,' at centroid (',f10.5,','
     +         ,f10.5,') is ',f12.5)
1400    format(2x,i2,' Reynolds stress at (',f10.5,',',f10.5,
     +        ') is ',f15.4)
        return
        end
```

Program Vorticity

```
        program VORTICITY
        character*41 Version
        parameter( Version='VORTICITY Version v1.6')
c
c       Description:
c               This program reads two data files produced by the
c       "sample" function of the MegaVision 1024XM.  The user must
c       input the data points in the same order from each data frame.
c       Unreadable data points must be entered as 0,0 and the program
c       will dismiss that data set automatically.
c
c               This program performs a number of tasks:
c       1) reads the data
c       2) throws out bad data points
c       3) interpolates in space and time to construct a velocity field
c          throughout the frame
c       4) uses the velocity information to determine stress and
c          strain values, vorticity, etc.
c
c       Author: Chuck Gendrich (Give me unix or ...)
c       Co-author: H. Sean Hilbert (Give me back my PC)
c
c       History:
c       August, 1987 -- 1st version -- written on freyja, an IRIS 3120
c               running 1 System V.
c       December, 87 -- 1.1 <cpg> Added graphics display of the
c               calculations (velocities, vorticity, etc.)
c               Added TURB3D-formatted output.
c       May, 1989 -- 1.2 <cpg> Modified the match subroutine so that
c               "close" is based on the shortest distance between 2
c               points in the first (reference) frame.
c        9 jul 89 -- 1.3 <cpg> Finished "triangle" polygon logic.
c               Added "Version" variable to main().
c       12 jul 89 - v1.4 <cpg> More "triangle" logic in make_poly.
c               Added "irreg()" considerations to fluids.  (oops!)
c               Took graphics stuff from "graphics.h" and "frame.f"
c               and put them here as step 2.
c       25 sep 90 - v1.5 <hsh> Removed iris graphics routines and
c               reconfigured input and output files. also removed
c               all point finding logic. user must now enter points
c               from both frames in the proper order.
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c # define DEBUG
# include "vorticity.h"
c
        real    Lft, Rt, Bottom, Top
c               Lft == Xmin
c               Rt == Xmax
c               Bottom == Ymin
c               Top == Ymax
c       This is the calling order for ortho2()...
        logical Again, UnDim
        real    MmPix, dt
        integer NumPts, NPol
        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
        character*80 choice,polymap
        character*1 answera,answerb
c               these are variables for the polygon routines
c
```

Program Vorticity

```
c
c       NOTE:  UnDim comes back TRUE when output will be non-dim'd
c
cccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c step 1 -- get the data
c
        write( *, '(a)') Version
10      continue
        call getfrm( frame1, frame2, MmPix, dt, UnDim, ierr)
        if( ierr.ne.0) then
c               error handle
                if( Again( ierr)) then
                        goto 10
                        endif
        endif
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c step 2    -- after data is read in it must be checked to make sure all
c              points are good. if a point is unreadable on the
c              Megavision screen then the user must move the cursor to
c              0,0 and record that as data. this routine will pick that
c              up and eliminate that data set from frame1 and frame2.
c
        open(unit=15,file='test.out')
        n=NumPts(frame1,1)
        print*,n
        do 20 i=1,n
              if(frame2(i,Y).ge.1019.)then
                     match(i)=0
              else
                     match(i)=i
              endif
        write(15,999)match(i),frame2(i,X)-frame1(i,X)
     +  ,frame2(i,Y)-frame1(i,Y)
999     format(i2,2x,2(f5.0))
20      continue
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c            write gnuplot 2.0 files to plot out vector fields
c
        open(unit=12,file='gnuplot.aro')
        open(unit=13,file='gnuplot.pts1')
        open(unit=14,file='gnuplot.pts2')
c
        do 100 i=1,n
              if(match(i).ne.0) then
                     write(12,1000)frame1(i,X),frame1(i,Y),
     +                 frame2(i,X),frame2(i,Y)
                     write(13,1100)frame1(i,X),frame1(i,Y)
                     write(14,1100)frame2(i,X),frame2(i,Y)
                else
                     goto 100
                endif
100     continue
c
1000    format('set arrow from ',f5.1,',',f5.1,' to ',f5.1,',',f5.1)
1100    format(2(f5.1,3x))
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c step 5 -- interpolate the velocity vectors
c
        call Velocities( frame1, frame2, match, vel)
```

Program Vorticity

```
        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)))

c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c step 6 -- define the polygons
c
30      continue
        write(*,1050)
        read(*,1060)choice
        data answera/'d'/
        data answerb/'c'/
        if((choice.eq.answera).or.(choice.eq.answerb))then
40              continue
        else
                goto 30
        endif
        if(choice.eq.answera)then
                write(*,1070)
                read(*,1060)polymap
                open(unit=35,file=polymap)
                read(35,1080)Npol
                read(35,1060)junk
c  reset array
                do 45 i=1,MaxPoly
                        poly(i,1)=0
                        poly(i,2)=0
                        poly(i,3)=0
                        poly(i,4)=0
45              continue
                do 50 i=1,Npol
                        read(35,1040)poly(i,1),poly(i,2),
     +                               poly(i,3),poly(i,4)
50              continue
        else if(choice.eq.answerb)then
                call Polygons(poly,Npol)
        endif
60      continue
        print*,Npol
        write(*,1030)
        write(*,1040)(poly(i,1),poly(i,2),poly(i,3),poly(i,4)
     +                  ,i=1,Npol)
        close(35)
c
c   format statements
c
1030    format(/' Here are the polygons which have been defined:'//
     +' ULH  URH  LRH  LLH')
        write( *, 1040) (poly(i,1),poly(i,2),poly(i,3),poly(i,4),
     +  i=1,Npol)
1040    format(4(2x,i2,1x))
1050    format(' Would you like to use a previously defined polygon'/
     + '    map, or would you like to create one?'/
     + '    Define = d      Create = c ')
1060    format(a80)
1070    format(' What is the filenme of the polygon map that you'/
     + '                    wish to load?')
1080    format(i2)
c
```

Program Vorticity

```
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c step 7 -- calculate fluid kinematic quantities
c
        call fluids( vel, poly, irreg, MmPix, dt, UnDim, ierr)
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c step 8 -- calculate reynolds stresses
c
        call spatave(vel)
c
        end
```

Program VORTAVE

```
      program vortave
C
C Description:
c
C       Read in a number of vorticity.F output files (grid*.vort)
c       and calculate average centroid locations and average
c       vorticities.
C
C
C Input file:
C       An input file is required in which operating parameters are
C       specified.  This file must be named 'vortave.dat'.  It should
C       contain the following:
C               # comment
C               # comment
c               output filename         -- filename of your choice
C               data frame 1 name       -- measurement 1
C                       ...             -- measurements 2, 3, ...
C               data frame n name       -- final measurement frame
c               end                     -- end of file marker
C Output file:
c    output file will contain averaged centroid locations and
c    averaged vorticities for those output files which contain
c    30 grid baxes. frames w/o 30 grid boxes must not be included
c    in the input file.
C
C Author:
C     H. Sean Hilbert
C
C History:
c    <hsh> ver1.0 -- used for crunching two-stroke engine data
c
C
      character*30    infil
      parameter( infil='vortave.dat') /* input data file */
c
      character*80    line    /* input line */
      character*80    datfil  /* data file containing frame information */
      character*80    outfil  /* output file containing average locations */
      character*80    junk    /* to read in junk lines in data */
      integer N,i
c        N = # of polygons
      parameter(N = 31)      /* # of polygons in data file */
      real x(N),y(N)         /* centroids read in from data files */
      real Vort(N)           /* vorticities read in */
      real Tvort(N)          /* total vorticity for each centroid */
      real xtot(N),ytot(N)   /* total centroid values */
c
C
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c
c
c Get operating parameters
      open( 1, file=infil, status='old', err=2000)
20    continue                      /* Comments are permitted only at */
      read( 1, 1000, end=2010) line /* the beginning of the file.  */
      if( line(1:1).eq.'#') goto 20 /* Skip them -- '# comment...' */
      read( line, 1000)       outfil
c
      open( 2, file=outfil, status='unknown', err=2020)
c
c set variables
      do 30 i=1,N
```

Program VORTAVE

```
                xtot(i) = 0.0
                ytot(i) = 0.0
                Tvort(i) = 0.0
30      continue
c
c  now read in data from data frames and do arithematic
c
40      continue              /* come here after each frame is done */
        read(1,1000) datfil
        if(datfil.eq.'end') then
                goto 800
        endif
        open(4,file=datfil)
c
        read(4,1000)junk
c
        do 60 i=1,N              /* read a frame */
        read(4,1300,end=60) x(i),y(i),Vort(i)
        xtot(i) = xtot(i) + x(i)
        ytot(i) = ytot(i) + y(i)
        Tvort(i) = Tvort(i) + Vort(i)
60      continue
        close(4)
        goto 40                         /* go back for next frame */
800     continue                        /* all data has been read */
c
        write(2,1600)
c   calculate averages
        do 70 i=1,N
                x(i) = xtot(i)/N
                y(i) = ytot(i)/N
                Vort(i) = Tvort(i)/N
                write(2,1700) i,x(i),y(i),Vort(i)
70      continue
c
1000    format( a80)
1300    format(26x,f10.5,1x,f10.5,4x,f12.5)
1600    format(//,' Averaged vorticity    in <1/sec>')
1700    format(' Average vorticity ',i2,' at centroid (',f10.5,',',
     +                f10.5,') is ',f12.5)
        stop '%vortave-A-OK: normal termination'
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c       ERROR HANDLING IS DONE HERE
c
2000    continue
        stop '%Avg-F-OPENFAIL:  Error opening input data file.'
c
2010    continue
        write( *, '(a80)') infil
        stop '%Avg-F-ONLY#:  Input file contains only comments'
c
2020    continue
        write( *, '(a80)') outfil1
        stop '%Avg-F-OPENERR:  Error opening MV output file'
c
2030    continue
        write( *, '(a80)') outfil2
        stop '%Avg-F-OPENERR:  Error opening stats output file'
c
2040    continue
        stop '%Avg-F-EOF:  Error reading velocity frame file'
c
```

Program VORTAVE

```
2050    continue
        write( *, '(a80)') basfil
        stop '%Avg-F-BADFRM:  Base data frame is bad (makfrm)'
c
2070    continue
        stop '%Avg-F-NODATA:  No data frames were read'
        end
```

Program STRESS

```
        program stress
C
C Description:
c
C       Read in a number of vorticity.F output files and
c       calculate u'v' and u'v'bar for each intersection
C
C
C Input file:
C       An input file is required in which operating parameters are
C       specified.  This file must be named 'stress.dat'.  It should
C       contain the following:
C               # comment
C               # comment
c               output filename        -- filename of your choice
c               avgvel.dat             -- avg vals for all frames
C               data frame 1 name      -- measurement 1
C                       ...            -- measurements 2, 3, ...
C               data frame n name      -- final measurement frame
c               end                    -- end of file marker
C Output file:
c    output file will contain u'v' for each frame as well as
c    UV which is the average Reynolds stress for each point averaged
c    over all of the frames in the data frame file.
C
C Author:
C     H. Sean Hilbert
C
C History:
c    <hsh> ver1.0 -- used for crunching two-stroke engine data
c
# include "vorticity.h"                   /* global variable defs */
C
        character*30    infil
        parameter( infil='stress.dat')  /* input data file */
c
        character*80    line    /* input line */
        character*80    basfil  /* data file containing frame information */
        character*80    datfil  /* data file containing frame information */
        character*80    outfil  /* output file containing average locations */
        integer N,mtch,Frm,i
c         N = # of points, Frm = frame counter
        parameter(N = 44)      /* # of points in Vbar file */
        integer save(N)        /* divisor for calculating u'v'bar */
        real vxr,vyr           /* velocities read in from data files */
        real Ubar(N),Vbar(N)   /* average velocities read in */
        real Uprm,Vprm         /* instantaneous fluctuations */
        real uvi               /* instantaneous reynolds stress */
        real UV(N)             /* u'v'bar */
        real conv              /* conversion to mm**2/sec**2 */
c
c Notice!!! this is a pain, but you must change all of the (44)
c   array statements if you have a grid with more than 44 points
C
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
        conv = (0.0342 * 0.0342)/(0.00014 * 0.00014)
c
c
c Get operating parameters
        open( 1, file=infil, status='old', err=2000)
20      continue                        /* Comments are permitted only at */
        read( 1, 1000, end=2010) line   /* the beginning of the file.  */
        if( line(1:1).eq.'#') goto 20   /* Skip them -- '# comment...' */
```

Program STRESS

