AFOSR-TR- 87-0925

United States Air Force

Office of Scientific Research

Contract F49620-86-C-0093

# THE WHOLE SKY SENSOR

Phase I

Final Report

28 February 1987

James K. Rocks

Principal Investigator





Micro Science Inc. Route 4 Box 286 Leesburg VA 22075

# DISCLAIMER NOTICE

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

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION: Unclassified | 1b. RESTRICTIVE MARKINGS: AD-A179 271 MAR 3 1987 |
| 2a. SECURITY CLASSIFICATION AUTHORITY: - | 3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; Distribution unlimited |
| 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE: | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S): 870228-1 | 5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 87-0345 |
| 6a. NAME OF PERFORMING ORGANIZATION: Micro Science Inc. | 6b. OFFICE SYMBOL (If applicable): |
| 7a. NAME OF MONITORING ORGANIZATION: AFOSR | |
| 6c. ADDRESS (City, State, and ZIP Code): Route 4 Box 286 Leesburg VA 22075 | 7b. ADDRESS (City, State, and ZIP Code): Bldg 410 Bolling AFB DC 20332-6448 |
| 8a. NAME OF FUNDING / SPONSORING ORGANIZATION: AFOSR | 8b. OFFICE SYMBOL (If applicable): NC |
| 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-86-C-0093 | |
| 8c. ADDRESS (City, State, and ZIP Code): Bldg 410 Bolling AFB DC 20332-6448 | 10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 3005; TASK NO. A1; WORK UNIT ACCESSION NO. |
| 11. TITLE (Include Security Classification): The Whole Sky Sensor--Phase I (U) | |
| 12. PERSONAL AUTHOR(S): James K. Rocks | |
| 13a. TYPE OF REPORT: Final | 13b. TIME COVERED FROM _____ |
| 14. DATE OF REPORT (Year, Month, Day): 86-02-26 | 15. PAGE COUNT: 59 |
| 16. SUPPLEMENTARY NOTATION: | |
| 17. COSATI CODES: FIELD / GROUP / SUB-GROUP | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Instrument, meteorology, clouds, height, velocity, cover, computer-controlled |

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

See reverse

| | |
|---|---|
| 20. DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | 21 ABSTRACT SECURITY CLASSIFICATION: Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL: Lt Col James P Koermer | 22b. TELEPHONE (Include Area Code): 202 767-4960 / 22c. OFFICE SYMBOL: NC |

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

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Block 19:ABSTRACT

The height and velocity of visible clouds and percent cover at several altitudes over a portion (28 degree cone) of the sky can been determined from the ground by means of triangulation with an instrument consisting of a pair of CID cameras and a computer. A system has been built which is stand-alone, automatic and produces reports every ten minutes. Results are repeatable and accuracy as determined by indirect calibration is within ten percent. The system can operate at night and in light snow or rain. The cameras can distinguish features in a cloud cover that is featureless to the naked eye. Cost of the instrument should not exceed $10,000.

A simple algorithm for the corresponding of points in 3-space has been found. There is some indication that types of clouds can be identified from an examination of the population statistics of the images.

The system should include shelters for the instruments and an auto-iris attachment. More experience with different kinds of clouds and extensive field testing are required. Algorithms for slant range viewing, non-horizontal cloud forms and time series methods to handle low altitude images should be developed.

## Table of Contents



## Annexes

# FINAL REPORT

## The Whole Sky Sensor Project--Phase One


## ABSTRACT

The height and velocity of visible clouds and percent cover at several altitudes over a portion (28 degree cone) of the sky can been determined from the ground by means of triangulation with an instrument consisting of a pair of CID cameras and a computer. A system has been built which is stand-alone, automatic and produces reports every ten minutes. Results are repeatable and accuracy as determined by indirect calibration is within ten percent. The system can operate at night and in light snow or rain. The cameras can distinguish features in a cloud cover that is featureless to the naked eye. Cost of the instrument should not exceed $10,000.

A simple algorithm for the corresponding of points in 3-space has been found. There is some indication that types of clouds can be identified from an examination of the population statistics of the images.

The system should include shelters for the instruments and an auto-iris attachment. More experience with different kinds of clouds and extensive field testing are required. Algorithms for slant range viewing, non-horizontal cloud forms and time series methods to handle low altitude images should be developed.

## 1. Objectives of the study

The purpose of this project was to construct an instrument consisting of two electronic cameras and a computer, to test its utility and effectiveness in determining the height and velocity of clouds in the sky and per cent cover, and to test, modify and as required add to a collection of computer algorithms which dealt with correspondence, pattern recognition and interpretation in the simulation mode.

## 2. Description of the instrument

### 2.1 Hardware:

The instrument consists of two CID cameras mounted 963.52 feet apart at the same elevation, 240.0 feet ASL, controlled by and transmitting data via coaxial cable to a computer which analyzes the data and produces reports every ten minutes of the height of visible layers of clouds, their velocity and direction and per cent cover.

Each camera is mounted in a gimballed frame and leveled to point straight up. It is also aligned so that its x-axis passes through the center of the other camera. The measured line of the x-axis lies N 62o25'04"W true. The survey was done by the licensed firm of Chamberlin, Wolford and Chin of Leesburg, VA. A map and details of the installation are given as Figure 1.

The cameras are General Electric Model TN2505 surveillance cameras each employing a 360 by 240 CID chip and several features which permit time exposures and auto-iris control.

Three 75-ohm coaxial lines and 60 htz 110 volt power are connected to each camera. The coaxial lines carry the analog signal (equivalent to a cable TV signal) to the computer, a clock signal for synchronizing the cameras with the computer, and control line for operating the "inject-inhibit" (time exposure) feature. A conventional converter at each camera converts 110 v house current to 12 v. dc for the operation of the cameras. The auto iris feature is not implemented.

The lenses are conventional 15 mm f16-f1.7 Fujinon TV lenses. No optical (light) filters have been tested, although the spectral response of the CID chips extends well into the infra-red so that additional information is available by selection of that portion of the spectrum.

In the computer, two boards, one for each camera, are installed to interface the cameras to the computer. These boards or "frame-grabbers" are manufactured by Epix, Inc. Each contains a flash analog-to-digital converter which converts the analog TV signal from the camera into a series of 86400 8-bit numbers (0..255) per frame and stores these in a 1 megabyte RAM 30 times per second. The boards also

output the received signal to a standard TV monitor so that what the camera sees can be seen by the operator. After the frame-grabbers collect two complete images from each camera 2 seconds apart, the computer transfers the data from the frame-grabber ram to computer ram for processing.

The computer is a PC's Limited 286-12 AT personal computer operating at 12 Megahertz, about the fastest inexpensive computer available. It features a 20-megabyte hard disk, an 80286/80287 chip set and 1 megabyte of RAM. A number of other features required for program development are included, but are not necessary for automated instrument operation itself.

Specifications of the cameras, frame grabbers and computer are given in Annex 1.

2.2 System software:

System software consists of a standard MS-DOS package, Epix's SVPCIP system which controls and permits access to the frame grabbers, the Run-C Professional C interpreter (RCP), used to edit and test program modules and function in interpretive mode, and the Lattice C compiler which is used to compile into machine language programs written and debugged in RCP. All of these have been installed in the computer and used as required during the development process. In a field instrument, portions of these will be transferred from their environments and compiled into a single machine language program eliminating the need for keyboard management, the current mode of operation. Specifications of these systems are given in Annex 2.

2.3 Application software:

The programs written by the Principal Investigator for this instrument are in three groups. The first group of routines controls the acquisition of cloud images, turning the cameras on, taking single frame pictures, and storing part of each of the four pictures into a single file on hard disk so that it can be shown on the monitor if required.

The second group calculates the altitudes of the cloud layers from the images, then their velocities, then converts these numbers into a report which is displayed on the screen, and can be printed or transmitted by modem.

The third group is a collection of service routines which provide for display of the various images, graphing of results, lens compensation, and smoothing the signals by means of a fast-Fourier algorithm, and .BAT procedures for data management. This last group is not part of the operating system, but was written to assist in its development. The programs are listed and described in detail in Annex 3.

2.4 Operation:

At start-up, the image is displayed, irises and inject-inhibit set for

the proper exposure, and automatic operation started. In normal operation, the instrument runs all the time. A clock in the computer is queried continuously for the end of the inter-observation interval (usually 10 minutes).

When this time is reached, the image is sampled, the end of the next such interval is calculated and stored, and the end of the inter-sample interval (usually 2 seconds) is calculated and stored. Then a single-frame picture is taken by each camera simultaneously, converted by that camera's ADC and stored in grabber ram (GRAM).

At the end of the inter-sample interval, another pair of pictures is taken and stored at a different location. Then GRAM is copied to computer ram and the two pairs of pictures analyzed, producing the layers and the velocities of the layers and an estimate of cloud cover. When analysis is complete, the report is prepared and transmitted or displayed and the program waits until the end of the inter-observation interval, at which time the process repeats.

In this Phase I development the procedure has been to set manually the irises on the basis of the pictures being displayed continuously on the monitor. When the exposure is correct, the automatic operation begins. With the exception of auto-iris, which is not implemented in this system, the automatic system is identical to the developmental system at this time.

## 2.5 Procedures:

The procedures divide into system control, data management, overlaying algorithms and report preparation.

System control consists of querying the realtime clock and executing the proper routines at the proper time as described above, and includes start up.

Data management is more complicated for two reasons. First, the largest data object that can be handled easily by the system software is limited to 65536 bytes, while the image produced by the cameras is 86400 bytes. Hence the image must be dealt with in chunks. Similarly, the "window" from grabber RAM into computer RAM is also 65536 bytes, thus each image must be partitioned, copied 21600 bytes (the most convenient chunk) at a time into computer RAM, compressed, transformed to integers, and filed as a record for monitor display if desired.

Secondly, because of optical distortion caused by the fact that the lens of the camera is not a pinhole, and because the corners of the image do not receive the same light intensity as the center, only the center 300 columns of the center 120 rows are usable without substantial compensation. This subset is extracted from the raw data and refiled as arrays of integers for later use. Thus the data goes through a number of steps between each analytic sequence, and a number of functions are devoted entirely to this activity.

the proper exposure, and automatic operation started. In normal operation, the instrument runs all the time. A clock in the computer is queried continuously for the end of the inter-observation interval (usually 10 minutes).

When this time is reached, the image is sampled, the end of the next such interval is calculated and stored, and the end of the inter-sample interval (usually 2 seconds) is calculated and stored. Then a single-frame picture is taken by each camera simultaneously, converted by that camera's ADC and stored in grabber ram (GRAM).

At the end of the inter-sample interval, another pair of pictures is taken and stored at a different location. Then GRAM is copied to computer ram and the two pairs of pictures analyzed, producing the layers and the velocities of the layers and an estimato of cloud cover. When analysis is complete, the report is prepared and transmitted or displayed and the program waits until the end of the inter-observation interval, at which time the process repeats.

In this Phase I development the procedure has been to set manually the irises on the basis of the pictures being displayed continuous'y on the monitor. When the exposure is correct, the automatic operation begins. With the exception of auto-iris, which is not implemented in this system, the automatic system is identical to the developmental system at this time.

## 2.5 Procedures:

The procedures divide into system control, data management, overlaying algorithms and report preparation.

System control consists of querying the realtime clock and executing the proper routines at the proper time as described above, and includes start up.

Data management is more complicated for two reasons. First, the largest data object that can be handled easily by the system software is limited to 65536 bytes, while the image produced by the cameras is 86400 bytes. Hence the image must be dealt with in chunks. Similarly, the "window" from grabber RAM into computer RAM is also 65536 bytes, thus each image must be partitioned, copied 21600 bytes (the most convenient chunk) at a time into computer RAM, compressed, transformed to integers, and filed as a record for monitor display if desired.