```
        read( line, 1000)        outfil
c
        open( 2, file=outfil, status='unknown', err=2020)
c
                read( 1, 1000, end=800)basfil
c
        open( 3, file=basfil)
100     continue             /* keep coming back until file is mt */
        read( 3, 1100, err=2040, end=500) i,Ubar(i)
        read( 3, 1100, err=2040, end=500) i,Vbar(i)
        goto 100
500     continue             /* end of file -- OK! */
C
c  now read in data from other frames and do arithematic
C
        Frm = 0              /* set frame counter */
40      continue             /* come here after each frame is done */
        Frm = Frm + 1
        write(2,1200) Frm
        read(1,1000) datfil
        if(datfil.eq.'end') then
                goto 800
        endif
        open(4,file=datfil)
50      continue             /* come here after each point is done */
        read(4,1300,end=60) mtch,vxr,vyr
        if(mtch.ne.0)then
                Uprm = vxr - Ubar(mtch)
                Vprm = vyr - Vbar(mtch)
                uvi = (Uprm * Vprm)*conv       /* instantaneous u'v' */
                UV(mtch) = UV(mtch) + uvi    /* create an ensemble */
                save(mtch) = save(mtch) + 1  /* index UV divisor */
                write(2,1400) mtch,uvi
        else
                write(2,1500)        /* no match */
        endif
        goto 50                      /* go back for next point */
60      continue
        close(4)
        goto 40                      /* go back for next frame */
800     continue                     /* all data has been read */
c
        write(2,1600)
c   calculate average reynolds stresses
        do 70 i=1,N
                UV(i) = UV(i)/save(i)
                write(2,1700) i,UV(i)
70      continue
C
1000    format( a80)
1100    format(9x,i2,10x,f10.5)  /* format of 'avgvel.dat' */
1200    format(' Instantaneous Reynolds stress for frame ',i2)
1300    format(i2,2x,2(f5.0))
1400    format(' u''v'' at ',i2,' is ',e12.4)
1500    format(' **** no match for this point ****')
1600    format(//,' Averaged Reynolds stresses')
1700    format(' u''v''bar at ',i2,' is ',e12.4)
        stop '%stress-A-OK: normal termination'
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c       ERROR HANDLING IS DONE HERE
c
2000    continue
```

Program STRESS

```
        stop '%Avg-F-OPENFAIL:  Error opening input data file.'
c
2010    continue
        write( *, '(a80)') infil
        stop '%Avg-F-ONLY#:  Input file contains only comments'
c
2020    continue
        write( *, '(a80)') outfil1
        stop '%Avg-F-OPENERR:  Error opening MV output file'
c
2030    continue
        write( *, '(a80)') outfil2
        stop '%Avg-F-OPENERR:  Error opening stats output file'
c
2040    continue
        stop '%Avg-F-EOF:  Error reading velocity frame file'
c
2050    continue
        write( *, '(a80)') basfil
        stop '%Avg-F-BADFRM:  Base data frame is bad (makfrm)'
c
2070    continue
        stop '%Avg-F-NODATA:  No data frames were read'
        end
```

Program UVPRIME

```
        program uvprime
C
C Description:
c
C       Read in a number of vorticity.F output files and
c       calculate u',v' and q = sqrt(u'**2 + v'**2) for each intersection
C
C
C Input file:
C       An input file is required in which operating parameters are
C       specified.  This file must be named 'uvprime.dat'.  It should
C       contain the following:
C               # comment
C               # comment
c               output filename          -- filename of your choice
c               avgvel.dat               -- avg vals for all frames
C               data frame 1 name        -- measurement 1
C                   ...                  -- measurements 2, 3, ...
C               data frame n name        -- final measurement frame
c               end                      -- end of file marker
C Output file:
c    output file will contain u',v' and q for each point averaged
c    over all of the frames in the data frame file.
C
C Author:
C     H. Sean Hilbert
C
C History:
c    <hsh> ver1.0 -- used for crunching two-stroke engine data
c
# include "vorticity.h"                  /* global variable defs */
C
        character*30    infil
        parameter( infil='uvprime.dat') /* input data file */
c
        character*80    line    /* input line */
        character*80    basfil  /* data file containing frame information */
        character*80    datfil  /* data file containing frame information */
        character*80    outfil  /* output file containing average locations */
        integer N,mtch,Frm,i
c         N = # of points, Frm = frame counter
        parameter(N = 44)      /* # of points in Vbar file */
        integer save(N)        /* divisor for calculating averages */
        real vxr,vyr           /* velocities read in from data files */
        real Ubar(N),Vbar(N)   /* average velocities read in */
        real Uprm,Vprm         /* instantaneous fluctuations */
        real q(N)              /* turbulence energy term per frame */
        real Upp(N),Vp(N)      /* RMS addatives */
        real conv              /* conversion to mm/sec */
c
C
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
        conv = (0.0342)/(0.00014)
c
c
c Get operating parameters
        open( 1, file=infil, status='old', err=2000)
20      continue                        /* Comments are permitted only at */
        read( 1, 1000, end=2010) line   /* the beginning of the file.  */
        if( line(1:1).eq.'#') goto 20   /* Skip them -- '# comment...' */
        read( line, 1000)        outfil
c
        open( 2, file=outfil, status='unknown', err=2020)
```

Program UVPRIME

```
c
                read( 1, 1000, end=800)basfil
c
        open( 3, file=basfil)
100     continue                /* keep coming back until file is mt */
        read( 3, 1100, err=2040, end=500) i,Ubar(i)
        read( 3, 1100, err=2040, end=500) i,Vbar(i)
        goto 100
500     continue                /* end of file -- OK! */
c
c  now read in data from other frames and do arithematic
c
        Frm = 0                 /* set frame counter */
40      continue                /* come here after each frame is done */
        Frm = Frm + 1
        read(1,1000) datfil
        if(datfil.eq.'end') then
                goto 800
        endif
        open(4,file=datfil)
50      continue                /* come here after each point is done */
        read(4,1300,end=60) mtch,vxr,vyr
        if(mtch.ne.0)then
                Uprm = vxr - Ubar(mtch)
                Vprm = vyr - Vbar(mtch)
                q(mtch) = q(mtch) + Uprm**2 + Vprm**2  /* create ensemble */
                Upp(mtch) = Upp(mtch) + Uprm**2          /* total u'**2 */
                Vp(mtch) = Vp(mtch) + Vprm**2          /* total v'**2 */
                save(mtch) = save(mtch) + 1            /* index divisor */
        endif
        goto 50                         /* go back for next point */
60      continue
        close(4)
        goto 40                         /* go back for next frame */
800     continue                        /* all data has been read */
c
        write(2,1600)
c   calculate averages
        do 70 i=1,N
                q(i) = sqrt(q(i)/save(i))*conv
                Upp(i) = sqrt(Upp(i)/save(i))*conv
                Vp(i) = sqrt(Vp(i)/save(i))*conv
                write(2,1700) i,Upp(i),Vp(i),q(i)
70      continue
c
1000    format( a80)
1100    format(9x,i2,10x,f10.5)  /* format of 'avgvel.dat' */
1300    format(i2,2x,2(f5.0))
1400    format(' u''v'' at ',i2,' is ',e12.4)
1600    format(//,' u'',v'' and turbulent energy')
1700    format('at ',i2,'   u''= ',e10.4,'   v''= ',e10.4,
     +'  and q = ',e10.4)
        stop '%uvprime-A-OK: normal termination'
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
c       ERROR HANDLING IS DONE HERE
c
2000    continue
        stop '%Avg-F-OPENFAIL:  Error opening input data file.'
c
2010    continue
        write( *, '(a80)') infil
        stop '%Avg-F-ONLY#:  Input file contains only comments'
```

Program UVPRIME

```
c
2020    continue
        write( *, '(a80)') outfil1
        stop '%Avg-F-OPENERR:  Error opening MV output file'
c
2030    continue
        write( *, '(a80)') outfil2
        stop '%Avg-F-OPENERR:  Error opening stats output file'
c
2040    continue
        stop '%Avg-F-EOF:  Error reading velocity frame file'
c
2050    continue
        write( *, '(a80)') basfil
        stop '%Avg-F-BADFRM:  Base data frame is bad (makfrm)'
c
2070    continue
        stop '%Avg-F-NODATA:  No data frames were read'
        end
```

Program VELPLOT

```
        program velplot
c
c this program makes gnuplot files which will plot
c   velocity vector fields. It uses averaged locations for plotting
c   ensembled velocity fields.
c
c Author:
c    H. Sean Hilbert
c
        real Dx,Dy              /* datum x and y locations */
        real Vx,Vy              /* averaged velocity locations */
        real conv               /* to convert pixels to mm */
        character*80 junk
c
        conv = 0.0342           /* mm/pixel */
c
c read in the points and make the output file
c
        open(1, file='datum.pts')     /* base point data */
        open(2, file='stats_nc.junk')     /* stats output from Average.F */
        open(3, file='gnuplot.aro')   /* output file */
c
c read in junk lines before data
c
        read(2,1200) junk
        read(2,1200) junk
        read(2,1200) junk
        read(1,1200) junk
c start reading loop
        do 100 i=1,44                          /* 44 is # of points */
                read(1,1000) Dx,Dy
                read(2,1200) junk
                read(2,1100) Vx
                read(2,1100) Vy
                Dy = (-Dy + 1024) * conv       /* upside down & convert */
                Dx = Dx * conv                 /* convert only */
                Vx = Vx * conv
                Vy = Vy * conv
                write(3,1300) Dx,Dy,Vx,Vy
100     continue
c
1000    format(8x,f4.0,7x,f4.0)
1100    format(6x,f5.2)
1200    format(a80)
1300    format('set arrow from ',f5.1,',',f5.1,' to ',f5.1,',',f5.1)
        stop
        end
```

Program PLOTIT

```
      program plotit
c
c
c  Description: This program will read in location data, convert it
c   to the proper units, and combine it with any other data in an
c   output file.  This output file can be used as a data file for
c   pv-wave for example.
c
c  Author:
c      H. Sean Hilbert
c
c  History:
c      <hsh>  ver1.0  --  Used for crunching two-stroke data
c
c
c  Variables:
      real conv
      real data,Dx,Dy
      character*80 junk
c
      conv = 0.0342        /* mm/pixel */
c
c  Get data from files
c
      open(1,file='datum.pts')
      open(2,file='prime.out')
      open(3,file='tenergy.out')
c
      read(1,1000) junk        /* read in junk lines */
      print*, junk
      read(2,1000) junk
c
c  Start reading loop
      do 100 i=1,43            /* 43 is # of points */
             read(1,1100) Dx,Dy
             read(2,1200) data
             Dy = (-Dy + 1024) * conv    /* upside down and convert */
             Dx = Dx * conv
             print*, Dx,Dy,data,i
             write(3,1300) Dx,Dy,data
100   continue
c
1000  format(a80)
1100  format(8x,f4.0,7x,f4.0)
1200  format(49x,e10.4)
1300  format(f10.5,f10.5,f10.2)
      stop
      end
```

Subroutine SPATAVE

```
        subroutine spatave(vel)
c
c
c  This subroutine averages the x and y velocities over any given
c   frame and prints out the statistics for them.
c
c
c Author:
c    H. Sean Hilbert
c
c History:
c    <hsh> 9 oct 1990 -- v1.0 only u' and v' calculated
c
# include "vorticity.h"
c
        real totVx,totVy,Vxav,Vyav,conv  /* totals, averages, etc */
        real Vprm(MaxPts,2)              /* array for u',v' */
        integer NumPts
        external NumPts
        n=NumPts(vel,4)
        print*,n
        totVx = 0.0
        totVy = 0.0
c
c  open output file
c
        open(unit=11,file='uvave.out')
c
c  set conversion factor from pix/frame to mm/sec
c
        conv = 0.0342/0.00014
c
c  add 'em up
c
        do 100 i=1,n
                totVx = totVx + vel(i,Vx)
                totVy = totVy + vel(i,Vy)
100     continue
c
c  calculate averages
c
        Vxav = totVx/n
        Vyav = totVy/n
c
        print*,Vxav,Vyav
c  calculate u' and v'
c
        write(11,1000)
        do 150 i=1,n
                Vprm(i,X) = (vel(i,Vx) - Vxav) * conv
                Vprm(i,Y) = (vel(i,Vy) - Vyav) * conv
                write(11,1100)i,Vprm(i,X)*Vprm(i,Y)
                print*,i,Vprm(i,X)*Vprm(i,Y)
150     continue
c
1000    format(' Reynolds stress using u'' and v'' from
     +spatially averaged frames (in mm**2/sec**2)')
1100    format('  Reynolds stress at ',i2,' is ',e14.4)
c
        return
        end
```

Subroutine GETFRM