Secondly, because of optical distortion caused by the fact that the lens of the camera is not a pinhole, and because the corners of the image do not receive the same light intensity as the center, only the center 300 columns of the center 120 rows are usable without substantial compensation. This subset is extracted from the raw data and refiled as arrays of integers for later use. Thus the data goes through a number of steps between each analytic sequence, and a number of functions are devoted entirely to this activity.

time, and the values in the cells of one subtracted from the matching cells of the other. The difference for each cell is squared and summed and the total divided by the number of cells which were differenced for this offset (ofs). In computerese

```
for each offset(ofs);          ofs=0..149
   for each row(i);            i=0..59
      for each column(j);      j=0..149
         del=zimg0(0,i,j)-zimg0(1,i,j+ofs);                    (2)
         sumsq=sumsq+del*del                                   (3)
      next column
   next row
   zofs(ofs)=sumsq/(60*(150-ofs))                             (4)
next offset
```

The minima among all the zofs are the layers. They are in order from highest altitude to lowest (since ofs increases with decreasing altitude). Those ofs which are very large (a low layer) are cast out, since the number of points is insufficient to be reliable, and the best five of those remaining are taken to be the layers. There may be only one or even zero layers, if the sky is clear.

The reason the minima are the layers can be seen from a consideration of either array. Each is, if the images did not overlap, a purely random collection of points relative to the other, so that at any arbitrary shift (the value of ofs), the difference in magnitude is a normal random variable. Hence the zofs for that shift will be close to the mean of the array.

But at some offset, a number of points will line up; their differences are no longer random. Hence zofs for that offset is biased downward. Since clouds are in general bands of points of roughly the same magnitude, a plot of zofs with ofs is largely a smooth curve but with sharp bottoms. The offsets at which these bottoms occur are the locations of layers and replace the term (U1-U0) in (1) above. That is (U1-U0)=minofs where minofs is the value of k at a local minimum of zofs(k), k=0..150.

The lowest layer that can be corresponded reliably can be computed from (1) above as 720*Sx/280 or 2.6 times the distance between the cameras. This altitude, 2600 feet, is too large for judging the height of low ceilings as, for example, in airport operations, so that the cameras need to be brought closer together and the focal length of the lens made shorter.

Both these approaches have negative aspects. Shortening the base line reduces the accuracy in measuring the height of very high clouds, while decreasing focal length (the practical optical limit due to reflection is now 8 mm) gains only a factor of 2. A combination of an 8 mm lens, and a base line length of 155 feet gives a minimum altitude of 200 feet, quite usable. Unfortunately, at 23,000 feet, the one-pixel resolution is about 50% of the altitude.

A solution for low altitudes which has not been worked out is a time

series approach. In this approach, a picture is taken from one camera and the radial velocity of the cloud layer determined. In principle, the picture will not change greatly as it moves across the sky, and eventually some part of it will appear in view of the other camera (unless it is moving directly across the base line.) Then, being seen again, some estimate of its altitude can be made by a sort of time lapse correspondence process.

A fourth option and probably the best compromise is to maintain a baseline sufficient to give required accuracy at high altitudes, then tilt the cameras toward each other until reliable overlap occurs for a layer at 200'. Distance apart (Sx) for overlap at such an altitude depends on focal length, FL, number of pixels in the x direction of the chip, Npx and angle of tilt, A:

$$Sx(Zmin=200') = 2*Zmin/\tan(90-2*\mathrm{atn}(Npx/2/FL))$$

where tilt angle A=Npx/2/FL equals half the apex angle so that the zenith is visible at one end of the chip.

Accuracy of an observation of altitude is the fraction e where

$$e = (Z(du)-Z(du+1))/Z(du)$$

and du is the offset. Here always, du=(1-e)/e. Then the altitude above which the accuracy of the reading is less than e is

$$Z(e) = (FL-Sx)*e/(1-e)$$

Since the field subtended by Npx pixels decreases as FL increases, altitude for a given error and given Zmin does not decrease as fast as FL increases. Doubling focal length, from 8 to 16 mm say, halves Sx from 533' to 213', but decreases the 5% Z from 10105' to 8085', only 20%. Further as focal length increases, spherical aberration decreases, and more of the chip can be used without compensation.

Hence for low altitude work, a reasonable siting policy is to select a longer focal length, a shorter base line and a tilt equal to half apex angle and accept imprecision at higher altitudes. If high altitude accuracy is required, a third camera, C, is added to the end of the line A,B,C located so that its field of overlap with A begins where the accuracy of the pair AB drops below the acceptable level. That is, its location is

$$Sx(AC) = [2*(e/(1-e))*FL*2*Zmin]/[\tan(90-2*\mathrm{atn}(Npx/2/FL)**2]$$

The necessary transformation of coordinates of a point U,V in the tilted chip plane to a point Q,R in an imaginary horizontal plane is computed as follows:

h= FL/cos(L) where A is the tilt angle, and FL the focal length,
B= atn(U/FL) where U is the x coordinate of the point in the chip.
Q= h*tan(A-B) where Q is the new x coordinate of the point and
R= V*h/(FL*cos(A-B)/cos(B), where R is the new y coordinate.

The overlaying process is applicable in the calculation of dx and dy, the velocities of the layers. First however, the points that "lined up" when a given layer was found must be identified. These are, of course, those points with small del, since these small dels were what reduced zofs in the first place. Hence, one zimg array is placed at that layer's ofs over the array from the other camera for the same time and all points for which del is "close" to zero are copied into a new array, vimg(0,i,j), while the others are set to the mean for the array. "Close" is taken arbitrarily to be the square root of variance of the values of the array around its mean, about one standard deviation.

If a copy occurs and the value is not "black" (meaning blue sky can be seen), a counter "cover" is incremented, and eventually divided by the number of points compared, giving the fraction of the sky that contains a cloud at this level provided it is not obscured by a cloud at a lower level. Later, in the Report writing routine, the fractions are changed to the standard meaning of cover.

The layer-finding procedure is performed on the arrays zimg from both cameras for the next time period, producing another array vimg(i,i,j), where again, the first index is the time period. Now vimg(0) is shifted over vimg(1), in a rectangular outward moving spiral pattern until a minimum is found that is not replaced for two full circles. The assumption is that there is only one minimum dx,dy in the layer, but if there are more than one, the true one is likely to be the one nearest 0,0. The algorithm is described in more detail in Annex 3.2.8.2.

The terminal values of du and dv are then passed along to Report() where they are adjusted for pixel width and altitude of the layer to dx and dy, and velocity and direction are computed approximately as

```
velocity(layer) = sqrt(dx*dx+k*dy*dy);
direction(layer) = atn(dy/dx)+angle of the base line.
```

(Accommodation is made for the pair of equal values of atn in the circle and for the fact that dx or dy may be zero.)

An error exists in these algorithms for the following reason. If the lenses were pinholes and the pixels square, the points that line up would have displacements from the center of the camera exactly proportional to their displacements in x and y from the "master" camera. The non-squareness of the pixels (1.17 to 1) is easily fixed, but the lens is not a pinhole. Hence motion of points in the periphery of the image for unit velocity, which is assumed constant for all points in the sky at the same altitude, is less than the motion of points in the center of the image. Hence there is some "blurring" of the overlay, since the entire image is compared. However, the error appears to be small--velocities are perhaps a percent low, and altitudes about a percent high.

There does exist a "worst case" condition under which the dxdy algorithm will fail. If the layer consists of one small cloud moving

at more than its radius in two seconds, there will be no correspondence. However, in this case, either the cloud is so small as to be insignificant, or else it is very low and moving so fast that the instruments themselves will likely have blown away. This does remove the problem of landing aircraft, birds, bugs and blowing leaves, however. If the problem is significant, the inter-sample interval can be reduced.

Percent cover is simply 100 times the ratio of number of cells in velocity image which are not "black" to the total number of cells, 9000, in the array. For each layer, the fraction of cover is 1 minus the ratio of the number of cells not included in this layer but included in lower layers. An error that exists here is that the image is not of the whole sky, but rather of a rectangular cone about 28 degrees in apex angle. Over time, however, if there is any wind aloft, clouds in an unobserved part of the sky will move over the instrument and become visible, so that a time integral becomes a suitable measure.

The computation of standard cover proceeds as follows:

```
cover(this layer) = cover(next lower layer) +
                    cover(this layer)*(1-cover(next lower layer))
```

where cover is the fraction derived in the function "makevimg" described above. There is some question that this is in fact a good estimate of cover, particularly for high scattered layers above broken layers. More experience with skies with these particular characteristics will be needed.

## 4 History of the Phase I Study

The work of the Phase I study was performed in four stages, essentially as described in the proposal.

The specifications of a number of electronic cameras were evaluated and the choice was made to employ the General Electric TN2505 CID units which seemed to have the best characteristics and the highest evaluation by users for sensitivity, uniformity, range and absence of artifacts in the images produced. A system integration firm, Poynting Products Inc, of Chicago packages the grabber boards made by Epix, Inc with their software and GE cameras for the surveillance market. After some negotiation and preparation of specifications for the WHSKY application, two units were ordered subject to the ability of the cameras to transmit the data 600 feet in coaxial cable. Experiments by Poynting being favorable, the equipment arrived at Micro Science in early October.

Meanwhile, a computer, the PCs Ltd 286-12, was purchased by mail order. After delivery it was assembled and the various software system packages installed. A "C" interpreter and Lattice C compiler were procured and installed. The principal investigator then integrated the Run C Professional, Lattice C, SVPCIP (Epix's demonstration program) with the computer, arranged for and supervised the survey of the base

line, constructed the stands and fences around them, procured, laid and buried 4000 feet of coaxial and 110 v. power cable, and designed the mounts for the cameras.

It was originally anticipated that the mounts would be constructed of jig stock by a machine shop, but the cost was unacceptably high, so the PI fabricated them in Micro Science's shop using high-quality plywood. As it turned out, the precision and rigidity of the mounts were more than sufficient for the current effort. Intermittently during this period, the algorithms of the simulations were recoded.

When the Poynting hardware and system software arrived, the boards were installed in the computer and connections made. Unfortunately one plug was inserted upside down blowing one of the boards which had to be replaced. Later, a different kind of failure on another board required it to be replaced. Then at various times both cameras had to be replaced.

A very small portion of the Epix software, called SVPCIP, is actually relevant to the WHSKY project. All of the functions used except invocation of the grabber boards themselves have been replaced by Micro Science programs; the major effort has been to figure out how to get at the boards without having access to their source code, which is for sale, but at a prohibitively high price. If Poynting and Epix are to be used again, particularly for a commercial product, a substantial negotiation will be required to decide how to split down the interests.

During October and early November, attempts were made, first with one board then the other and one camera then the other to solve two apparently nearly intractable problems. One was to adjust the image in such a way that the sky as seen through a standard lens (which simulates the eyeball) was transformed to appear as if it were seen through a pinhole. The reason for doing this is discussed in detail in Annex 3.4.2 Adjustlens(). The second problem was to find a way to handle substantial variation in the signals from the same sky as seen by two cameras 1000 feet apart. The solution is also described in Annex 3.4.1. Filter(). As it has turned out, neither solution was required, since the problems are insignificant, easily solved in much simpler fashion.

By mid-November, the instrument itself consisting of the two cameras, the mounts, the stands, the cables, the boards and the computer was completely assembled, installed on the test bed and made operational. Serious development of the algorithms began. Although the development went through some 20 editions of the code, the final product, which is described in Annex III is essentially what was planned in November. The demonstration program presented in early January made liberal use of "filter". This has since been scrapped and replaced by modifications of the two major overlay programs which have the effect of producing filtered images. It is possible that in some cases of layers with relatively few points (the layer is heavily obscured by lower layers) that some kind of filtering routine will be needed to reduce the number of minima (which give dx or dy) to one. To

date, no cloud pattern has been recorded which is not tractable to existing algorithms.

By the end of January the program was complete except for cleanup and reorganization. The current operational version is listed and discussed in Annex 3. Due to the delays and problems described above, plus an unanticipated paucity of the kind of skies which contained the clouds of interest, extremely cold weather in December and two feet of snow in January, studies of summer cumulus, thunderstorms, lenticulars and other such exotic forms did not take place.