```
        subroutine getfrm( frame1, frame2, MmPix, dt, UnDim, ierr)
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.
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 <cpg> 31 may 89  v1.1 -- added MmPix and dt to the parameter list
c <hsh> 25 sep 90  -- changed file handling routines
c
cccc-cccc1cccc-cccc2cccc-cccc3cccc-cccc4cccc-cccc5cccc-cccc6cccc-cccc7cc
c
# include "vorticity.h"
c
        character*80 filnam1, filnam2
        real Xoffset, Yoffset, MmPix, dt
        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 ',a80/' Please try again.'/)
10      continue
c               return here on error #1
                filnam1 = '/usr2/hilbert/bin/datum.pts'
                write( *, 1000)
                call getline( filnam1)
c
                call makefrm( frame1, filnam1, ierr)
                n = NumPts( frame1, 2)
                if( ierr.eq.1) then
c                       try again if error opening the frame data file
                        write( *, 1010) filnam1
                        goto 10
```

Subroutine GETFRM

```
                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 = '/usr2/hilbert/bin/grid1.pts'
                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
c
c                       d's here are dummy variables, not used...
                        call convert( d1, d2, MmPix, dt ,UnDim,0)
                        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

```
        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
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
# include "vorticity.h"
c
        integer set, start
        character*80 filnam
        character*80 line1, line2, line3, junk
        integer HowMany
        external HowMany
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
        open( 3, err = 2000, file=filnam, status = 'old')
        print*, ' Reading data from ',filnam
c
1000    format( a80)
1010    format(8x,f4.0,7x,f4.0)

1020    format(' Zero objects found in coordinate set ',i2,'.')
        read( 3, 1000, end=2010) junk
c
        start=1
c               start stores the current number of points that have
c               been read in.
        set = 0
100     continue
c               this command reads in data that is produced by
c               the SAMPLE command in XM. it does not read in data
c               from command files written for Megavision.
        read(3,1010,end=2020,err=2050)frame(start,X),frame(start,Y)
        print*,start,frame(start,X),frame(start,Y),X,Y
        start=start+1
        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
2015    continue
c       Too many points
        print*,'Can''t store all the points from this frame.'
2020    continue
c       end of file --- OK
```

Subroutine MAKEFRM

```
        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
```

Subroutine GETLINE

```
        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, '(f10.8)') value
c
        getreal = value
        return
        end
```

Subroutine VELOCITIES

```
        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       May, 1989 v1.1 -- permit a base velocity in the Y-direction, too.
c       Sept, 1990 v1.2 -- <hsh> remove turb3d stuff and redefine
c                          match(i)
c
cccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
# include "vorticity.h"
c
        character*80 line
        real baseVx, baseVy

        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 far a point moving at the freestream'/
     +' velocity will move between frames.'/
     +' (Vx) = (deltaX) + baseVx')
        line  = '   0.0000      pixels/frame'
        baseVx = getreal( line)
        write( *, 1005)
1005    format(' Please enter the Y base velocity such that:'/
     +' (Vy) = (deltaY) + baseVy')
        line = '    0.0000      pixels/frame'
        baseVy = getreal( line)
        scale = 4.0 * exp( -abs(baseVx) / 28.0) /* crazy, isn't it?  :-) */
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
```

Subroutine VELOCITIES

```
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) + baseVx
                vel(i,Vy) = frame2(match(j),Y) - frame1(j,Y) + baseVy
c
            endif
100     continue
c
c       Now construct the output file.
c
        open( unit=4, file='vel.q')
c               unit4 -- flow field information
c
c       write out the X and Y locations, and the velocities
        write(4,1030)
        write( 4, 1020) (vel( j,X),vel(j,Y),vel(j,Vx)
     +     ,vel(j,Vy),j,j=1,i)
c
1020    format( 4(f10.5,3x),i4)
1030    format(4x,'X',13x,'Y',12x,'Vx',11x,'Vy',9x,'I')
c
        return
        end
```

Subroutine FLUIDS

```
        subroutine fluids( vel, poly, irreg, MmPix, dt, UnDim, ierr)
c
c       Description:  calculates the fluid mechanical properties of the
c       flow.  The object is to produce calculations of
c           (1)  uv
c           (2)  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       19 sep 90 <hsh> -- removed iris commands,removed spatial
c                          derrivative stuff
c       18 aug 89 <cpg> -- resurrected v1.4 with the args for v1.6
c        7 apr 88 <cpg> v1.4 -- defines a colormap based on Wz[min-max]
c       22 jan 88 <cpg> v1.3 -- graphics to draw the polygons
c       16 aug 87 <cpg> v1.2 -- added the non-dim title stuff
c       15 aug 87 <cpg> v1.1 -- removed triangle logic.  see older
c                               versions to recover it.
c       August, 1987 v1.0 -- lots of linear interpolation.  ugh.
c
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c# define DEBUG
# include "vorticity.h"
c
c
        real uv
        integer NumPts, NumPoly
        external NumPts, NumPoly
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
c
        real Wzs( MaxPoly), WZmax, WZmin
c               Stores the values of Wz, their max, and min.
c
c
c
        p = NumPoly( poly)
```

Subroutine FLUIDS

```
        n = NumPts( vel, 4)
        if( UnDim) then
                XYunits = '(noD)'
        else
                XYunits = '(mm)'
        endif
c
c   === vorticity
c
c
        if( GoOn()) then
                title  = 'vorticity'
        endif
        result = '  Wz '
        write(9,1300)title
        call CutHere( title)
        if( UnDim) then
                Runits = '(noD)'
        else
                Runits = '(1/sec)'
        endif
        write( *, 1000) XYunits, XYunits, result, Runits
c
        GAMMA = 0.0
        WZmin=+999999.
        WZmax=-999999.
        do 100 i=1, p
           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
                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)
c
           area   = area + suba
           intgrl = intgrl + subi
           Wz = -intgrl / area
c     minus because V*dS is taken in the wrong direction
           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 * 0.0342 *0.0342 / 0.00014
c     converts intgrl to mm*mm/sec
c               circ = (pix**2 / frame) * (frame/sec) * (mm/pix)**2
           GAMMA  = GAMMA - intgrl
c     minus because V*dS is taken in the wrong direction
```

Subroutine FLUIDS

```
            call convert( x0, y0, Wz, 0.0, UnDim,i)
c
100      continue
         write(9,1200)GAMMA
         close(9)
         if( GoOn()) then
                 continue
         endif
c
c   === Reynold's stress
c
         title = 'instantaneous Reynolds stress'
         write(10,1300)title
         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
            uv = vel(i,Vx)*vel(i,Vy)
            call convert( vel(i,X), vel(i,Y), 0, uv,UnDim,i)
600      continue
c
1000     format('       X',a5,'       Y',a5,3x,a5,a7)
1200     format('Total circulation is ',f12.5,' (mm^2/sec)')
1300     format(a80)
         return
         end
```

Subroutine INTEGRT

```
        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       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
ccccccccc1ccccccccc2ccccccccc3ccccccccc4ccccccccc5ccccccccc6ccccccccc7cc
c
c# define DEBUG
# include "vorticity.h"
c
        integer p1, p2
        real area, intgrl
c
        real avg, dist, tan1
        external avg, dist, tan1
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)
```

```
Subroutine INTEGRT

c
        thetaV = tanl( Vbar, Ubar)
        thetaS = tanl( 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
```

Integer function NumPts

```
        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 CENTROID

```
        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 Gendrich
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,
```

Subroutine CENTROID

```
     +' 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
#                       ifdef DEBUG
                        write( *, 1000) x0, y0
#                       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
```

Subroutine CENTROID

```
                                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
#               ifdef DEBUG
                write( *, 1010) m1, m2, b1, b2, x0, y0
#               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
#               ifdef DEBUG
                write( *, 1020) 'line 1', m2, b2, x0, y0
#               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 GETREAL

```
        real function getreal( prompt)
c
        character*80 prompt
        real value
c
        call getline( prompt)
        read( prompt, '(f10.8)') value
c
        getreal = value
        return
        end
```

example polygon descriptor file

31

ULH	URH	LLH	LRH
1	2	10	9
2	3	11	10
3	4	12	11
4	5	13	12
5	6	14	13
6	7	15	14
7	8	16	15
9	10	18	17
10	11	19	18
11	12	20	19
12	13	21	20
13	14	22	21
14	15	23	22
15	16	24	23
17	19	26	25
19	20	27	26
20	21	28	27
21	22	29	28
22	23	30	29
23	24	31	30
25	26	33	32
26	27	34	33
27	28	35	34
28	29	36	35
29	30	37	36
30	31	38	37
32	34	40	39
34	35	41	40
35	36	42	41
36	37	43	42
37	38	44	43

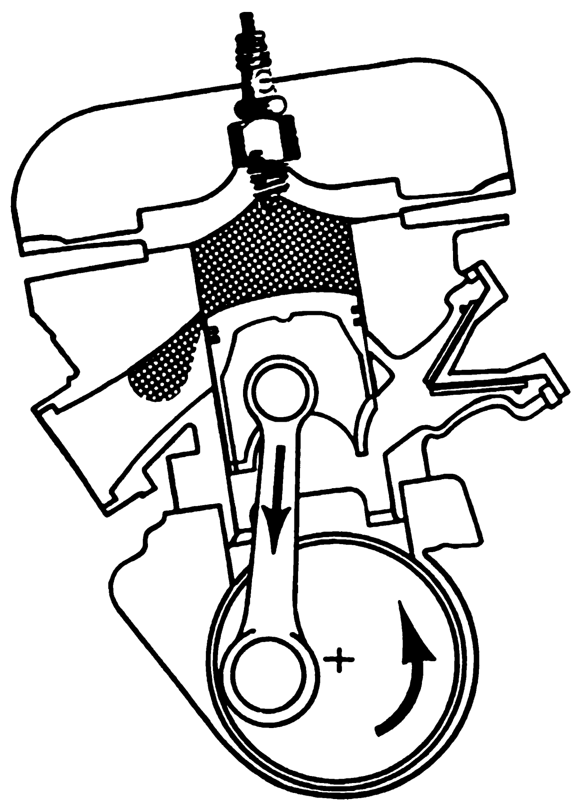


Figure 1.1a. Ignition has just occurred, and the combustion chamber is full of expanding exhaust gasses which are pushing the piston down. As the piston moves downward it begins to pressurize the lower crankcase, pushing fresh charge up through the transfer ports.

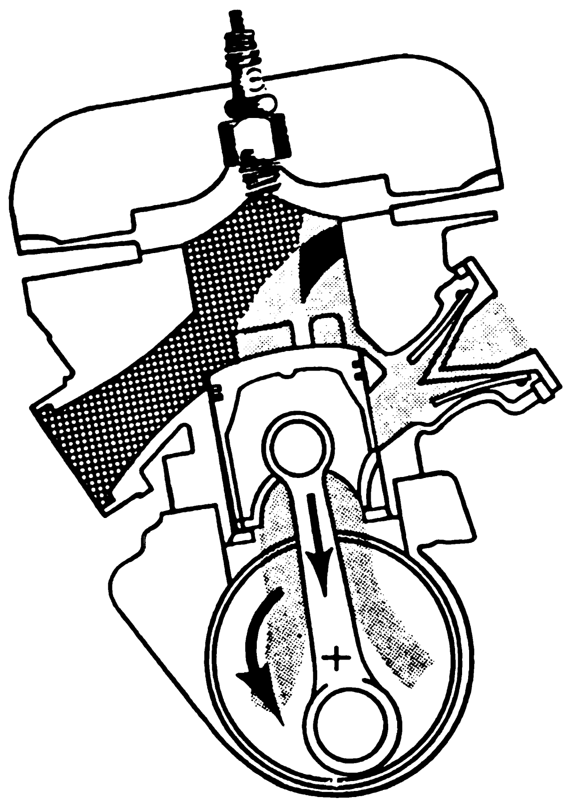


Figure 1.1b. As the piston reaches bottom dead center pressure in the lower crankcase has pushed fresh charge up through the transfer ports into the cylinder. There the loop scavenging process takes place and forces the burnt exhaust gasses out of the open port.

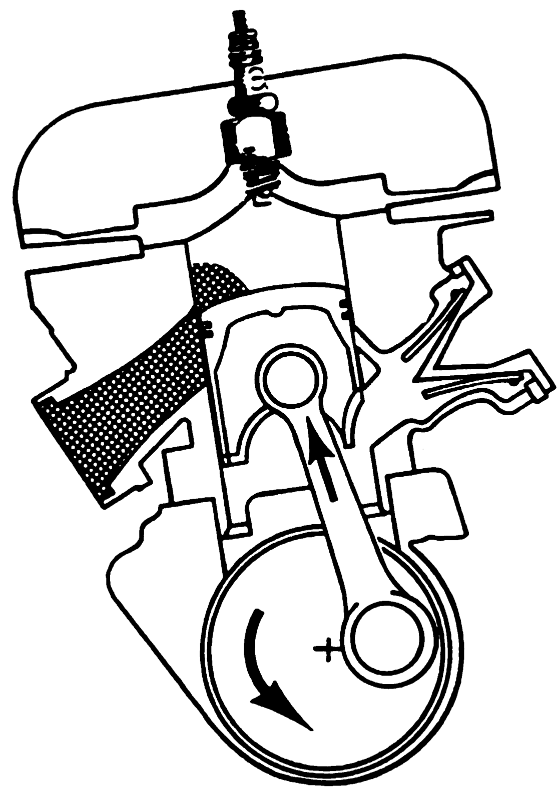


Figure 1.1c. The piston moves back up the cylinder closing off the exhaust ports as the last remaining exhaust gasses are pushed out. The fresh charge in the combustion chamber begins to undergo compression. The piston's upward travel in the cylinder creates a vacuum in the lower crankcase that pulls fresh charge from the intake track down into the crankcase.

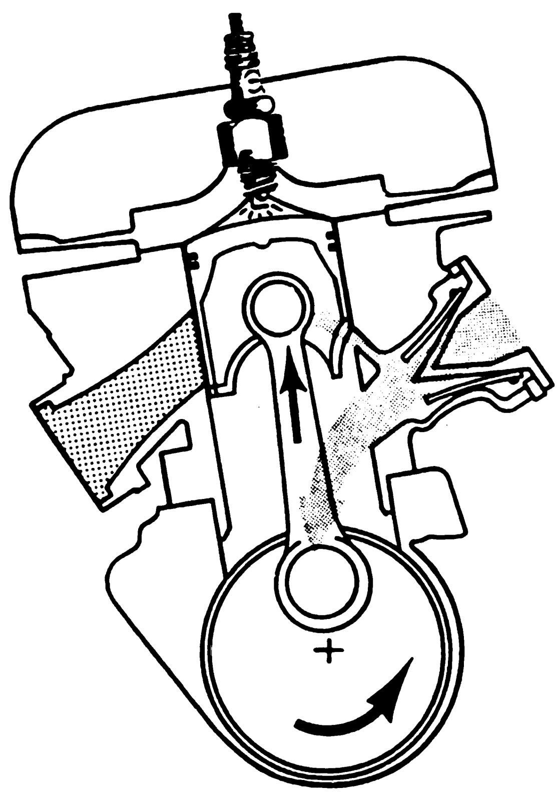


Figure 1.1d. As the piston reaches top dead center the ignition system discharges a spark, and combustion will once again take place. Meanwhile fresh charge continues to be drawn into the crankcase. The V-shape reed valve will not allow the charge to flow back out into the induction track once the crankcase has become pressurized again.

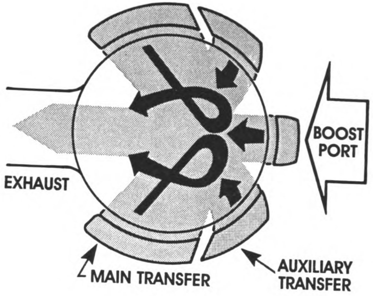


Figure 1.2 Schematic of scavenging flows at BDC in a five transfer port engine arrangement.

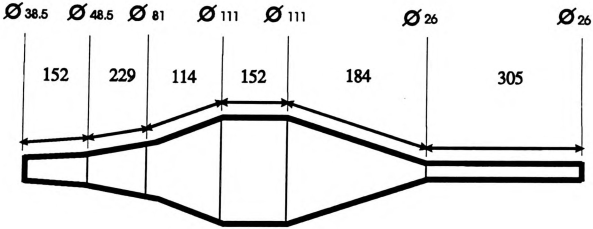


Figure 2.1. Tuned exhaust pipe dimensions showing diameters and lengths in millimeters.

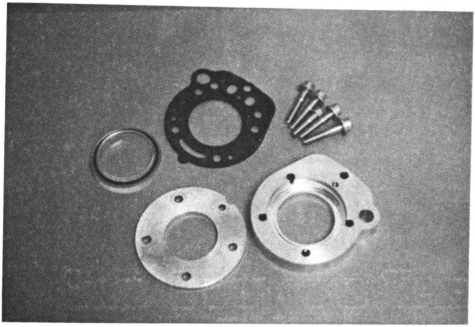


Figure 2.2. Photograph of the optical head assembly.

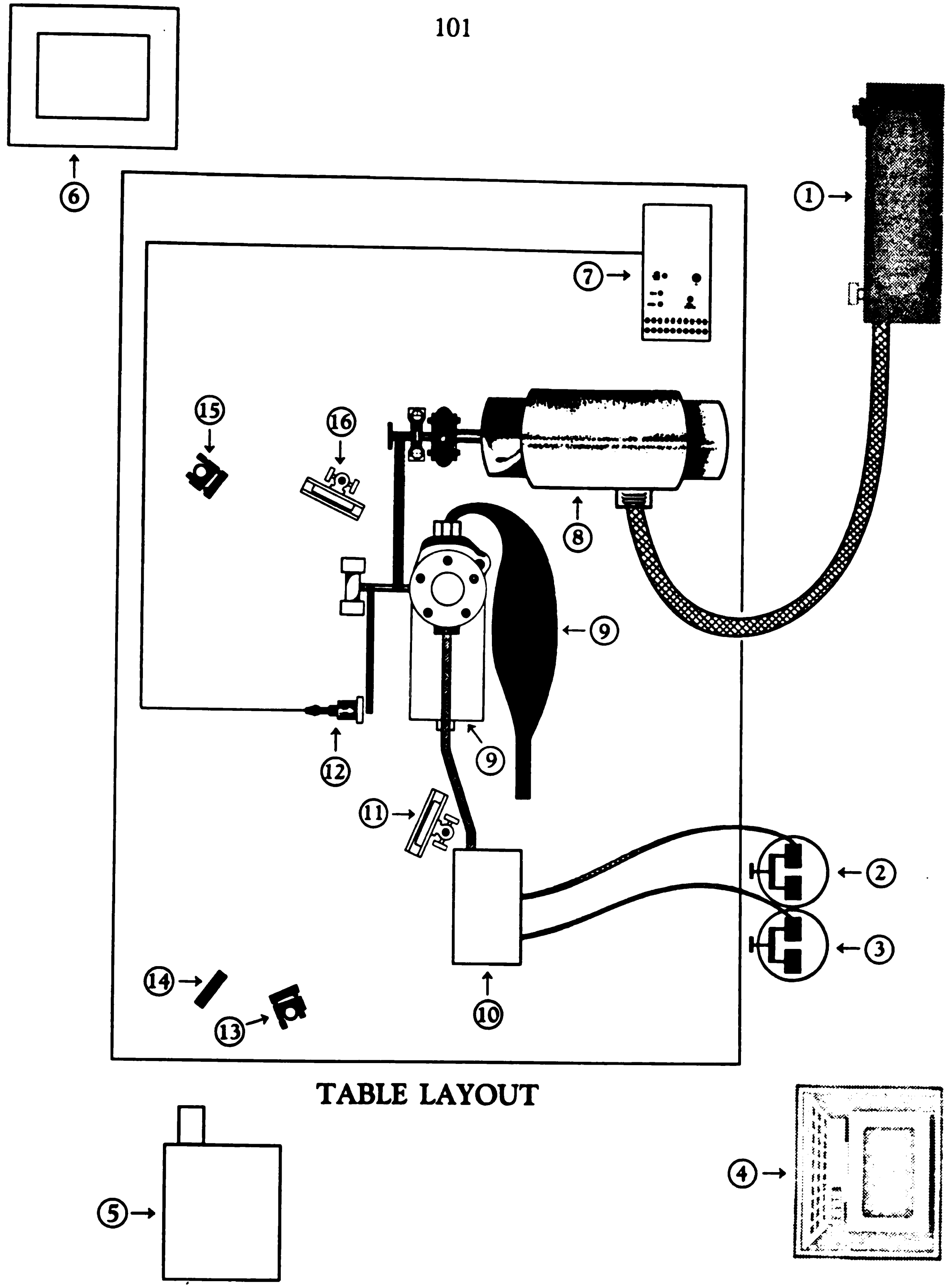


Figure 2.3 Table layout. (1) Engine controller, (2) and (3) Nitrogen tanks, (4) Laser controller, (5) Excimer laser, (6) Monitor/VCR, (7) Engine/laser/camera electronic timing box, (8) 10HP eddy current motor, (9) Engine shown with exhaust pipe, (10) Biacetyl evaporation chamber, (11) and (16) beam dividers, (12) crank angle encoder, (13) and (15) 308nm dialectrically coated mirrors, (14) 308nm 50:50 beam splitter.

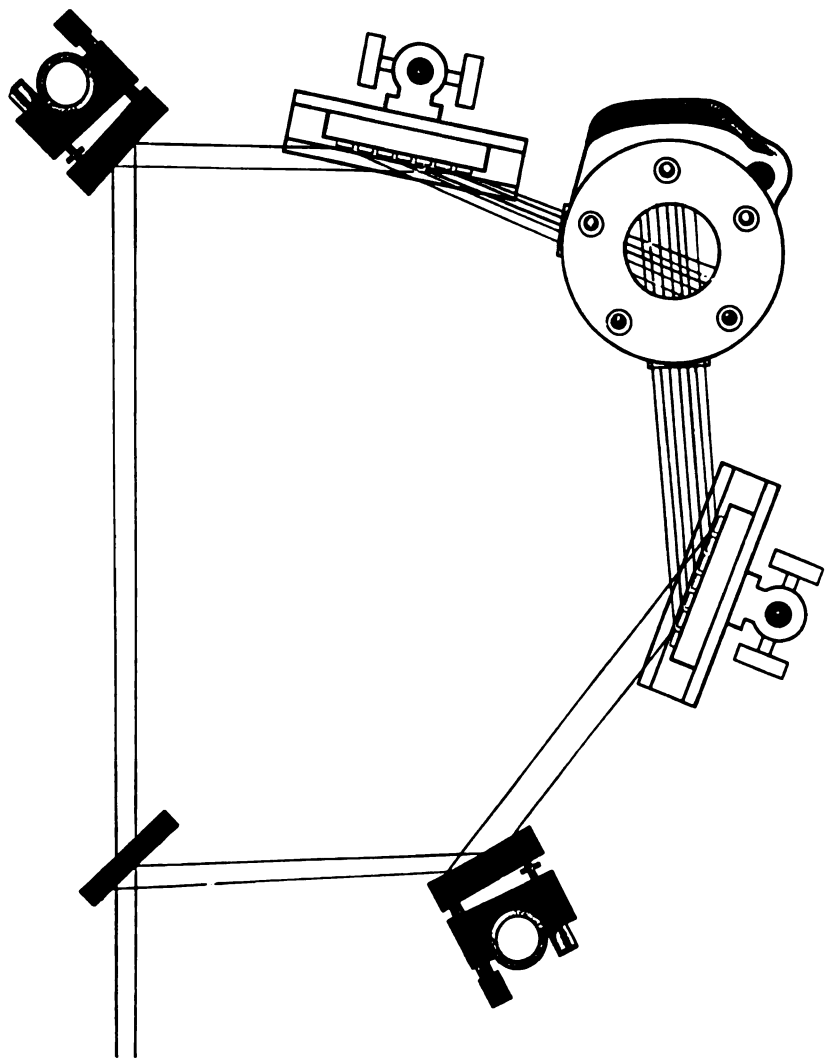


Figure 2.4. Schematic of the optical setup used to produce the grid of laser lines in the engine.

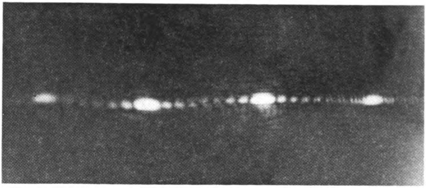


Figure 2.5. Beam divider diffraction patterns with a Helium-Neon laser.