On the other hand, a great deal of work was done with stratus and, slant-range from indoors, on snowing clouds. The cameras do not work well below freezing, and since they are not sheltered cannot be left out. The kinds of problems which remain to be dealt with, as described in the proposal for stage 3 of this Phase I, are addressed in the new proposal for Phase II.

In summary, the project did not present any particularly unexpected or unsolvable problems, and has been completed within the original time period, within funds, even though a no-cost extension, requested when such an outcome did not seem too likely, was granted.

## 5 Findings of the Phase I Study

While the primary goal of Phase I was to get an instrument assembled and into operation with existing algorithms, which was achieved, a number of discoveries were made which are of interest. These were primarily of the form, in what way does the real sky differ from the simulated skies that have been used heretofore? The general answer is, in quite a few ways, but in the direction to make analysis easier rather than harder than expected.

Admittedly, part of this simplification was the result of certain unexpected characteristics of the overlay algorithms whose behavior has only recently been understood. The complexity of the problem turns out to be smaller than anticipated as well because of the number of points in the sky directly overhead, that is, in the cone of view (28 deg. apex angle around the zenith).

Since we are not normally in the habit of looking straight up at the sky, we imagine cloud structures to be complex, as seen in the first 20 or so degrees above the horizon where we see the sides of clouds and a far greater expanse of sky and number of clouds than is actually directly overhead. Hence a given observation is a very small sample of the visible sky, and therefore quite homogeneous.

Subsequent studies will want to combine time series of the local area of visibility, developing changing patterns from a series of individual reports rather than trying to grasp the entire sky from one picture.

Another discovery of interest is that because the CID chips in the cameras have a much wider spectrum of frequency sensitivity, cloud layers, particularly stratus, which appear completely blank to the naked eye, do in fact have quite elaborate structure, allowing for triangulation under unexpected conditions. These may be holes through which the cameras see overlying layers. Even a completely clear sky will, at sunset or dawn, display a granular image, this being the reflection of sunlight from the ridges and surfaces of the top of the smog layer which is normally invisible.

Furthermore, clouds can be seen even when light snow or rain is falling, and experiments with the time-exposure feature of the cameras show that stars themselves are visible on a dark clear night. Pixel electronic noise is severe, however, and further work needs to be done to implement methods to cancel out the noise.

It is certainly clear that cameras must be equipped with lenses with auto-iris. The cameras and the Poynting hardware make provision for the control of "motorized" irises. It is simply a matter of adding the feature.

The cameras and their stands must be sheltered in some way. We propose a shelter resembling a miniaturized version of the conventional Post Office street corner mailbox, which has a pull-down letter chute and a round top. Suppose the door were hinged to the top of the slot and the camera mounted inside looking not straight up, but at a small angle through the doorway. When it was time to take a picture, a fan inside would start, blow the door open and the airstream would blow away any snow or rain or other foreign matter from the opening. After the picture was taken, the fan would turn off letting the door close. Some arithmetic (see 2.6) would be required to perform the necessary transformation of the image into what would be seen if the axis were perpendicular, but other than that there seem to be no problems of significance.

The actual application and system software package for an operating system can be much smaller than that used in the development system. No hard disk would be required, for example, but rather the code could be written to ROM (read-only memory) and a much simpler computer used. Likewise, there is no need for 2 megabytes of grabber ram, nor a monitor. In the most likely application, the signals would be transferred by telephone line to the point of use, or combined with other messages from an automatic weather observation station. A large block of ram could be provided in the hardware to accumulate data from successive pictures, more or less like a flight recorder, so that the newest data simply overwrites the oldest. This would allow rather sophisticated time series analyses to be made without requiring any mechanical storage device.

Finally, we sense from the kinds of problems encountered that the algorithms are quite robust, that is, not easily thrown off track by unexpected conditions. A great deal of data is thrown away or aggregated, but the sky is, at least to the degree that it has been studied, not a difficult phenomenon. While it is a statistical

phenomenon, in aggregate the characteristics of interest are easily found. In spite of their apparent chaotic and turbulent nature, clouds are in fact computationally tractable.

## 6 Conclusions

A prototype Whole Sky Sensor has been built and tested and while there has been a very limited amount of real time utilization, has proven adequate for the task of determining height and velocity of layers of clouds. Considerable additional research, particularly into cloud types and formations of interest to the micro-meteorologist needs to be done, and the device needs a suitable shelter.

The next stage of development should be directed toward answering these requirements, with extensive field testing under user management, further algorithmic development, and interfacing with existing met systems. As a part of this phase, it should be possible to design a unit for general commercial production and installation.

# Annexes

## A1: Equipment specifications

### A1.1 Cameras

The General Electric 4TN2505A3 Solid State Surveillance cameras contain a Charge Injection Device (CID) chip with a 248(V) by 388(H) pixel array. The pixels themselves are 27.3 uM(V) by 23.3 uM(H). The width of the usable portion of the chip (360 pixels) is 8.388 mm. This divided by the focal length of a 16 mm lens gives the subtended angle to be arctan(8.388/16)=27.66 degrees. At 10,000 feet above the camera, the width of the field is thus 5242 feet long (along the x-axis) and 240/360*1.117*.32425 = 3904 feet wide. A pixel subtends, at this height, in x, 5242/360 = 15 feet, and in y, 16 feet.

Relevant pages from manufacturer's data sheets are attached at the end of this Annex.

### A1.2: Frame grabbers

There are two grabbers, identical except that one is programmed to respond to signals directed to the "master" and the other as "slave". This discussion describes either one of the units, except for that difference.

The grabber, manufactured by Epix, Inc. is a printed circuit board which occupies one of the two short slots in the computer, sharing the data bus. It is powered by the computer power supply and SNC connectors provide access for signals entering and leaving the grabber. The board includes a flash 8-bit ADC, a ROM chip holding certain proprietory software, a RAM chip by which instructions for a specific application can be set into the grabber from the computer, 1 megabyte of data RAM, and the usual ancillary components.

When the board is invoked, parameters are stored in the RAM and if set to do so, the analog TV signal, which streams from the camera into the ADC continually as long as the camera is powered, is routed as a string of numbers from the ADC to a location in the grabber data ram. Every frame (equivalent to a TV picture frame) the previously written data is overwritten by new data.

However, a software function can be used to change the starting address of the frame so that N pictures can be written to ram where N= 1048576 modulo frame size, in bytes, so that a kind of "movie" can be accumulated, or several frames in sequence stored for later analysis.

The picture is simultaneously passed through a digital to analog converter (back to its original form) and output as a signal available to drive a standard TV set, the monitor, on which the user can observe

the picture being taken.

Once the desired picture is in the grabber ram, the input to the area is disabled and the resulting block of memory can be transferred elsewhere in the computer or stored to disk.

A very large number of abilities have been programmed for this device by Epix, but most are unused in this application. Relevant data sheets from the manufacturer's documentation are attached at the end of this Annex.

### A1.3: Computer

The PC's Ltd 286-12 computer acquired for this project is the fastest of their family of IBM Personal Computer compatibles, and as of last August, the fastest such machine available from anyone. By Epix's algorithm (which determines the clock and processing rates of the CPU in order to manage its own firmware) the 286-12 is very nearly 4 times as fast as a standard 8086-based IBM PC, twice as fast as an IBM AT and is a third the price of the latter. Nonetheless, it has proven entirely reliable, and the 12 megahertz clock rate has made child's play of the necessary number crunching and array manipulation.

Relevant pages from the manufacturer's specifications are attached at the end of this annex.

### A1.4: Installation

The cameras are mounted in semi-gimballed frames which are placed on the stands at either end of the baseline. Since these may not be visible from one another, adjustments need to be made to assure that the cameras are in line and looking straight up. This latter is done by "swinging" the cameras on their gimbals so as to achieve registration at various angles along the line, thus assuring there is no tilt of the z-axis around the x-axis, and that the x-axis passes through the centers of both cameras.

Then the z-axes are set perpendicular by means of a float level. Error in this setting appears as error in altitude of layers. It also appears as a "smearing" of the bottoms of the overlay display, since the effect of an error is as if the sky were tilted along the x-axis in either or both of the cameras. A very precise alignment can be made from stars, since the rays from a star to each of the cameras are essentially parallel. Two images should exactly match (within the resolution of a pixel) if the cameras are set correctly.

A very precise measurement of perpendicularity of the z-axis can be made by the method of bore-sighting. In this method, a straight tube one inch or less in inside diameter 10 feet long is suspended directly over the camera by the top end. Then if the camera is set correctly, the image of the opening at the top of the tube is visible on the monitor through the bottom of the tube. Assuming that the images can be located within one tenth inch, the angular error out of

perpendicularity is not more than 1 part in 1200, or 3 minutes of arc, which is less than the optical width of one pixel. 360 pixels subtend, with a 16 mm lens, 28 degrees or 4.6 minutes per pixel. Hence the boresighting method achieves an accuracy which exceeds the camera's resolution.

## Annex 2: System Software specifications

### A2.1 MS-DOS

This package, the Microsoft Disk Operating System, v. 3.12, was purchased with the computer and is tailored for the PCsLtd 286-12, a machine closely related to the IBM PCAT from whose specifications it has grown. At least part of the virtue of RCP (see below) is the "shell" facility which allows DOS functions to be called from application programs running under RCP. Many of these functions, in particular copy and concatenate, used to combine and store collections of files containing segments of pictures, get considerable exercize.

### A2.2: Lattice-C

The C compiler language, written by Kernighan and Ritchie of Bell Labs in the late '70's primarily for the development of the Unix operating system has become the language of choice for system developers, quickly superceding Pascal, PL/1, Fortran, APL, BASIC and assembly language as the tool to use in the development of compact highly efficient easily written and error-resistnat code.

Because of its popularity, several C-interpreters have emerged in the last few years to provide C programmers with a tool by which programs can be written and executed in the interpretive mode, and then compiled into machine language after all the "bugs" have been removed.

This combination of a powerful language in both the compiled and interpretive forms with easy links between the two and with a large supply of off-the-shelf program packages and programming aids, viz., "C-food Smorgasbord" has made software development rapid and accurate, permitting the analyst, who figures out what the computer is to do, to write his own programs and debug them, module by module as he generates the algorithms and procedures to be implemented, then compiling the specific modules into machine language, and calling them from the interpreter as required.

This procedure is very fast and very efficient, and has been employed in this development. If there is any problem it is that programs are so easy to write and test that they pile up, and keeping track of the procedures that are available becomes a bit of a bookkeeping problem, since large harddisk memories tempt the programmer to save functions which ought to be discarded, simply on the grounds that they might sometime be useful. But this is a small price to pay for the fact that rather large programs can be turned out in short order by a single

analyst working alone, once he has mastered the language.

There are a number of texts on "C" which make the transition to the language not necessarily easy but achievable by anyone with a reasonable amount of programming experience, even if it has been limited to BASIC, an interpretive language. I strongly recommend that any scientist or engineer who has a need for computation try this package.

The Lattice C compiler is neither the easiest nor the fastest C compiler available, but it is probably the most popular, in terms of the number of lines of code that have been written in it. Lattice was used to write SVPCIP and Run-C Professional with which it interfaces nicely, and the large memory and -k2 options are available for the Intel 80286/80287 processor chip pair which exist in the PC's Ltd computer.

### A2.3: Run/C Professional

This C interpreter, written by Walton and Brooks of Lifeboat Associates is very nearly flawless as a program. At every point at which I imagined that the interpreter had made an error, it turned out to be my programming error. If there is a criticism it is that certain kinds of errors, which should be reported as syntax errors, produce major crashes or completely unrelated diagnostics.

Program size, particularly data arrays of the sort used in WHSKY, can be cramping, in part because a third of the memory is blocked off from use by the grabber boards, and by the way RCP organizes memory. On the other hand, with practice it is almost always possible to write programs which run the first time and their compilation takes a couple of minutes at most. This allows the prgrammer to concentrate on the problem to be solved rather than the means of solving it, a highly desirable characteristic of any language.

### A2.4: SVPCIP

This software package, written in C by Epix Associates is designed primarily to be used in testing and evaluating the Silicon Video frame grabber, and to serve as a basis for customizing applications in the surveillance area. It is not a stand-alone program, requiring that the user enter menu selections from the keyboard. It contains a large number of functions of various sorts, such as different kinds of digital filters to be applied to the pictures to enhance viewing, for changing frame size, for rearranging parts of the pictures, zooming and the like.

The package as a whole, albeit a very respectable bit of work, is overkill for this project, but still prevents stand-alone operation which ought to begin automatically with power on, and continue without intervention until power down. Nonetheless, SVPCIP has served this Phase I effort well and without question the minor inconveniences can easily be eliminated, at a price.

## Annex 3: The WHSKY Computer Program

### A3.1 Overall flow:

In the normal mode of use, the instrument runs continuously, making observations every ten minutes as long as power is on. But to get the system going, certain steps are required of the operator. First both cameras are turned on, and the computer powered up. DOS boots control directly to the RCP interpreter by an Autoexec.bat invocation, RCP\RCP (with sizing parameters). Then operator loads and runs RCP\WHSKY.200, the current version of the system. Whsky wakes up the grabber software package, SVPCIP, as a "shell" activity from within RCP. Control is returned to RCP by the user entering "qu" and 2 carriage returns.

Some further initializing occurs in this version before the main loop begins. RCP loads the library of compiled Whsky modules, called FEBLIB.C and listed elsewhere, MATHLIB, a package of mathematical functions and GRAPHLIB, a package of graphics functions. Both MATHLIB and GRAPHLIB were provided with RCP. Then the set of parameters required by the grabber boards is copied from a disk file to the grabbers. These parameters were originally input manually (from the keyboard) and copied to hard disk by a small service routine written for the purpose.

Then the program sends an eight-bit string of flags, hex number 0xac, via output port 0x301 to the grabber boards, telling them to digitize the incoming camera images. The images also appear on the monitor, so that operator can adjust the irises of the cameras. Focus is set to infinity. The program waits for a keyboard input signalling that the cameras are set to the correct exposure. Once that key stroke has been entered, the system runs unattended, looping through the balance of the program unattended as long as power is on.

The first module, Grabimage(), reads a sky from each camera at some time t0 into the two grabber rams simultaneously, and does it again at time t1, 2 seconds later. Before returning, the time of the next observation, t3, is calculated and stored. Then the four images are copied from grabber ram into computer ram by module Gratozimg(), 21600 bytes at a time and a subset of these data points transferred (and transformed) to arrays zimg0 and zimg1 by module Copyimage().

Then the routine Zolay0() runs. The image (now in zimg0[0][i][j] ) from the master board for time t0 is overlaid onto the image (now in zimg0[1][i][j] from the slave board for time t0 to find the values of the difference of the corresponding pixels for each offset. The results are stored in an array zofs0[k], where k=0..149. The process is repeated for t1, using Zolay1() and arrays zimg1[0] and zimg1[1]. Then the module Calcz() executes, finding the minima in the two zofs and after some manipulation, storing them in the two arrays dw[l][t] amd zdw[l][t], where dw contains the index (offset) of each layer, and zdw the value of zofs at that offset. Here, l is the layer number, and the second index is the time (sample number).

After the layers are found, Dxdy1() executes, taking one layer at a time. First it makes two new arrays for each layer, using Makevimg(). Each array is the same points from the same layer from the master camera for two different times. One of these is overlaid in a rectangular pattern onto the other finding the minimum, vofs[du][dv] for that layer which represents its shift in x and y in the time interval, hence its velocity. Then du and dv for this layer are stored in lists du[l] and dv[l] for use by the final module in the loop, Report().

Report() spends most of its time putting together an impressive looking display on the screen. It prints the kind of report (WHSKY), the facility location, its latitude and longitude, height above sea-level, the time and date, and a column stub identifying the data that will subsequently be listed. The items are "cloud base feet ASL", "moving toward degrees true", "velocity knots" and "percent cover". Then the conversions of the dw, du and dv found in earlier programs are made to give usable values for each layer. The formulae are detailed below.

The reason for the rather clumsy heading for wind direction is that in common parlance, winds are described in terms of the direction from which they are blowing, yet clouds and storm systems and fronts are described in terms or the direction toward which they are moving, e.g., "a northwest wind is pushing the clouds southeast", which can be confusing.

The data in this report can be stored as a set of numbers rather than being printed, or can be transferred to a modem for transmission to a manned weather station for interpretation or other use. The present system does not provide this nicety.

After the report is presented, the program reads the time from the clock in the computer, compares it with time t3, the next observation, and if time now is less than t3, repeats the reading. Eventually, time now is equal to t3, control returns to the beginning of the loop and Grabimage() runs again. The interval is usually ten minutes, somewhat longer than it takes to run through the program.

A3.2 Detailed description of the program:

It is recommended that the listing of the program, Annex 4, be studied in parallel with this text, since the purpose of the text is to explain how the code works, and the how and why of certain algorithms which are not always self-evident from the code.

A3.2.1 Preprocessing:

When Whsky is loaded to be executed under the control of RCP, three options are implemented. The group of "#define" statements instruct RCP to replace each occurrence in the program of the first string in the line with the second. This greatly simplifies changes in parameters. For example, the maximum number of layers (nl) that can be found is currently set to 6, but it has been at various times in the

development 1,2 or 3 as well. Since nl shows up many times in the code, it is useful to be able to change it everywhere with just one sdit operation.

Npin (number of pixels in) is one fourth of the TV image of 86400 bytes, that is 240 rows by 360 columns. Since 86400 is a data object too large to be handled routinely by any computer with a 16-bit address bus, and 2**16=65536 does not divide nicely into 86400, and 86400/2 = 43200 is too large to be expressed as a 16 bit signed integer, and because using only the middle third of the image throws away too much usable data at this point, the standard block of data was selected to be of size 21600.

Nsamples, the number of pictures to be taken by a camera at each observation, is actually one larger than the number of pictures used. The reason is given in the discussion of Grabimage() below.

Two useful #defines not here would be nrows 60 and ncols 150, since these two parameters appear throughout the program. It turns out for other reasons that a better selection of the dimensions of an overlay array might have been 120 by 135. Had these defines been installed, it would be easy to change them, and change them back if wrong.

Several variables and arrays are declared "static". These data items are initialized to zero and permanent areas are set aside for the data in memory for all programs to use. Arrays or data items that are local to a particular subroutine are exactly that. Every time the function is invoked, an address for the item is assigned from whatever space is available at the time. By declaring arrays static, their locations are permanently fixed and any function can get at them. This is not true of the compiled functions which have to be told on each call where the data named in the function is stored. This is the purpose of the string of "formal" parameters which follows the name of a called function.

A3.2.2 Data items: (declared static)

time: a floating point number containing a single number combining hours, minutes, seconds and hundredths of seconds, generated by the function btimer(), which reads the computer clock and does the arithmetic.

delaytime: any time which is to be compared with the clock.

time2: next observation, usually ten minutes after this one.

time3: next sample, usually two seconds after this one.

time4: an extended delay due to poor seeing (fog, heavy snow) or transmission difficulties. This would be set by "totlight" which will be included when the cameras are equipped with autoiris, or by "Report" if a modem is in use.

dt2=600: seconds added to time now to get time2.

dt3=2: added to time now to get time 3.

dt4=1200: twenty minutes, added to time now to get time 4.

img0[npin]: a 21600 byte array of characters (the numbers 0..255), into which the grabber panels are transferred and held while being transformed into the zimg's.

zimg0[2][60][150], zimg1[2][60][150]: Two 2 by 60 by 150 arrays of integers which hold selected bytes for each camera for each time. The number in the name is the time, the first index the board, the second index the row and the third index the column. Each array holds 18,000 integers, or 36,000 bytes. Since input values are characters (8 bit strings), these could also be arrays of characters, but for other reasons the bytes coming from the grabbers are converted to integers by Copyzimg() for subsequent use.

vimg[2][60][150]: An array of 2 by 60 by 150 integers which holds the values from zimg0[0][i][j] and zimg1[0][i][j] that are identified as being from a particular layer. The array is reused and refilled for each layer.

zofs0[150], zofs1[150]: Two one-dimensional arrays holding the results of the z-overlaying process, one for each time.

vofs[302]: a scratch string used to hold temporary results from the smoothing function.

A3.2.3 Main()

This line marks the beginning of the program proper. Since "C" is a structured language, the Main program, which must be included, should be short and limited to global management of the process, which it does by calling functions by name which in turn call other functions, each limited to data to which it is given access by the caller, or which is global (as described above). When completely purged of development modules, Whsky's Main does some initialization, sets up a loop to run forever, and then within that loop, calls just seven functions. These are Grabimage(), Gratozimg(), Zolay0(), Zolay1(), Calcz(), Dxdy1() and Report(). Each of these will be discussed in detail below.

The initialization process in Main() involves waking up the grabber software by a "Shell" instruction, loading the libraries, and setting some flags to permit easy exit from graphics programs (which change the screen format) during development.

Let me digress briefly here to present a lecture on graphics. A graphics capability, particularly in systems dealing with an enormous amount of data, is extremely valuable, since relatively short programs can turn thousands of numbers into simple graphs which are easy to understand and which display trends and relationships in the data that are invisible in the numbers themselves. Another advantage is that the display shows program errors and discontinuities in the data, or hardware error in a way that is rarely obvious in the numbers themselves. It is also a first rate marketing or training tool if properly used.

In the operating program, there is currently no requirement for graphics since there will be no screen, but for development it is almost mandatory, at least if the process is to be efficient. Regrettably the IBM graphics package, which is implemented in the PC's Ltd machines, is primitive relative to the very classy graphics of the Texas Instruments Professional Computer, a very sophisticated cousin of the original IBM PC which did not, regrettably gain a sufficient share of the market to survive. End of lecture.

Next, the poweron flag is set to 1, and Initxram() is called. The images are displayed by sending the proper flags to the grabbers. Then a 1024 byte array of data is used by the function Initxram() to copy data from disk to "Xram", the grabber control ram, and located at machine address 0xa0000 (the eleventh block of 2**16 bytes--beginning at decimal 655360. Grabber ram runs from 655360 to 720895, each board has the same addresses, but is selected by a flag fed to the board via a computer output port. Similarly, if parameters are to be changed, they also occupy logical address space beginning at 0xa0000, but another flag is used to tell the boards that the reference is to the data in Xram.

An image is displayed from each camera on the monitor and the computer waits while operator adjusts the irises of the cameras. The desired iris setting provides an average light flux across all the pixels of 128, half the range of the converted number. The gain and offset of the ADC's can be adjusted so that for the same iris opening a uniform sky (either all clear or all overcast) each camera produces the same average. With the lenses in the test configuration, f16 is suitable from two hours after sunrise until two hours before sunset if the sky is not heavily overcast. As darkness approaches, f stops have to be rather rapidly decreased (opening the iris), and after the last stop is reached, time exposures, using the inject-inhibit option are required. Miscellaneous functions, Totlight(), Setiris() and Setii() have been written to be included with any system equipped with auto-iris.

When the irises are set to the satisfaction of the operator, any key entry initiates the endless loop, "while (poweron=1) {...}. Main() proceeds through the functions listed above, ending with report() and then returns to the beginning of the endless loop executing all the functions again.

A3.2.4 Grabimage():

Initially, when the exposure is satisfactorily set, any key can be struck, or thereafter, upon return from the end of the loop Grabimage() executes. The time is determined from the on-board clock by a function "btimer()" which reads the clock and converts the hours, minutes and seconds to a number. Time2 and time3 are computed and stored and the address in grabber ram into which the pictures are to stored calculated.

Initially, the grabbed images, those being observed to set the irises, are stored in grabber ram at address 0 up to the size of the image. The first true image is collected in one frame after the address is changed by Grab(a,b,c), the second true image is collected in one frame and the address increased again. At the third address change, the image is turned on and stays on, being overwritten into this fourth block until the next observation, at which the first true image of the second observation is stored in the second block, and so on. The reasons for these gyrations have to do with the interactions of the various flags and the desirability of being able to see the image in between observations.