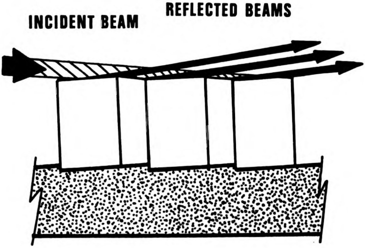


Figure 2.6. Side view of the beam divider showing the steel base, mirrors, and reflection.

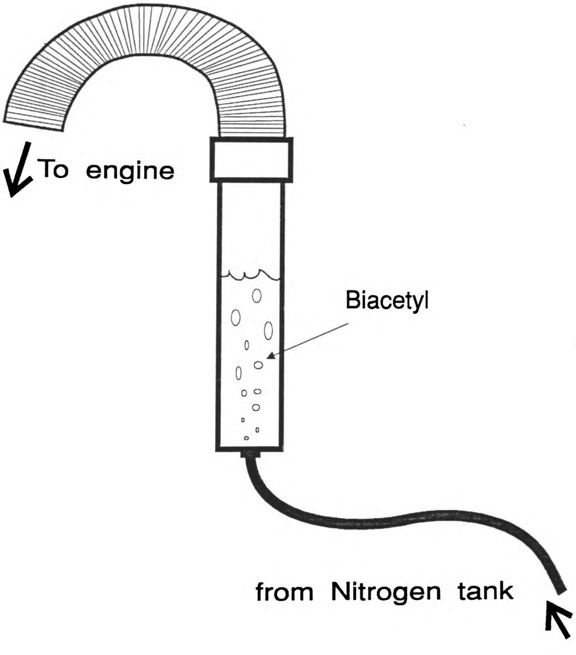


Figure 3.1. The biacetyl delivery system.

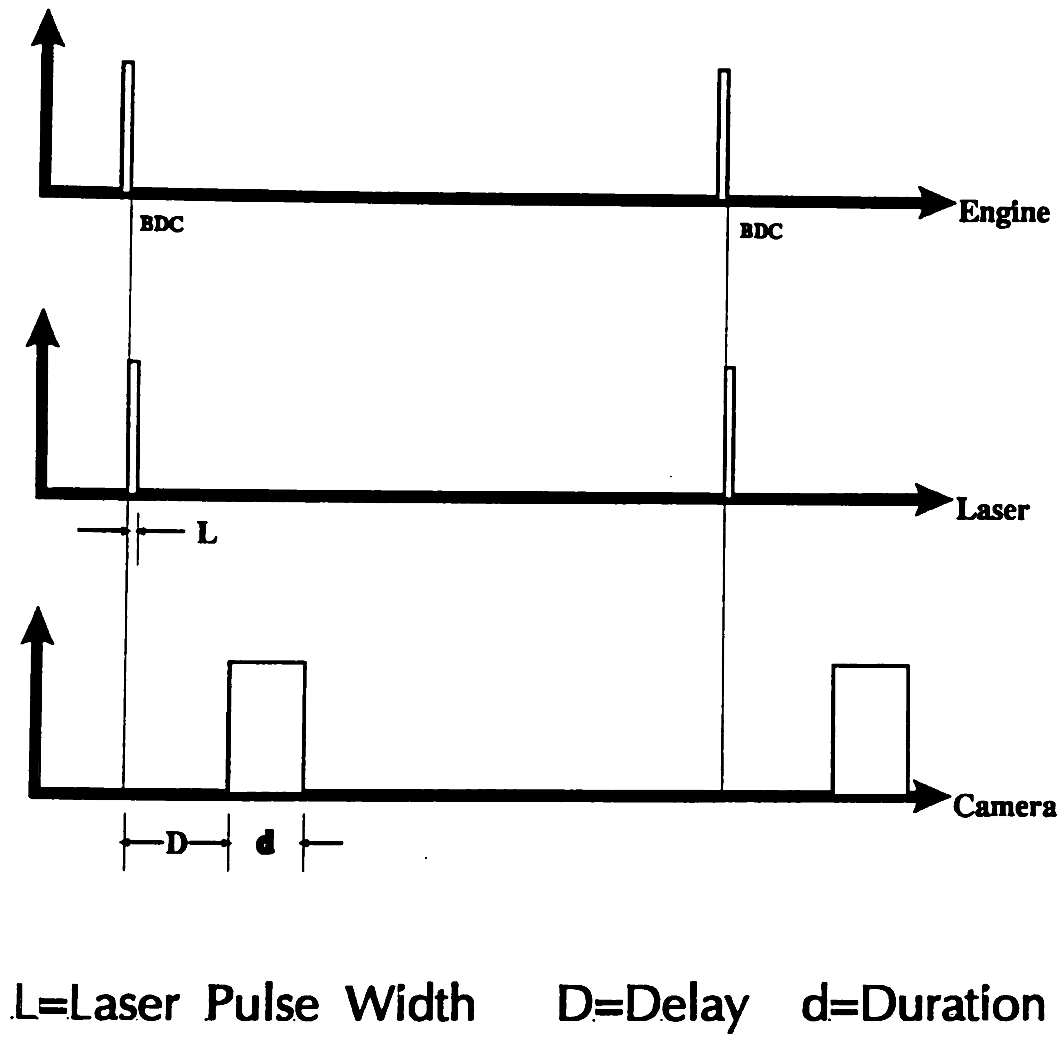


Figure 3.2. A time line illustrating the timing of the engine, the laser, and the camera during data acquisition.

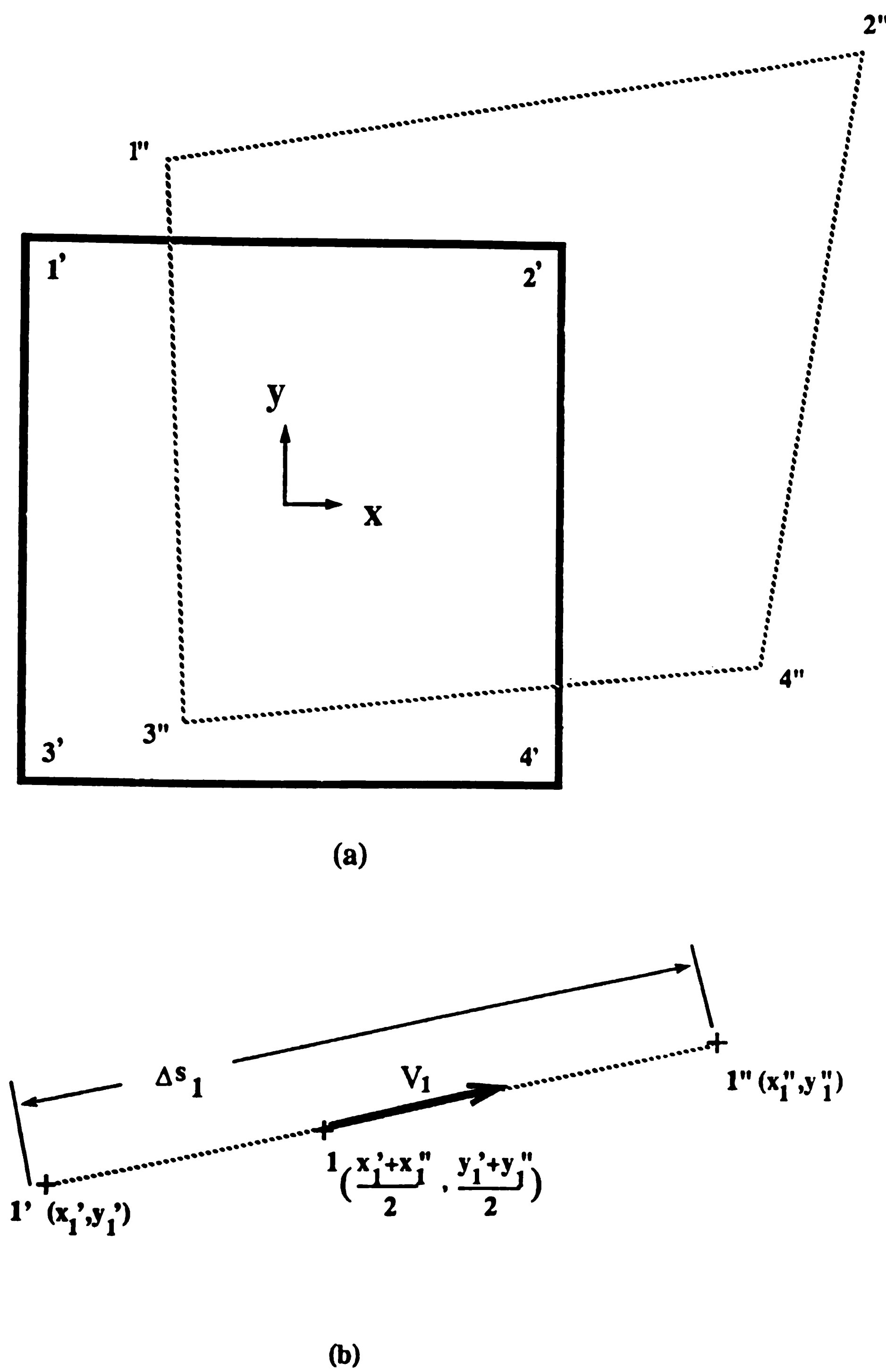


Figure 3.3. (a) A theoretical grid box, and its distortion shown a time delay (Δt) later. (b) A line projected between like corners of the undistorted and distorted grid boxes. This illustrates how velocities are calculated.

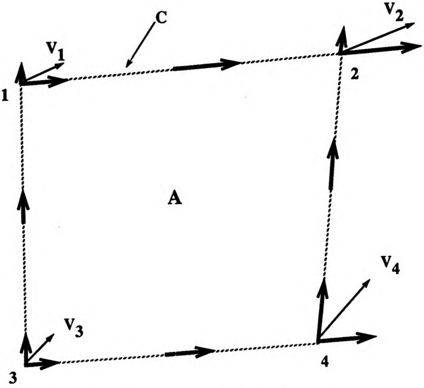


Figure 3.4. Decomposition of the velocities used to calculate the circulation around a grid box.

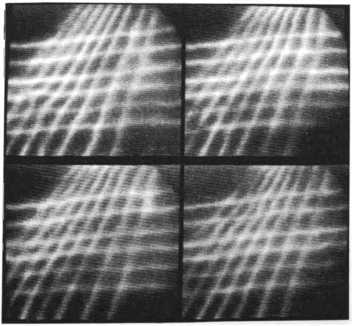


Figure 3.5. Four different raw data grids as photographed in the engine.