The TV image coming in from either camera is essentially the same as any TV transmission protocol. At the beginning of the frame, the pixel in the upper left corner of the chip is read, amplified and transmitted. Then the next pixel in the horizontal row is read, and so on until all 360 pixels in the row have been read and transmitted. The number 360 is one of the parameters fed to the grabber control table by Initxram().

Then a delay occurs, called the horizontal blanking time, while the putative TV receiver moves its beam back to the first pixel of the next row, after which the second row is transmitted in the same way. This process continues until all 240 rows have been transmitted, at which time a longer delay, called the vertical blanking time occurs, during which the beam is returned to the upper left corner of the chip. During transmission, the chip has been gathering light for the next picture.

After conversion to a number in the computer's 8-bit flash ADC, the pixel values are stored in grabber memory cell (x) where x=(row number)*(pixels per row)+ column number. x is actually a single cell address which increments by one with each new entrant, and starts at the address provided to the grabber. This cell can be anywhere in grabber ram, but its value must be transmitted to the grabber, and time must be allowed to change the address. The horizontal blanking time is not long enough for this, but the vertical is, hence the process requires that the beginning of vertical blanking be identified, the input turned off, the address changed during this interval, and then the input turned on again. That is, the storage of numbers coming in must begin with the first number after vertical blanking is finished and terminate for this block of memory when vertical starts. Then the address can be changed and the next block of data stored somewhere else. There is room in the ram for 12 complete 86400 value blocks, but they must not be allowed to overlap, nor can acquisition be started in the middle of a frame reliably.

Thus "grab()" performs four tests. If, when first entered, vertical is off, (vertical retrace is in progress) it is not known how long this has been going on and hence there may not be time enough left to insert the address at which the next picture will be started. Hence the machine first makes sure the video is being recorded in the old address, waits until vertical comes back on and a picture is being input. A bit in an output port of the grabber is zero while input is occurring. At the end of the picture, that bit goes to 1, and the vertical blanking interval has begun. The address of the new block of grabber ram into which the next picture is to be placed is copied into the appropriate registers from the formal parameters a,b,c in "grab(a,b,c)" and the machine then waits until this vertical retrace interval is over (about 1.6 ms.). When the bit drops to 0, the machine tests the bit continuously until the vertical retrace begins again--the bit goes to 1--and at that time turns the video off, so that no more data will be recorded until the address is changed to the next block.

The address of the block into which the frame is placed increments by

86400 bytes per image, beginning at 86400. This is hex 5460 times 4 (a 2 bit-offset is required in the address field.) Hence the addresses are manufactured as a triple of 2-digit hex numbers, aabbcc, one pair to each of three ports, where addr starts at 21600 and increments by this amount each frame, a=addr/65536, d=addr-a*65536, b=d/256, c=d-b*256, all unsigned integers or long (32-bit) integers. Thus the first four block addresses are 0x5460, 0xabc0, 0xfd20 and 0x15180 of which only the first three are currently used, and the hex values of a,b and c are respectively 00,54,60, then 00,ab,c0, then 00,fd,20 and finally 01,51,80. The parameter nsamples is currently 3, but can be increased to 11 if needed to allow a larger number of pictures to be taken at any given observation. For complex or slow moving skies, this may prove useful.

Each board is controlled by the same signals from the computer, but the data are collected independently in each grabber ram from its own camera.

After the first sample is collected by "grabimage()", the interval between samples, normally 2 seconds, is added to the time, and the clock is examined continuously until that time is reached. Then another sample is collected. When nsamples have been collected, control returns to Main() which calls Gratozimg(). There is a possible error in this protocol, but it appears to have no effect on the output.

A3.2.5 Gratozimg():

The job of this function is to copy, from each board, 2 samples of 86400 bytes each through a 65536 byte window, into computer ram, break up the resulting blocks into four 21600-byte long segments each, transform the data items, which are 8 bit numbers (characters) into integers (15 bits plus sign) and copy a selected subset of the data into 2 tables, zimg0 and zimg1.

The problem lies in the fact that the window "slides" along the grabber ram memory space 65536 bytes at a time, while the data from the cameras has been blocked into grabber ram in 86400 byte contiguous chunks. The first chunk begins at address 86400, which extends 86400-65536=20864 bytes beyond the end of the first window. So the window is shifted to 65536, and everything from window + 20864 to window + 20864 + 21600 - 1 copied to a block of computer ram called img0[]. Then the block from 42464 to 64063 is copied into the same area and transfered to zimg0 by the function Copyimage().

There are now only 1472 bytes remaining that can be reached through this window, so these are copied into the low end of img0. Then the window address is incremented by 21600 and the remaining 20129 bytes brought into the high portion of img1[] from the low end of the window, and transferred,again by Copyimage(). This process is repeated for both boards, until 16 records of 21600 bytes each have been input.

A3.2.5.1 Copyimage()

This function transfers only the data in quadrants 1 and 2 (the middle

two) and only the data from times t=1 and t=2, and of these, only alternate rows and and only the alternate columns from 30 to 298, the middle 150 pixels. Thus the 86400 byte picture is reduced to 9000 integers by Copyimage() for each time, for each board. The first valid picture is stored in zimg0[board][row][column] and the second picture in zimg1[board][r][column]. The cast-out data does decrease precision, but adjacent pixels appear to have some sort of electronic bias which produces a "comb" effect. This may be a left-over bit in the ADC. Further experiments will be required to determine the trade-off.

A3.2.5.2 Zimg(t)out():

After the two pictures are taken and transferred to zimg[t][b][r][c], 2 36000 integer arrays are in memory. These are filed as 4 blocks of 18000 integers to a harddisk file rcp\zimg2.

A3.2.5.3 Showcloud():

Next, if desired, the service routine Showclouds() executes, retrieving the files just stored transforming the integers to characters, and re-filing them into 4 21600-byte arrays which are concatenated into a new file RCPZIMG3.000, and returned to hard disk, in which form they can be retrieved by the SVPCIP "file read" instruction and displayed on the monitor, providing operator with the image of all quadrants that are subsequently to be processed.

A3.2.6 Zolay0(zimg0,zofs0), Zolay1(zimg1,zofs1):

At this point, all the pre-processing of the camera image has been completed, and all the data to be processed for this observation are in two arrays of integers, zimg0[board][row][column] and and zimg1[][][], for the two times. First the layers are found by overlaying zimg0[0][][] onto zimg0[1][][], by routine Zolay0 for time 0, and the same layers found again at time 1 using zimg1[0][][] and zimg1[1][][]. The two routines are identical except for the arrays involved.

First, all the values of zimgt[0] and zimgt[1] are summed into two numbers meena and meenb where a and be refer to the boards. Then the normalizing number dmeen equal to meena/meenb is made. Every cell in board b is multiplied by dmeen, resulting in two data sets in zimg0[0] and zimg0[1] whose means are identical. The purpose of this exercize is to compensate for the irreducible differences in the two ADC's or camera voltages which do vary over time even if the gains and offsets of the two grabbers are tuned very closely. The reason for normalization is to assure that if two pixels are of the same point in the sky, the only difference between them will be related to parallax effects, not by camera effects. Since the former appears not be significant, the difference is often very close to zero, enhancing the accuracy of the overlay process.

The next step is the actual overlay process. Since for triangulation, negative overlay is meaningless, zimgt[0] is placed directly over zimgt[1] and shifted "right" one column at a time, the number of columns shifted being the "offset". As the shift proceeds, the sum of

the squares of the differences between corresponding pixels in the two images will change, depending on the number of pixels in the overlay that correspond, that is, report the light received from the same point in the sky. For any given offset, the square root of the the resulting number divided by the number of points compared, is retained in zofst[offset], and the minima in it found later. These minima are the layers.

A3.2.7 Calcz():

After the two Zolay()'s have run, two lists of 150 numbers, zofs0[] and zofs1[] are in memory. The next step is to find the minima and clean up the lists of minima so that as nearly as possible the layers found at t=0 are matched with the layers found at t=1. The first step is to smooth the zofs themselves so the minimum finding algorithm works correctly. This has been done in two ways; either the zofs list can be smoothed with a Fast Fourier smoothing routine, or each point can be set to the average of the five adjacent points of which it is the central value. Both schemes work well. The former is more accurate (returns a better estimate of the true value of the point, assuming that the curve is smooth), while the second is somewhat faster, and takes less program.

A minimum is defined as any point in the set of zofs which is equal to or less than its predecessor and less than its successor. If the curve has a W-shaped bottom which could occur due to noise, it is possible for "minima" to be found two points apart by this rule. The filter technique essentially eliminates W's and thus eliminates these occasional strings. On the other hand, the cleanup routine matches points which are most nearly equal, so that if two nearby points are found by the minimum finding routine, for both curves, two minima will remain only if both sets produce the same pairs, somewhat unlikely. Since the zofs are double length floating point numbers the equality test is not necessary.

When a minimum is found, its index is placed in array dw[l][t], the value of zofs at this point placed in array zdw[l][t] and l, the layer number, is incremented. If l reaches the maximum size of the array, the largest minimum in the array is deleted, and all the minima below it in the list pushed up one. In a reasonably well behaved sky, this is unlikely to happen, since the size of the array is three times as large as any reasonable number of layers (3). On the other hand, it is not yet known what the consequences will be of a towering cumulus whose perpendicular face is visible to both cameras. That could produce a lot of layers, so that exceeding array size is an indicator of a special situation, or cloud type, no necessarily of signal noise.

After both strings are processed and the minima found, the stacks are processed. A square array is filled in with the absolute values of the differences of each layer value in one board with each layer value in the other. Then any row columnn pair intersection which contains a zero is flagged in both row and column. Then each intersection which contains a 1 and is not flagged in either row or column is flagged in both. Then each row column intersection which contains a 2 and is not

flagged in either row or column is flagged. This process reduces the lists to two sets of equal length which contain at equal indices layers of equal or nearly equal offsets.

Offsets in one list which have no match within 2 in the other are discarded. Each list is, of course, the layers found by triangulation from both cameras at one time. The reason for this clean up is that in the next step, the layers must be taken in pairs of nearly equal offsets for the two different times in order effectively to measure velocity.

A3.2.8 Dxdy1():

Beginning with the lowest layer (the largest offset), the images from board 0 for times 0 and 1 are overlaid for correspondence. If for both, zdw (the value of zofs for this layer) is zero (no layer has been entered in this slot) or the value of the offset is greater than 140 (the layer is too low to be have a suffiently large number of points to be reliable) the layer index is decremented and the next layer examined. It is possible for there to be no layers at all, in which case the words "clear" are printed in the report.

When a layer is found, the first step is to make an image restricted to just the points in this layer, so that just these can be overlaid. Function Makevimg(ofs0,ofs1,layer number,zimg0,zimg1,vimg,cover) is used for this purpose. (The string of labels following the function name are the formal parameters which tell the function which is compiled and in the library, the value of the two offsets (one from each list for the same layer, very likely equal), the layer number and the locations of the four arrays containing the data Makevimg will use or make.)

A3.2.8.1 Makevimg():

The function treats the two zimgs in turn. First a normalization number, dmean, is found for zimg0[1][i][j+ofs0] in terms of zimg0[0][i][j] for the area which overlaps (which may be different from the total image). Then a zero-level for this layer is found as the mean of zimg0 for the area. This value is copied to all the cells in a new array vimg[t][i][j] which are not in the overlapping area. Then the standard deviation of all the points in the layer is found as the square root of the average of the sum of the squares of the deltas (the difference between the values at one time and the corresponding values at the second time).

Then the deltas are computed again (they should have been saved, but space is tight), and if a delta is within one standard deviation of the mean of the array the value of that pixel from zimgt[0][i][j] is copied to an array vimg[t][i][j], and (for t=0 only), a counter, cover[layer number], is incremented. Otherwise the zero-level is copied to that cell and cover is not incremented. Cover, therefore at the end of this exercise is a count of all the points in this layer that correspond and when divided by the number of points in the area considered is in some sense a measure of the share of total cover provided by this layer

alone. The pixel value should be compared with a cutoff value, representing blue sky, and the point excluded if it falls below this cutoff. This is not done at present, hence cover over-represents the actual degree of cloudiness by not picking up "holes" in the overcast if they exist.

The same process is performed on zimg1[0][i][j,ofs1] to produce vimg[1][i][j] and two arrays, of the same layer of the overlapping part of the same sky at two different times are ready for analysis. Dxdy1() proceeds to move one image around the other, first with the center of one image exactly on the center of the other, using the smaller of the widths of the images as the limiting width of the two. Again, using this time a function Dvolay(), the deltas are computed for the points in the common area, summed and averaged to produce a value, vofs, for this particular amount of shift of one image over the other.

A3.2.8.2 Dvolay():

In the first call to Dvolay(), there is no shift du=0,dv=0; in the successors, one image is moved first one pixel right du=1,dv=0, and Dvolay() is called to find vofs for the overlapping area for this shift. Then vimg[0] is shifted one cell up (du is now 1 and dv=1) and vofs calculated. Then the shift is two left, then two down, then three right and so on. At each shift,the vofs produced is compared with its predecessor. If it is less, it replaces the minimum and the shift in x and y for this new minumum vofs saved. If after the last minimum was found (successive values of vofs were all higher) and the current shift is at least two steps outside that minimum all the way around, then that shift is accepted as the true local minimum. The corresponding du and dv of that minimum are now directly proportional to the velocity of this layer in x and y, that is, along the axis of the cameras and perpendicular thereto.

The process is repeated for all the layers.

A3.2.9 Report():

Report converts the values of dw[layer],du[layer],dv[layer], which were found in the preceding steps, to actual altitudes and directions, and displays them on the monitor for the user. The following computations are made.

Altitude in feet=k1*k2/(k3*dw[ln]) + k4 where

k1=focal length of the cameras in pixel widths =720,
k2=distance between the two cameras = 963.52 feet,
k3=number of pixels per point in the image = 2,
dw[ln]=z-offset of this layer.
k4 = the altitude ASL of the station = 240 feet.

Thus altitude of the layer is 346867.2/dw[ln] + 240.0 feet ASL, +/- 2 inches. The altitude error is actually at least +/- 1 pixel width at that altitude. If dw[ln]=94, say, then the reported cloud layer would be 3690+240=3930. But 94 is the integer reported for any dw greater

than 93.5 and less than 94.5, so that the layer lies somewhere between 3910 and 3950 feet--one percent. More to the point, the value of k1 is an approximation, arrived at as follows:

The width of the pixels in x is 22.2 micrometers. The focal length of the lens is 16 mm. Then k1=16/.0222=720. But focal length changes very slightly with iris opening and certainly with focus (the cameras are maintained at "infinity"), the pixel width is an engineering specification which undoubtedly has a small unknown error, and the cameras lines of sight are almost certainly not parallel nor perpendicular within some small error, for example the fact that the lines to the center of the earth from each are not parallel.

These errors accumulate randomly and could average out to zero, but to be safe, in the worst case, we suggest that measured altitude is true altitude within 5%. If the cloud base in the range of view is not level, the altitude produced is the average altitude, about that over the midpoint between the two cameras.

After altitude is found for each layer, velocity and direction are computed as follows:

dx=du*alt*k4/k5/k6*k7 where

du is the offset returned by Dxdy1() for this layer,
alt is the altitude of this layer in feet as found above
k4 is the number of pixels per unit of x offset =2
k5 is focal length in x-pixel widths = 720
k6 is the interval between the exposures, 2 seconds
k7 converts feet per second to knots=3600/6080
k4*k5/k6*k7=.0008223687, the constant found in the code

Next, dy is found in the same way, but with an adjustment for the fact that the y-dimension of a pixel is 1.17 times the x-dimension. This implies that unit velocity in y will cover less pixels than the same velocity in x, so that k4(y)= 2*1.17=2.34. Thus the constant found in the code is .000962775. The other coefficients are the same.

After dx and dy are found for a layer, Report() computes

Velocity = sqrt(dx 2+dy 2), knots.

First, if cloud movement is along the axes:

If dx=0 and dy=0 then direction is zero (and velocity is zero as well).
If dx=0 and dy)0 then direction is baseline direction.
If dx=0 and dy(0 then direction is 180 plus baseline direction.
If dy=0 and dx)0 then direction is 90 plus baseline direction.
If dy=0 and dx(0 then direction is 270 plus baseline direction.

Otherwise, angle = arctan(dy/dx)*degrees/radian, in degrees relative to the base line direction, 27.3037 true.

If dx<0 then direction = 270+baseline - angle,
If dx>0 then direction = 90 +baseline - angle.

After a header is printed giving the name, latitude and altitude of the station, the date and the time, column headings are printed as "cloud base ASL moving toward deg true, velocity kts, percent cover", and the altitude, direction, altitude and cover are printed, one row for each layer, from lowest to highest.

"Report()" returns to main, which spins until time for the next observation, at which point the entire program repeats.

A3.3 Annotated Code. Attached after this Annex

The operating system and its functions which run in interpretive mode (RCP) is given first, followed by listings of FEBLIB.C, the compiled code, called from RCP. After this, listings of the ancillary programs which were developed but not included in the operating code are given. No proprietary software (to my knowledge) is included except the WHSKY program itself, which is copyrighted by Micro Science, Inc.

A3.4 Miscellaneous software

Several programs not included in the operational WHSKY were written during the development process. These are discussed below and the code is attached following the operational system code (see above).

A3.4.1 Fourier noise filter, Filter():

This algorithm was devised by Aubanel and reported in Byte, 2/84. It has been translated to C and compiled, and appears to work exactly as specified. It is interesting to note that the routine returns the coefficients of the frequencies which are summed to produce the cleaned up function. If a time series of cloud pictures is made, the coefficient of the fundamental frequency displays a shift, and the frequency of this shift relative to the fundamental is in fact proportional to the velocity of the cloud.

A3.4.2 Lens compensation, Adjustlens():

If the lens were a theoretically perfect pinhole, the distance between any two points in the sky at the same altitude would be exactly proportional to the distance between them on the plane of the chip. If the points were projected onto a sphere rather than a plane, points away from the perpendicular would appear closer to the center. That is, if a picture of a perfect grid on a plane normal to the line of sight were taken with a standard wide angle lense, the grid lines at the periphery of the picture would appear closer together than those in the center, resulting in the "pin cushion" effect. Thus the camera "lies". The error is small, less than a pixel width for most of the field of a 16 mm lens, and because other errors are far greater, the compensatory routine for this error is available but not part of WHSKY.

A3.4.3 File processing routines:

In the development process, it is highly desirable to save intermediate results so that checked out routines do not have to be re-run to produce the input for a function which is under development. Hence it is common practice to store these intermediate results along the way. In the final version, which need save nothing, the routines have been deleted.