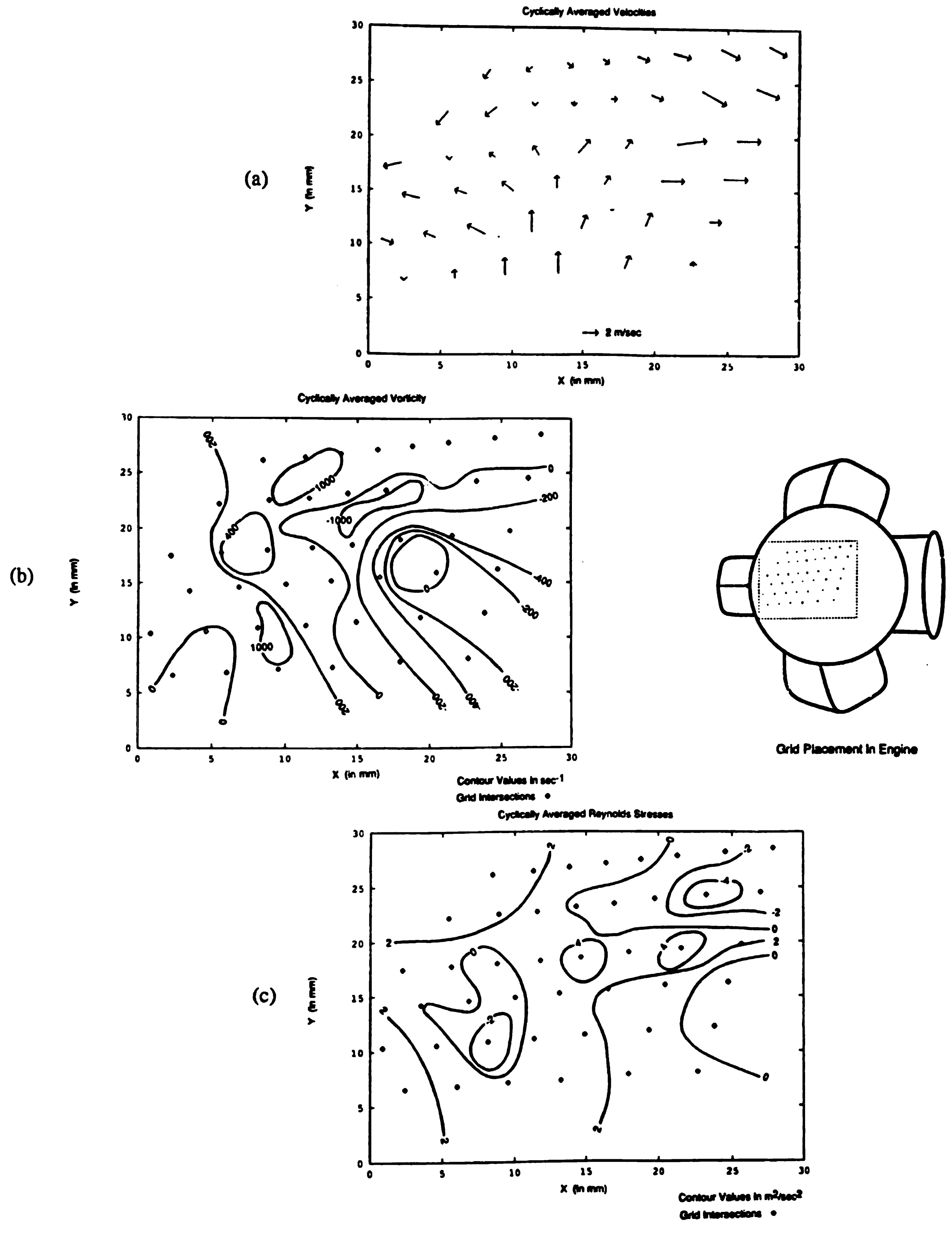
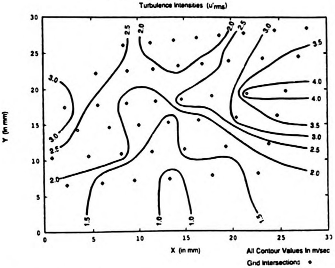
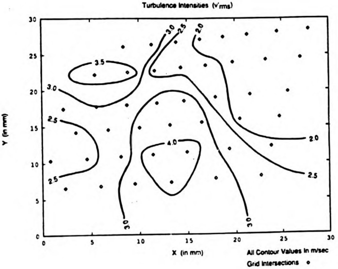


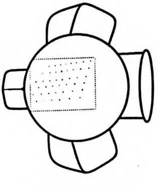
Figure 4.1. Cyclically averaged plots of velocity vectors, vorticity, and Reynolds stress.

(a)



(b)





Grid Placement In Engine

(c)

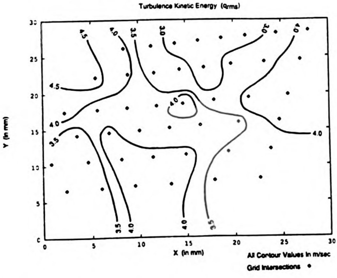
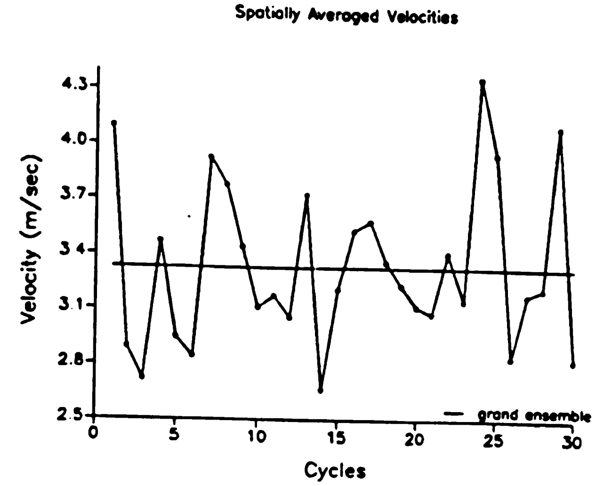
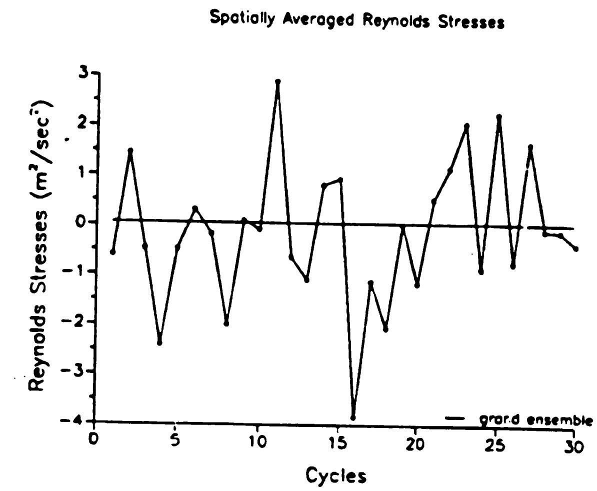


Figure 4.2. Contour plots of turbulence intensities in each coordinate direction and of turbulence kinetic energy.

(a)



(b)



(c)

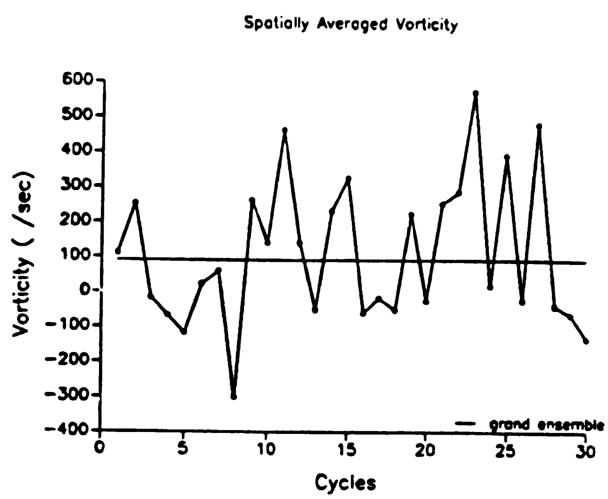


Figure 4.3. Plots of spatially averaged velocity, vorticity, and Reynolds stress for the thirty consecutive cycles. The horizontal line in each graph represents the grand ensemble average for these quantities.

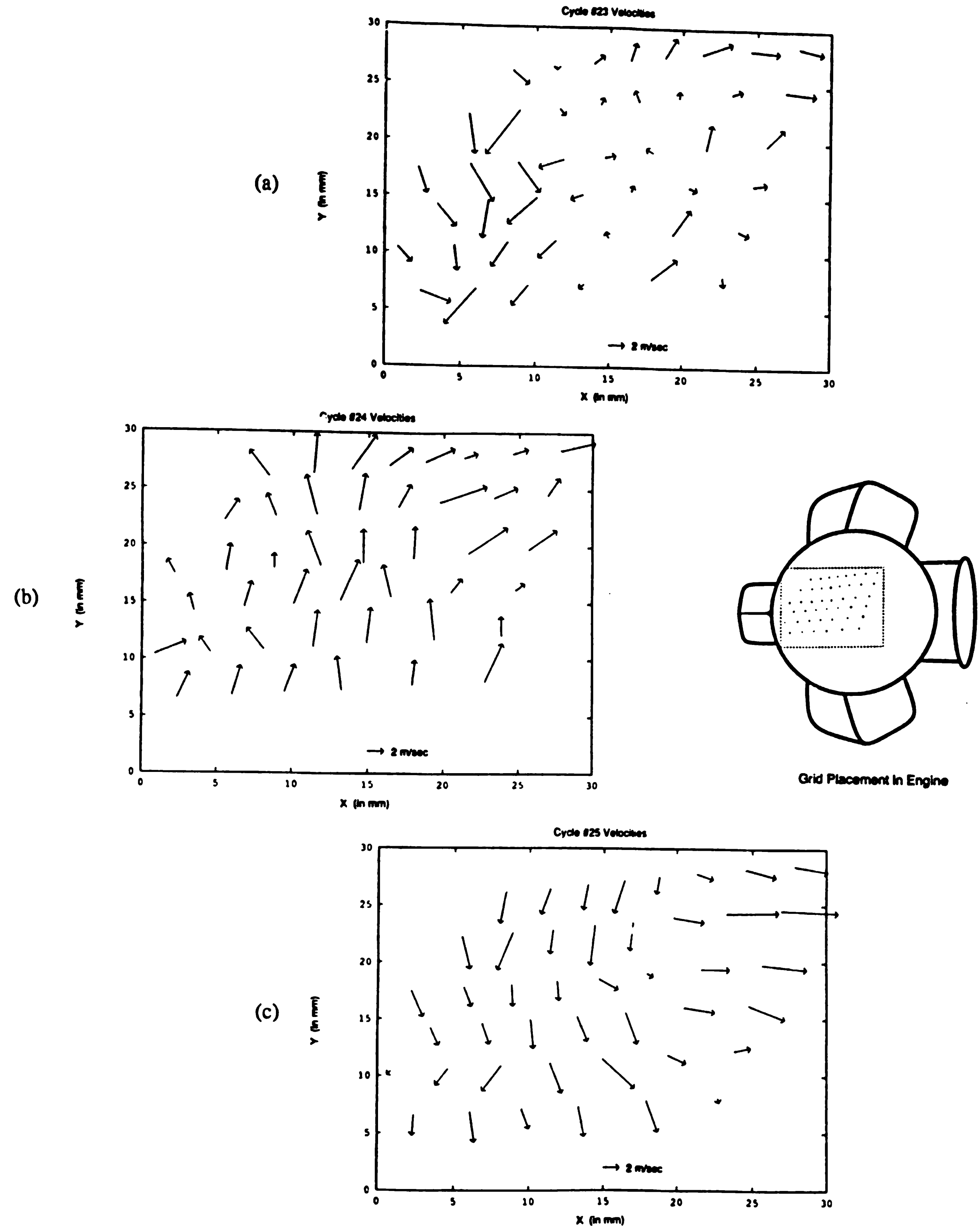


Figure 4.4. Velocity vector fields for three consecutive cycles showing very large cyclic variability.

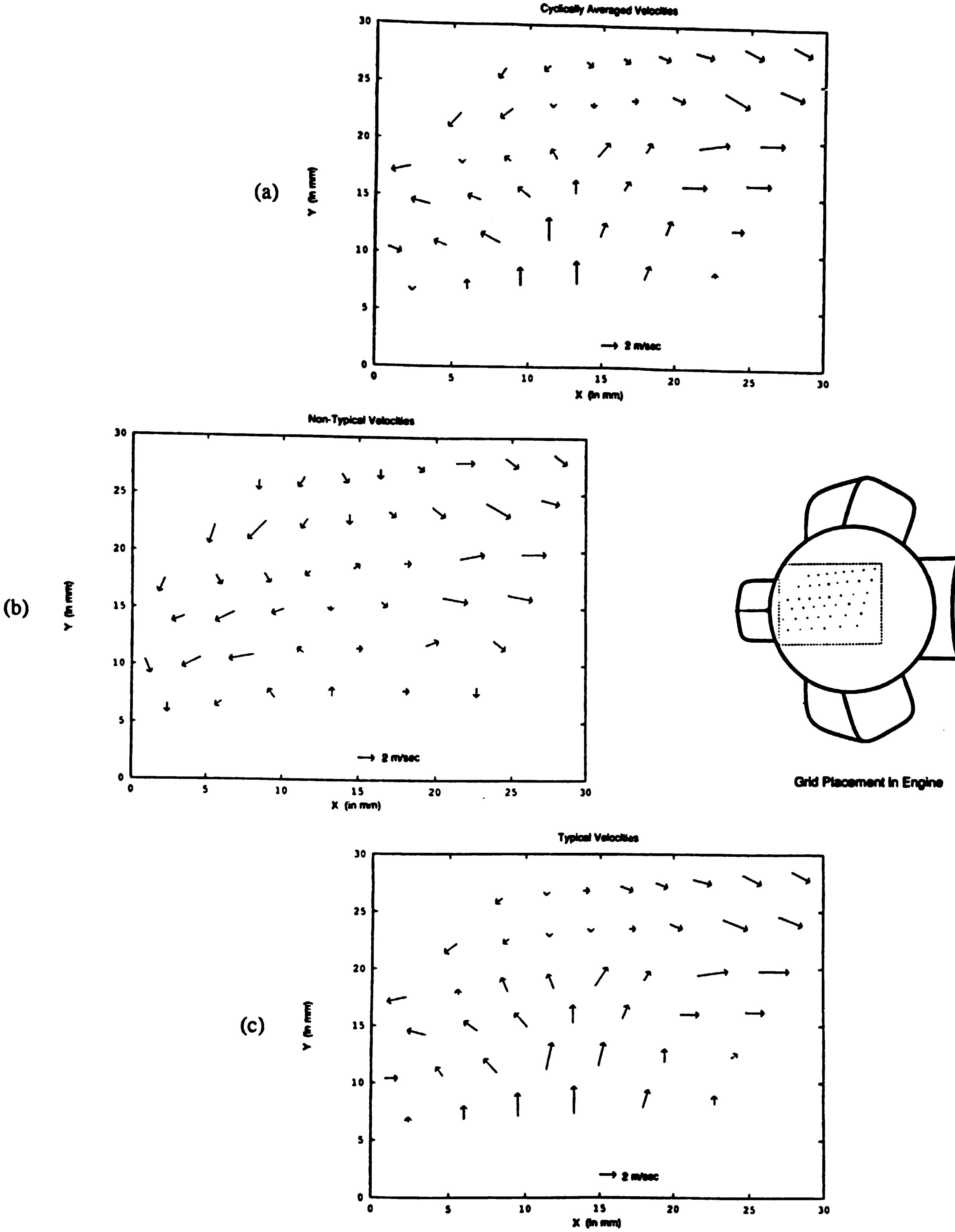


Figure 4.5. a) Velocity vector field for all cycles averaged. b) Velocity vector field for a subset which differed from the overall average. c) Velocity vector field consistent with the overall average.

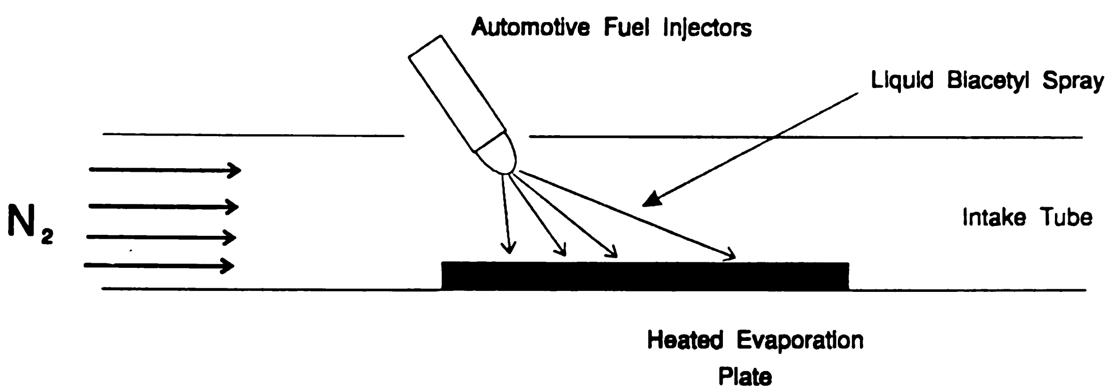


Figure 5.1. Proposed biacetyl delivery system.

Figure B1. Delivery rate vs. RPM for a 124cc engine with SR=0.8.

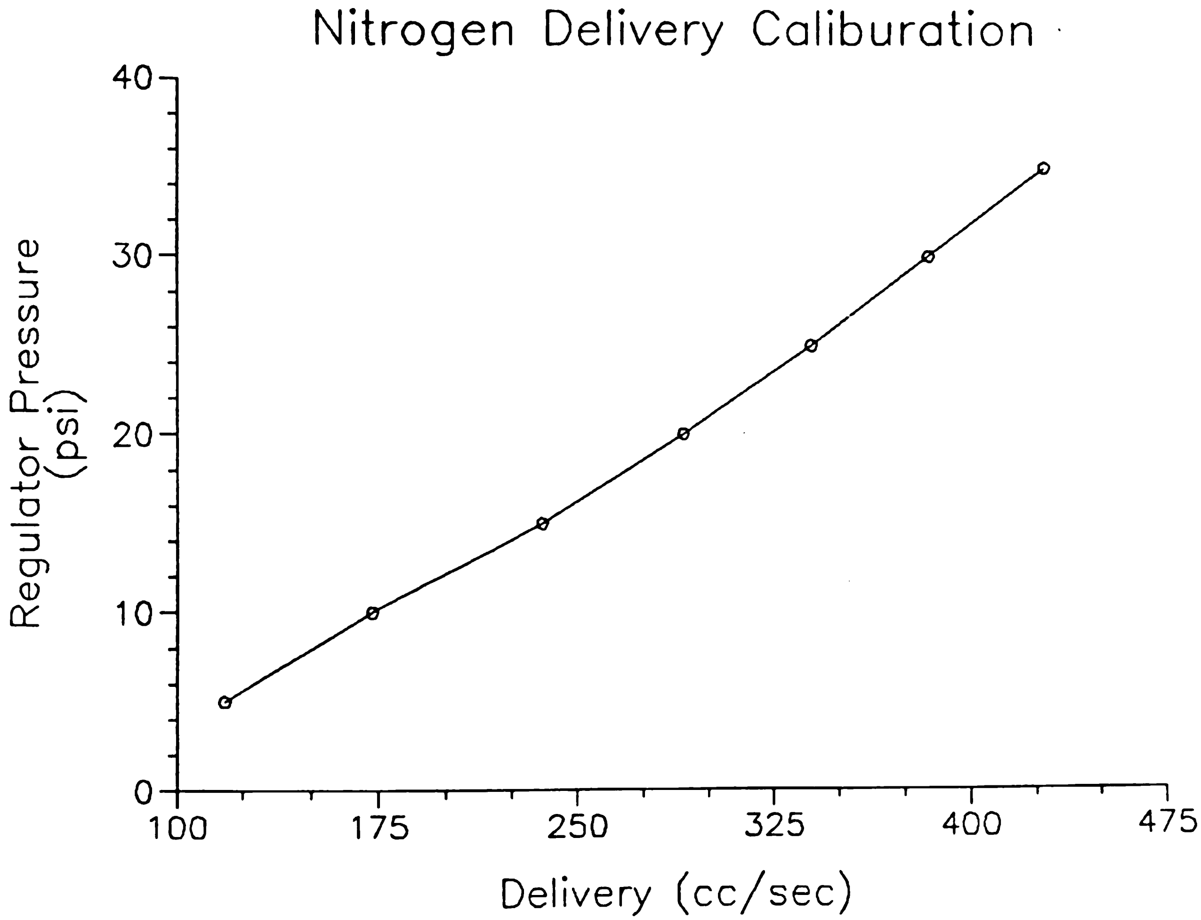


Figure B2. Calibration for nitrogen delivery.

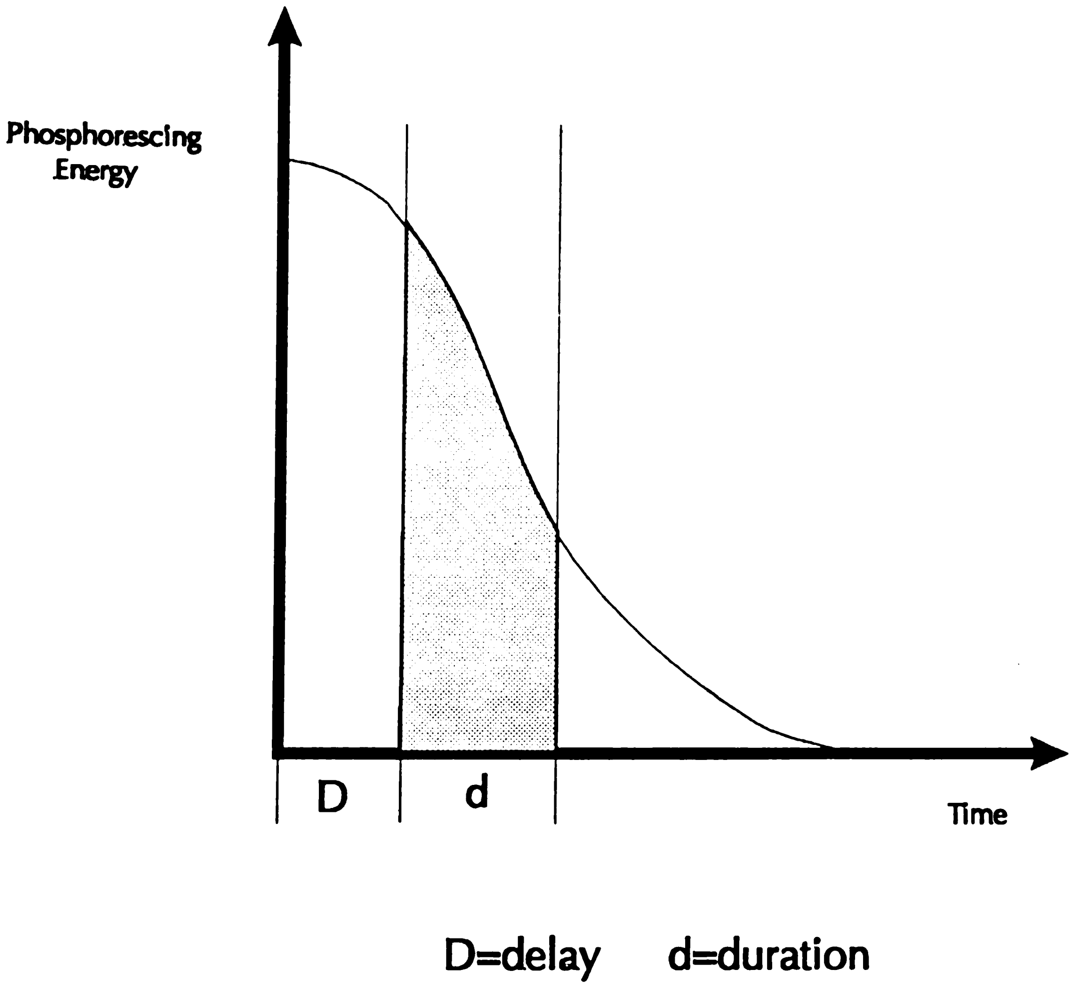


Figure C1. Example phosphorescence decay rate curve.

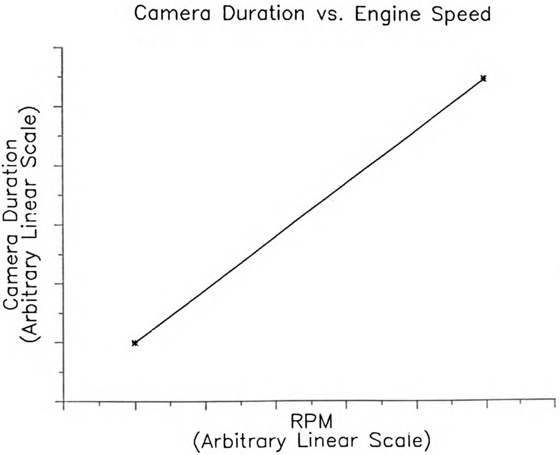


Figure C2. Camera duration vs. engine speed.