## Annex 4. Source code listings

```
  /*                          RCP\WHSKY.200                                */
  /*                Complete operating system 2/18/87                      */
  /*  Copyright (C) 1987 by Micro Science Inc.  ALL RIGHTS RESERVED*/
  /*        Development supported in part by US Air Force OSR              */
  /*                     Contract F49620-86C-0093DEF                       */
 #define npin 21600
 #define nl 6
 #define nsamples 3
   /* npin=21600=60*360 one fourth of a grabbed image */
   /* nt = number of grabbed images per observation*/
   /* nsamples, one more than required for dx,dy at a given obs'n.*/
   /* for a single cloud image (8 files), set nsamples=2 */
   /* arrays should be redimensioned to [2][120][135] */

 static float time,delaytime;
 static float time2,time3,time4,dt2=600,dt3=2,dt4=1200;
        /*next observation, next sample, intensity wait*/
        /*dt3 should be increased if altitude of layer is high */
 static char img0[npin];/*holds grabber part for copyimage*/
 static int zimg0[2][60][150],zimg1[2][60][150]; /*t, brd,row,col */
 static int vimg[2][60][150]; /*time,r,c(board0) for one layer */
 static double zofs0[150],zofs1[150], vofs[302],cover[nl];
     /*z offsets, at times 0,1,vofs at any time,% cover */
 static  int  dw[nl][2],zdw[nl][2],du[nl],dv[nl];/*by  lr,alt,vx,vy  */
report */

 main() {
    runc("SHELL SVPCIP -t 0xa0000"); /*initialize grabber */
    runc("LIBL FEBLIB");
    runc("LIBL RCP\MATHLIB\MATHLIB");
    runc("LIBL RCP\GRAPHLIB\GRAPHLIB");
    runc("SET BREAK  SCREEN(2,0,0,0);SCREEN(0,0,0,0);");
    runc("SET END  SCREEN(2,0,0,0);SCREEN(0,0,0,0);");
 int poweron=1,i=0;
 initxram(); /* read in XRAM.001, copy to 0xa0000, both boards, */
 outp(0x301,0xac); /*video on both boards*/
 printf("Adjust irises in cameras.  When ready, hit any key\n");
 getchar();
 while (poweron == 1) { /* forever */
   grabimage(); /*grab both images for t0 and t1*/
   gratozimg(); /* brings images into c-ram, re-pots into zimg0,1*/
   zimgout(); /* 2 arrays for showclouds--adj'd values of mid-sky */
   showclouds(img0,zimg0,zimg1); /* makes file to show all four
      zimgs on monitor */
    /* printf("end of showclouds.  Key\n"); getchar(); */
    /* zimgin(); /*recovers both arrays */
    /* printzimg(); */
   zolay0(zimg0,zofs0); /*overlays zimg0[0] onto zimg0[1] */
    /* zofs0out(); */ /*stores zofs0 produced by zolay0 */
```

```
    zolay1(zimg1,zofs1); /*overlays zimg1[0] onto zimg1[1] */
     /* zofs1out(); */ /*stores zofs1 produced by zolay1 */
     /* zofs0in(); */ /*recovers zofs0 for calcz */
     /* zofs1in(); */ /*recovers zofs1 for calcz */
     /* printzofs(); */
     /* screenzofs(); */
     calcz(); /*makes dw,zdw, the layers, from the zofs */
     dxdy1(); /* makes vimgs by layer and zimgs, and du,dv */
     /* printdws(); */
     report();
     delaytime = time2; spin();
    } /* while */
  } /*end of whsky */

/* ********************** subroutines ***************************** */

btimer() { /*produces the time in seconds, floating */
  char ac[10]; char *acpoint; int ret; float hr,min,sec;
  dostime(ac,4); ret=sscanf(ac,"%f:%f:%f",&hr,&min,&sec);
  time = 3600*hr + 60*min+sec;
  /* printf("Time in btimeris %.2f\n",time); getchar();*/
  return(); }

spin() { do btimer(); while (time < delaytime); }

initxram() {
  int i,blcnt; long a=0xa0000; char xram[num]; char *destin=xram;
  FILE *in1; if ((in1=fopen("RCP\XRAM.001","rb"))!=NULL)
    {blcnt=fread(destin,num,1,in1);fclose (in1);}
       else {printf("fileopen RCP\XRAM.001 failed");exit();}
    outp(0x301,0x24); /*ready boards (at source) to receive prams */
    runc("SET TRUST ON"); char *source=(char *)a;
    movmem (destin,source,num); /*copy prams to boards*/
    runc("SET TRUST OFF"); outp(0x301,0xac); /*return brds to digitg*/
   }

grabimage() { /*moves image memory into SVPCIP ram for gratozimg()*/
  int i=0, ck=-2,v=2; long addr,t16=65536L; unsigned a,b,c,d;
  outp(0x301,0xac); /*cameras on, both boards */
  btimer(); time2=time; /* converts dostime to a single floater */
  time3=time2; time2=time2 + dt2; addr=npin;a=0;b=0x54;c=0x60;d=0;
  for (i=0;i<nsamples;i++) { /*nsamples from cameras */
    printf("adr=%ld i=%d,a=%x,b=%x,c=%x\n",addr,i,a,b,c);
    grab(a,b,c); /*from FEBLIB.  Finds vert blanking time (1.6ms)*/
    addr+=npin; a=addr / t16; /*calc new xram memory address */
    d=addr - a*t16; b=d / 256; c=d - b * 256;
    time3+=dt3; delaytime=time3;spin(); } } /* end of grabimage() */

gratozimg() {
    /* for each t, for each grabber, copies 4 panels of npin=21600
       bytes from grabber to ram, then extracts middle alt bytes
       integerizes, copies to zimg(t)[b][60][150], replacing
       copyimage()+makezimg().  zimgout stores to disk.
       Cloud can be displayed on the monitor under SVPCIP using the
```

```
        "showcloud()" result.  Perform f r RCPZIMG3.000.  Note no \.  */

    FILE *out;
    char name[20];int a,b,t,i,im,j,jm,blcnt,board,quad,ww=0;
    unsigned delt=0;
    char *source, *destin=img0,*p1[7],*p4[6];
    *p1="rcp\img";*p4="002.00";
    long xram=0,window=0xa0000,t16=65536L,upper=window+t16;
    runc("SET TRUST ON"); outp(0x308,ww);
    for (board=0;board<2;board++) {
      if(board==0) outp(0x301,0x28); else outp(0x301,0x0c);
      window=0xa0000;ww=0;
      for (t=0;t<nsamples;t++) {
        printf("t q b window ww delt source destin sink xram\n");
        for (quad=0;quad<4;quad++) {
          printf("%2d %d %d %7ld ",t,quad,board,window);
          if (window + 21600L > upper ) {
            delt=upper - window;source=(char *)window;
            printf(" %2d %6u %7ld %7ld %7ld\n",
                    ww,delt,(long)source,(long)destin,xram);
            movmem(source, destin, delt); /*move the bottom part */
            ww++;outp(0x308,ww); xram+=npin; /*move the window up*/
            source=(char *)0xa0000; destin=(char *)(img0 + delt);
            printf(" %2d %6u %7ld %7ld %7ld\n",
                 ww, npin-delt,(long)source,(long)destin,xram);
            movmem(source,destin,21600-delt); /*move the top part*/
            window=0xa0000 + npin - delt; /*finish moving the window*/
            destin=img0; }
          else {source=(char *)window; movmem(source,destin,npin);
            window=window + 21600L; xram+=21600; printf(" %2d ",ww);
            printf(" %7ld %7ld %7ld\n",
                     (long)source,(long)destin,xram);}

/* one block of char raw data is now in destin=img0[21600].
   If q is 0 or 3 or t=0, continue (the data is not of interest),
   else write it to a file, copy alternate bytes to zimg0[2][60][150],
   zimg1[2][60][150]--t,b,r,c.  Then using zimgouts file the 2 zimgs */

          if (quad!=0 && quad!=3 && t>0) {
            copyzimg(img0,zimg0,zimg1,t-1,quad,board);
          } /*if*/
        } /*quad*/
      } /*t*/
    } /*board*/
    outp(0x301,0xac); /*video on both boards*/
    runc("SET TRUST OFF");
      /* zimg0 and zimg1 are now full, ready to be stored by zimgout*/  }
/* end of gratoimg() */


calcz() { /*from zofs0,zofs1, finds dw[ln][0],dw[ln][1] */
  int i,j,ka[2],k,l,dt[n1+1][n1+1]; /* first 5-point smooth*/
  vofs[2]=(zofs0[0]+zofs0[1]+zofs0[2]+zofs0[3]+zofs0[4])*0.2L;
  for (j=3;j<148;j++)
```

```
    vofs[j]=(5*vofs[j-1]-zofs0[j-3]+zofs0[j+2])*0.2L;
  for (j=2;j(148;j++) zofs0[j]=vofs[j];
  /* printzofs();*/
  zofs0[1]=zofs0[2]-1;zofs0[0]=zofs0[1]-1;
  vofs[2]=(zofs1[0]+zofs1[1]+zofs1[2]+zofs1[3]+zofs1[4])*0.2L;
  for (j=3;j(148;j++)
    vofs[j]=(5*vofs[j-1]-zofs1[j-3]+zofs1[j+2])*0.2L;
  for (j=2;j(148;j++) zofs1[j]=vofs[j];
  zofs1[1]=zofs1[2]-1;zofs1[0]=zofs1[1]-1;
  /* screenzofs();*/
  for (j=1,ka[0]=0,ka[1]=0;j(146;j++) {
    if (zofs0[j-1])=zofs0[j] && zofs0[j+1])zofs0[j])
       {dw[ka[0]][0]=j;zdw[ka[0]][0]=(int)zofs0[j]; ka[0]++;
        if (ka[0])=nl) {pushw(0);ka[0]=nl-1;} }
    if (zofs1[j-1])=zofs1[j] && zofs1[j+1])zofs1[j])
       { dw[ka[1]][1]=j;zdw[ka[1]][1]=(int)zofs1[j]; ka[1]++;
        if (ka[1])=nl) {pushw(1);ka[1]=nl-1;} } }
  /* clean up dw/zdw stacks */
  for (j=0;j(nl;j++) printf("zdw,dw: %5d %5d %5d %5d\n",
                        dw[j][0],dw[j][1],zdw[j][0],zdw[j][1]);
  for (i=0;i(nl;i++) { dt[i][nl]=0;dt[nl][i]=0; /*initialize flags*/
    for (j=0;j(nl;j++) {dt[i][j]=dw[i][0]-dw[j][1];
      if (dt[i][j](0) dt[i][j]=-dt[i][j];
      printf("%3d",dt[i][j]); } printf("\n"); } /*fill it in*/
  for (k=0;k(3;k++) { for (i=0;i(nl;i++) { for (j=0;j(nl;j++) {
        if (dt[nl][j]!=0 || dt[i][nl]!=0) continue;
        if (dt[i][j]==k) {dt[i][nl]=1; dt[nl][j]=1;} } } }
  for (i=0,k=0;i(nl;i++) { /* push up the selected items */
    if (dt[i][nl])0) {dw[k][0]=dw[i][0];zdw[k][0]=zdw[i][0];k++;} }
  for (i=k;i(nl;i++) {dw[i][0]=0;zdw[i][0]=0;} /* end fill*/
  for (i=0,k=0;i(nl;i++) { /* push up the selected items */
    if (dt[nl][i])0) {dw[k][1]=dw[i][1];zdw[k][1]=zdw[i][1];k++;} }
  for (i=k;i(nl;i++) {dw[i][1]=0;zdw[i][1]=0;} /* end fill*/
  for (j=0;j(nl;j++) printf(" %5d %5d %5d %5d key\n",
      dw[j][0],dw[j][1],zdw[j][0],zdw[j][1]);
  /*getchar();*/ }

pushw(b) int b; { int i,imx=0,mx=zdw[0][b];
   for (i=1;i(nl;i++) {if (zdw[i][b])mx) {mx=zdw[i][b];imx=i;}}
   if (imx(nl-1) { for (i=imx;i(nl-1;i++)
    {zdw[i][b]=zdw[i+1][b]; dw[i][b]=dw[i+1][b];} } }


dxdy1() /*v.  2/18/87 */
/* dithering routine--moves center of vimg0 around center of vimg1*/
{
 int dul=0,dvl=0,mdu=0,mdv=0,t,n,l;
 int i1s,i2s,iend,j1s,j2s,jend;
 int ofs0,ofs1,jrng,f1,f2;
 double vof,minvofs;
 for (l=nl-1;l)=0;l--) /*take a layer*/
   if (zdw[l][0])0 && dw[l][0] (140 && zdw[l][1])0 && dw[l][1](140)
   { ofs0=dw[l][0];ofs1=dw[l][1];f1=0;f2=0;dul=0;dvl=0;mdu=0;mdv=0;
```

```
      makevimg(ofs0,ofs1,1,zimg0,zimg1,vimg,cover);

      if (ofs0)ofs1) jrng=150-ofs0; else jrng=150-ofs1;
      minvofs=dvolay(vimg,0,0,60,0,0,jrng);
      printf("minvofs=%lf dul=%3d dvl=%3d jrng=%3d cover=%lf\n",
        minvofs,dul,dvl,jrng,cover[1]);
      for (t=1;t(30;t+=2) {
        for (n=1;n(=t;n++) { dul++;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf \n",
                t,n,dul,dvl,vof); } /*n*/
        for (n=1;n(=t;n++) { dvl--;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s.i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                t,n,dul,dvl,vof);} /* n */
        if (mdv)dvl+1 && mdu(dul-1) f1=1; else f2=0;
        printf("mdu=%3d mdv=%3d f1=%2d f2=%2d\n",mdu,mdv,f1,f2);
        if (f1==1 && f2==1) break;
        for (n=1;n(=t+1;n++) { dul--;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                t,n,dul,dvl,vof);
        } /* n */
        for (n=1;n(=t+1;n++) { dvl++;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                t,n,dul,dvl,vof);
        } /*n*/
      if (mdv(dvl-1 && mdu)dul+1) f2=1; else f1=0;
        printf("mdu=%3d mdv=%3d f1=%2d f2=%2d\n",mdu,mdv,f1,f2);
      if (f1==1 && f2==1) break;
      } /*t*/
    du[l]=mdu;dv[l]=mdv;
    } /* layer */
```

```
      makevimg(ofs0,ofs1,l,zimg0,zimg1,vimg,cover);

      if (ofs0)ofs1) jrng=150-ofs0; else jrng=150-ofs1;
      minvofs=dvolay(vimg,0,0,60,0,0,jrng);
      printf("minvofs=%lf dul=%3d dvl=%3d jrng=%3d cover=%lf\n",
        minvofs,dul,dvl,jrng,cover[l]);
      for (t=1;t(30;t+=2) {
        for (n=1;n(=t;n++) { dul++;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
           {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf \n",
                  t,n,dul,dvl,vof); } /*n*/
        for (n=1;n(=t;n++) { dvl--;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                  t,n,dul,dvl,vof);} /* n */
        if (mdv)dvl+1 && mdu(dul-1) f1=1; else f2=0;
        printf("mdu=%3d mdv=%3d f1=%2d f2=%2d\n",mdu,mdv,f1,f2);
        if (f1==1 && f2==1) break;
        for (n=1;n(=t+1;n++) { dul--;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                  t,n,dul,dvl,vof);
        } /* n */
        for (n=1;n(=t+1;n++) { dvl++;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                  t,n,dul,dvl,vof);
        } /*n*/
      if (mdv(dvl-1 && mdu)dul+1) f2=1; else f1=0;
        printf("mdu=%3d mdv=%3d f1=%2d f2=%2d\n",mdu,mdv,f1,f2);
      if (f1==1 && f2==1) break;
      } /*t*/
    du[l]=mdu;dv[l]=mdv;
    } /* layer */
```

```
      makevimg(ofs0,ofs1,1,zimg0,zimg1,vimg,cover);

      if (ofs0)ofs1) jrng=150-ofs0; else jrng=150-ofs1;
      minvofs=dvolay(vimg,0,0,60,0,0,jrng);
      printf("minvofs=%lf dul=%3d dvl=%3d jrng=%3d cover=%lf\n",
        minvofs,dul,dvl,jrng,cover[1]);
      for (t=1;t(30;t+=2) {
        for (n=1;n(=t;n++) { dul++;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf \n",
                t,n,dul,dvl,vof); } /*n*/
        for (n=1;n(=t;n++) { dvl--;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                t,n,dul,dvl,vof);} /* n */
        if (mdv)dvl+1 && mdu(dul-1) f1=1; else f2=0;
        printf("mdu=%3d mdv=%3d f1=%2d f2=%2d\n",mdu,mdv,f1,f2);
        if (f1==1 && f2==1) break;
        for (n=1;n(=t+1;n++) { dul--;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                t,n,dul,dvl,vof);
        } /* n */
        for (n=1;n(=t+1;n++) { dvl++;
          if (dvl)0) {i1s=0;i2s=dvl;iend=60-dvl; } }
            else {i1s=-dvl;i2s=0;iend=60;}
          if (dul)0) {j1s=0;j2s=dul;jend=jrng-dul;} ng;}
            else {j1s=-dul;j2s=0;jend=jrng;}
          if((vof=dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend))(minvofs)
            {minvofs=vof;mdu=dul;mdv=dvl;}
          printf("t=%3d n=%3d dul=%3d dvl=%3d vof=%lf\n ",
                t,n,dul,dvl,vof);
        } /*n*/
      if (mdv(dvl-1 && mdu)dul+1) f2=1; else f1=0;
        printf("mdu=%3d mdv=%3d f1=%2d f2=%2d\n",mdu,mdv,f1,f2);
      if (f1==1 && f2==1) break;
      } /*t*/
    du[1]=mdu;dv[1]=mdv;
    } /* layer */
```

```
    blcnt=fread((char *)zofs0,1200,1,in);
    fclose(in);}
    else printf("fileopen rcp\zofs03 failed\n");
 return;}

 zofs1in()
  {/* recovers offsets stored by zofs1out()in file 'rcp\zofs13' */
  int blcnt;
  FILE *in;
  if((in=fopen("rcp\zofs13","rb"))!=NULL) {
    blcnt=fread((char *)zofs1,1200,1,in);
    fclose(in);}
    else printf("fileopen rcp\zofs13 failed\n");
 return;}

 printzofs()
 { int i;
  printf("zofs t=0\n");
  for(i=0;i(150;i++) printf("%3d %9lf ",i,zofs0[i]);
  printf("zofs t=1\n");
  for(i=0;i(150;i++) printf("%3d %9lf ",i,zofs1[i]);
 return;}

 screenzofs()
 { int i,j=0;
  printf("in screenzofs:key\n");getchar();
  INITBC();SCREEN(1,1,0,0);
  for (i=0;i(150;i++) PSET(2*i,150-(int)(zofs0[i]),1);
  for (i=0;i(150;i++) PSET(2*i,200-(int)(zofs1[i]),2);
  printf("*");getchar();SCREEN(2,0,0,0);SCREEN(0,0,0,0);
 }

 printdws() {
  int i;
  for (i=0;i(nl;i++)
    printf("%3d dw0 %3d dw1 %3d zdw0 %3d zdw1 %3d du %3d dv %3d \n",
            i,dw[i][0],dw[i][1],zdw[i][0],zdw[i][1],du[i],dv[i]); }

 printvimg(jrng) int jrng; { /* int vimg[t][i][j] */
  int i,j,t1,t2;
  printf("\n vimg row totals for jrng=%3d\n",jrng);
  for (i=0;i(60;i++) {
    for (j=0,t1=0,t2=0;j(10;j++)
      {t1+=vimg[0][i][j];t2+=vimg[1][i][j]; }
    printf("\n%6d,t1,%6d,t2\n",t1,t2);} }

/* ********************************************************************** */
```