APPENDIX E

FIGURES

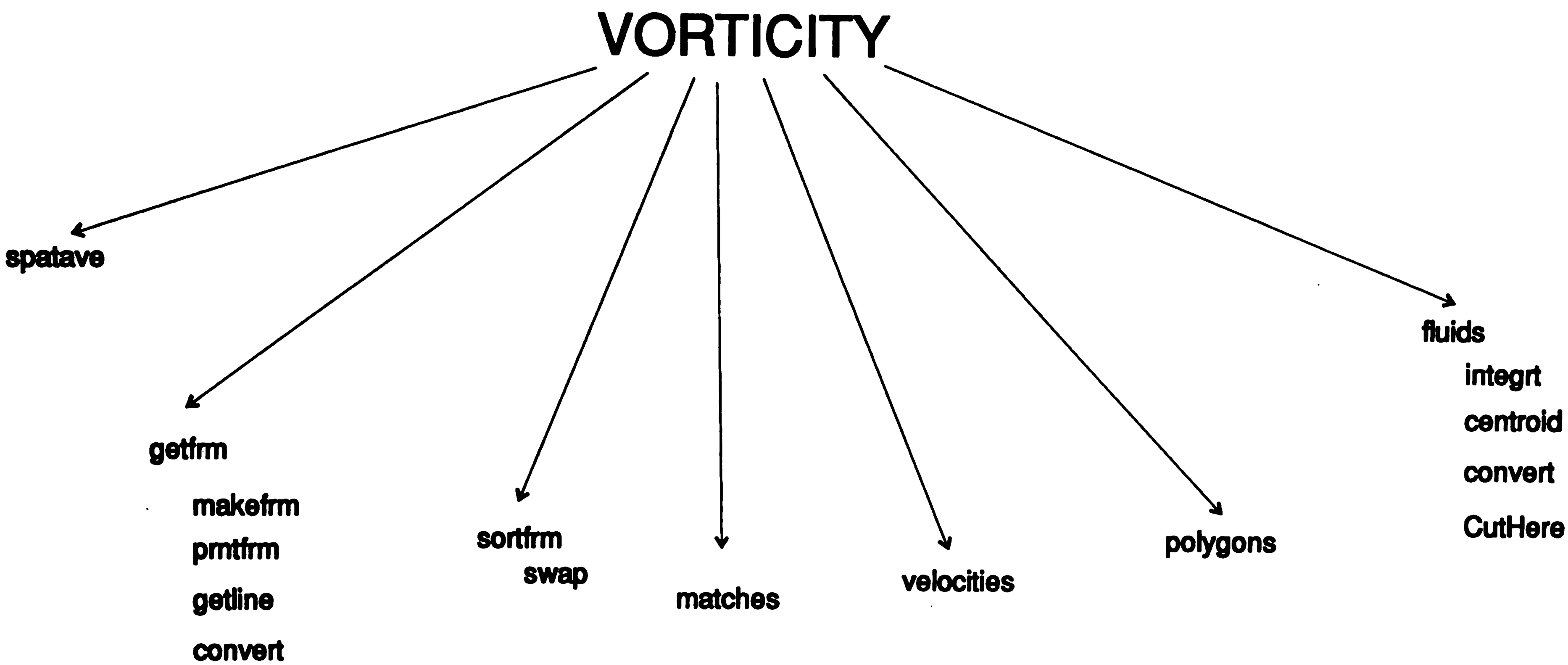
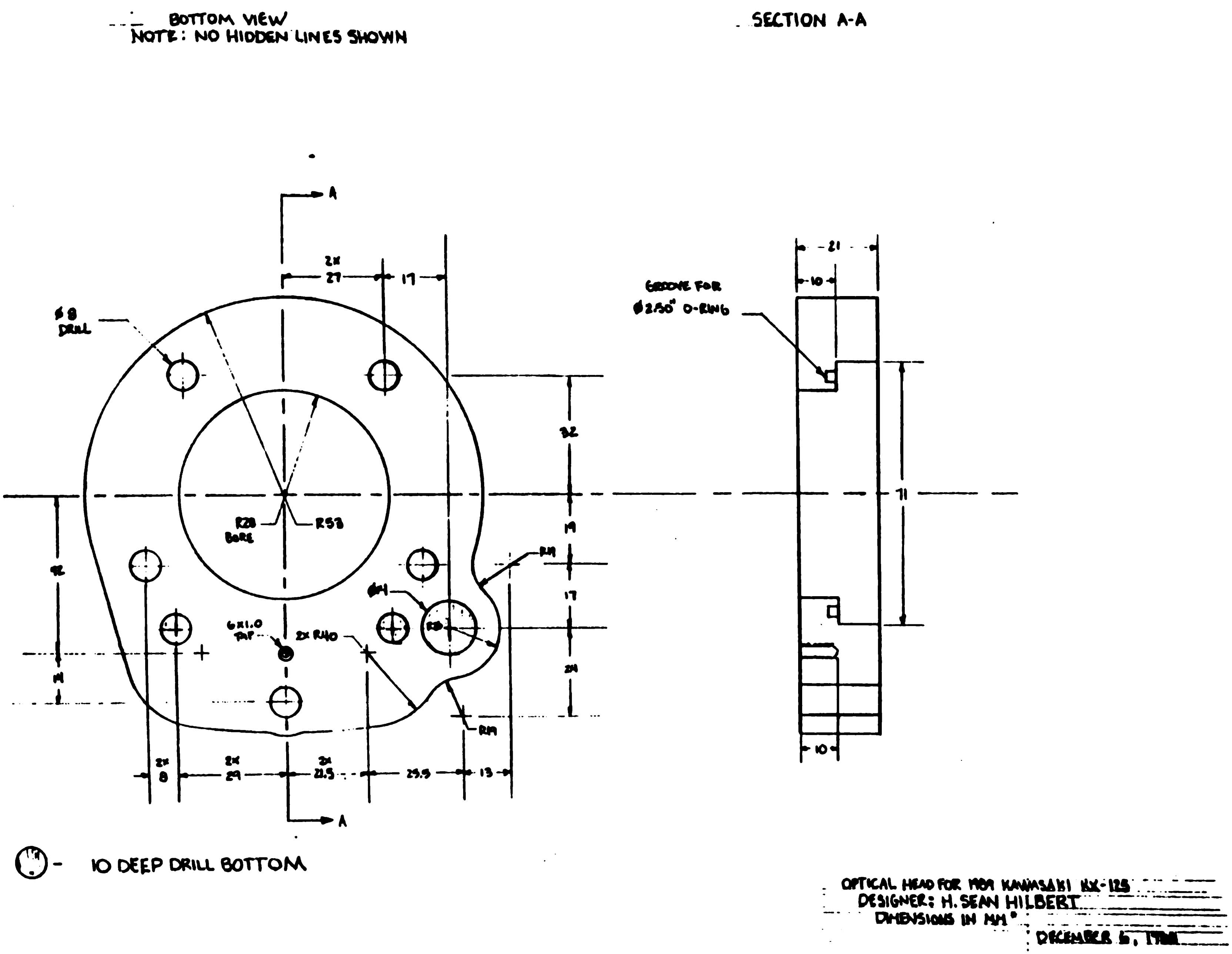
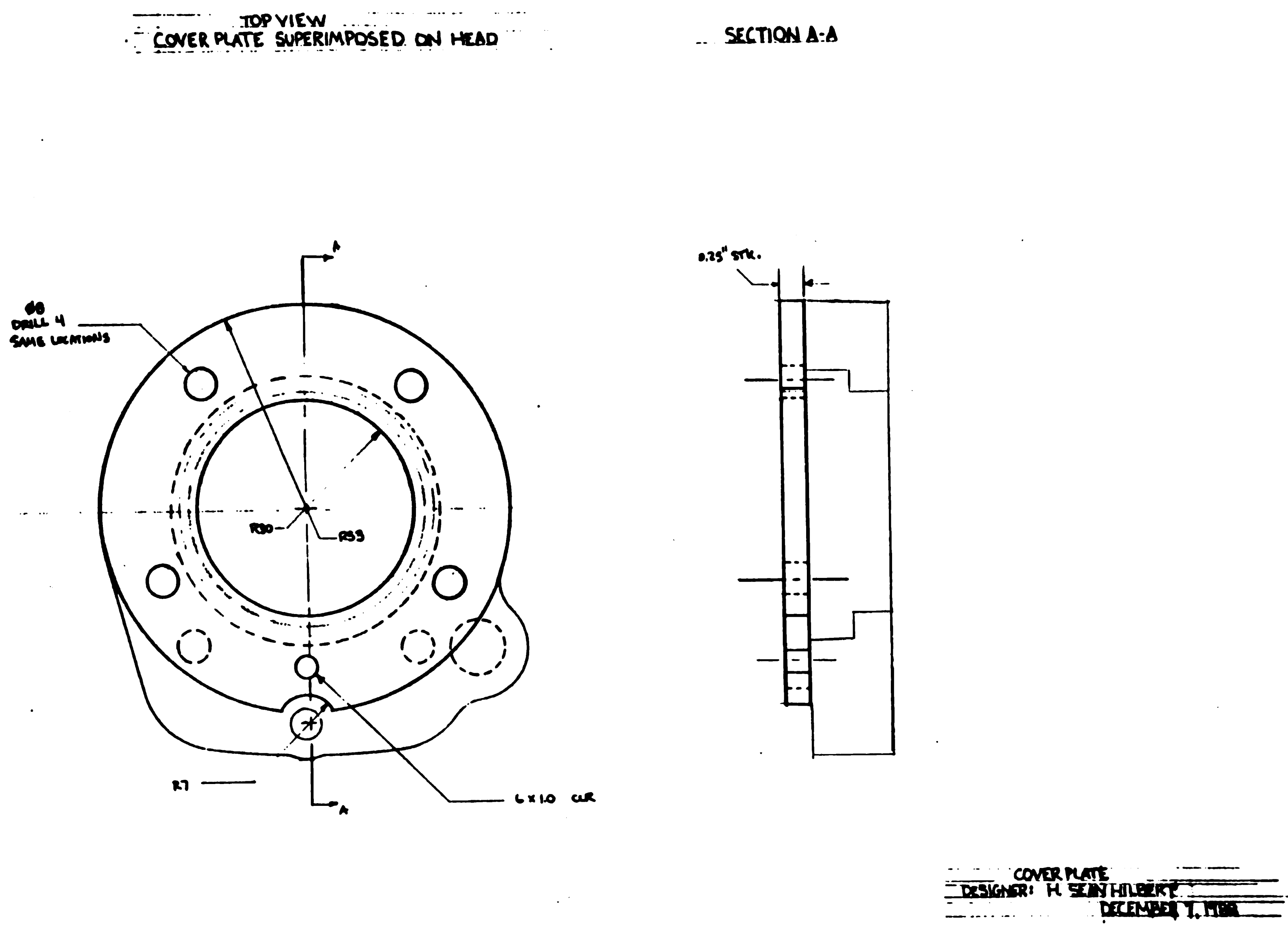


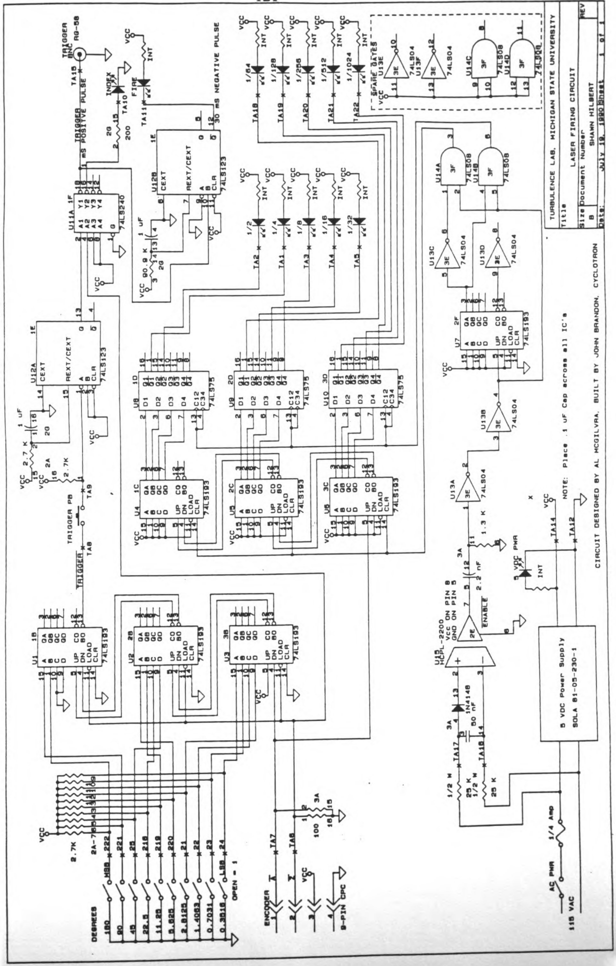
Figure D1. Structure of program VORTICITY.

APPENDIX F

ENGINEERING DRAWINGS


BOTTOM VIEW
NOTE: NO HIDDEN LINES SHOWN
SECTION A-A
ϕ8 DRILL
2X 27
17
A
R28 BORE
R53
6X1.0 TAP
2X R40
ϕ41
R19
32
19
17
24
92
14
2X 8
2X 29
2X 21.5
25.5
13
GROOVE FOR ϕ2.50" O-RING
21
10
71
10 DEEP DRILL BOTTOM
OPTICAL HEAD FOR KAWASAKI KX-125
DESIGNER: H. SEAN HILBERT
DIMENSIONS IN MM
DECEMBER


TOP VIEW
COVER PLATE SUPERIMPOSED ON HEAD
SECTION A-A
0.25" STK.
DRILL 4
SAME LOCATIONS
R30
R55
R7
6 X 1.0 CLR
COVER PLATE
DESIGNER: H. SEAN HILBERT
DECEMBER


TURBULENCE LAB, MICHIGAN STATE UNIVERSITY
Title LASER FIRING CIRCUIT
Size B Document Number SHAWN HILBERT REV
Date: July 19, 1990 Sheet 1 of 1
NOTE: Place .1 uF Cap across all IC's
CIRCUIT DESIGNED BY AL MCGILVRA, BUILT BY JOHN BRANDON, CYCLOTRON