Compiled functions

FEBLIB.C

```
#include "stdio.h"
#include "math.h"
```

```
void grab(a,b,c) unsigned a,b,c; {
   outp(0x301,0xac);
   while ((inp(0x300) & 2)==2) continue;
   while ((inp(0x300) & 2)==0) continue;
   outp(0x30f,a); outp(0x30e,b); outp(0x30d,c);
   while ((inp(0x300) & 2)==2) continue;
   while ((inp(0x300) & 2)==0) continue;
   outp(0x301,0x2c);
   return; }

void copyzimg(img0,zimg0,zimg1,t,q,b)
     /*copy img0 to zimg|t=0,1|q=[2] i=[60] j=[150]*/
 char *img0;
 int zimg0[2][60][150],zimg1[2][60][150];
 int t,q,b;
 { int i,im,j,jm,a;
   for (i=(q-1)*30,im=0;i(q*30;i++,im+=720) {
     for (j=0,jm=30+im;j(150;j++,jm+=2) { a=(int)img0[jm] & 255;
       if(t==0) zimg0[b][i][j]=a; else zimg1[b][i][j]=a;
     } /*j*/
   } /*i*/
return;}

void zolay0(zimg0,zofs0) int zimg0[2][60][150]; double zofs0[150];
{ int ofs,i,j;
  double sumsq,meena=0,meenb=0,dmeen,del;
  for (i=0;i(60;i++) {
    for (j=0;j(150;j++) {
      meena+=(double)zimg0[0][i][j];meenb+=(double)zimg0[1][i][j];}
  } dmeen=meena/meenb;
  printf("meena,meenb,dmeen %9.1f %9.1f %9.21f\n",meena,meenb,dmeen);
  for (i=0;i(60;i++) { for (j=0;j(150;j++) zimg0[1][i][j]*=dmeen;}
  for (ofs=0;ofs(150;ofs++)
  { for (j=0,sumsq=0;j((150-ofs);j++)
    { for (i=0;i(60;i++)
      { del=zimg0[0][i][j]-zimg0[1][i][j+ofs]; sumsq+=del*del;} }
  zofs0[ofs]=sqrt(sumsq/(150-ofs));
  printf("zofs0[%3d]=%91f sumsq=%9.1f \n ",ofs,zofs0[ofs],sumsq);
  }
return; }

void zolay1(zimg1,zofs1) int zimg1[2][60][150]; double zofs1[150];
{ int ofs,i,j;
  double sumsq,meena=0,meenb=0,dmeen,del;
  for (i=0;i(60;i++) {
    for (j=0;j(150;j++) {
      meena+=(double)zimg1[0][i][j];meenb+=(double)zimg1[1][i][j];}
  } dmeen=meena/meenb;
  printf("meena,meenb,dmeen %9.1f %9.1f %9.21f\n",meena,meenb,dmeen);
  for (i=0;i(60;i++) { for (j=0;j(150;j++) zimg1[1][i][j]*=dmeen;}
  for (ofs=0;ofs(150;ofs++)
  { for (j=0,sumsq=0;j((150-ofs);j++)
    { for (i=0;i(60;i++)
      { del=zimg1[0][i][j]-zimg1[1][i][j+ofs]; sumsq+=del*del;} }
```

```
   zofs1[ofs]=sqrt(sumsq/(150-ofs));
   printf("zofs1[%3d]=%91f sumsq=%9.21f\n ",ofs,zofs1[ofs],sumsq);
   } return; }

 void makevimg(ofs0,ofs1,l,zimg0,zimg1,vimg,cover)
   int ofs0,ofs1,l;
   int zimg0[2][60][150],zimg1[2][60][150],vimg[2][60][150];
   double cover[10];
 { int i=0,j=0,zerolev,tm1,tm2;
   double sigma,sumsq=0,del,den,meana=0,meanb=0,dmean;
   for (i=0;i(60;i++) {
     for (j=0;j(150-ofs0;j++)
       {meana+=zimg0[0][i][j];meanb+=zimg0[1][i][j+ofs0];} }
   dmean=meana/meanb;
   printf("%9.21f %9.21f %9.21f\n",meana,meanb,dmean);
   den=1.0/(60*(150-ofs0));
   zerolev= meana*den;
   printf("den=%lf zero-level=%5d\n",den,zerolev);
     /* calculates and sets zero level for points not in this layer.*/
   for (i=0;i(60;i++) {
     for (j=150-ofs0;j(150;j++) vimg[0][i][j]=zerolev;
     for (j=0;j(150-ofs0;j++) {
       del=zimg0[0][i][j]-(int)(zimg0[1][i][j+ofs0]*dmean);
     sumsq+=del*del;} }
   sigma=sqrt(sumsq*den);
   printf("ofs=%3d sumsq=%9.01f,sigma=%9.21f\n",ofs0,sumsq,sigma);
   for (i=0;i(60;i++) {
     for (j=0;j(150-ofs0;j++) {
       del=zimg0[0][i][j]-(int)(zimg0[1][i][j+ofs0]*dmean);
       if ((del=fabs(del))(sigma) {vimg[0][i][j]=zimg0[0][i][j];
        cover[1]++;}
       else vimg[0][i][j]=zerolev;
   } }
   cover[1]=cover[1]*den;
   sumsq=0;meana=0;meanb=0;
   for (i=0;i(60;i++) {
    for (j=0;j(150-ofs1;j++)
      {meana+=zimg1[0][i][j];meanb+=zimg1[1][i][j+ofs1];} }
   dmean=meana/meanb;
   printf("%9.21f %9.21f %9.21f\n",meana,meanb,dmean);
   den=1.0/(60*(150-ofs1));
   zerolev=meana*den; printf("den=%lf zero-level=%5d\n",den,zerolev);
     /*calculates and sets zero level for points not in this layer.*/
   for (i=0;i(60;i++) {
    for (j=150-ofs1;j(150;j++) vimg[1][i][j]=zerolev;
    for (j=0;j(150-ofs1;j++) {
      del=zimg1[0][i][j]-(int)(zimg1[1][i][j+ofs1]*dmean);
    sumsq+=del*del;} }
   sigma=sqrt(sumsq*den);
   printf("ofs=%3d sumsq=%9.01f,sigma=%9.21f\n",ofs1,sumsq,sigma);
   for (i=0;i(60;i++) {
    for (j=0;j(150-ofs1;j++) {
      tm1=zimg1[0][i][j];tm2=zimg1[1][i][j+ofs1];
      del=tm1-(int)(tm2*dmean);
```

```
      /*printf("ij=%3d %3d zimg1[0]=%5d zimg1[1]=%5d",i,j,tm1,tm2);
        printf("tm2*dmean=%lf (int)( .  )=%5d, del=%lf",
              tm2*dmean,(int)(tm2*dmean),del); */
      if ((del=fabs(del))(sigma)
          vimg[1][i][j]=zimg1[0][i][j];
      else vimg[1][i][j]=zerolev;
      /*printf("%4.2lf %3d %3d ",del,zimg1[0][i][j],vimg[1][i][j]);*/
  } } return;}

double dvolay(vimg,i1s,i2s,iend,j1s,j2s,jend)
    int vimg[2][60][150];
    int i1s,i2s,iend,j1s,j2s,jend;
{ double smsq=0;
  int i1,i2,j1,j2,k=0,del;
  for (i1=i1s,i2=i2s;i1(iend;i1++,i2++) {
   for (j1=j1s,j2=j2s;j1(jend;j1++,j2++) {
     del=vimg[0][i1][j1]-vimg[1][i2][j2]; smsq+=del*del;k++; }
  }
  smsq=smsq/k; return(smsq);
}


void showclouds(img0,zimg0,zimg1)
 char *img0;
 int zimg0[2][60][150],zimg1[2][60][150];
{
         /* converts zimgt[b][r][c] to chars and writes to disk*/
         /* then waits for svpcip command to display the file */
 int q,i,im,j,k,blcnt;
 FILE *out, *fopen();
 if ((out=fopen("rcp\zimg3.000","wb"))==NULL) {
   printf("fileopen rcp\zimg3.000 failed:key\n");getchar();exit();}
 for (k=0;k(21600;k++) img0[k]=0;
 for (q=0;q(2;q++) {
   for (i=q*30,im=i*360;i((q+1)*30;i++,im+=360)
     {/*1st q,both b's,t=1*/
     for (j=0,k=im;j(150;j++,k++) {
       img0[k+30]=(char)zimg0[0][i][j];
       img0[k+180]=(char)zimg0[1][i][j];}}}
 blcnt=fwrite((char *)img0,21600,1,out); /* q1,2; t1; b1,2 */
 for (q=0;q(2;q++) {
   for (i=q*30,im=i*360;i((q+1)*30;i++,im+=360) {
     for (j=0,k=im;j(150;j++,k++) {
       img0[k+30]=(char)zimg1[0][i][j];
       img0[k+180]=(char)zimg1[1][i][j];}}}
 blcnt=fwrite((char *)img0,21600,1,out); /* q1,2; t2; b1,2 */
 fclose(out);
 return;}
```

LINKER TO FEBLIB.C

This library,FEBLIB, contains functions for WHSKY.200

```
30=program size
2=memory needed
void grab(unsigned,unsigned,unsigned)
  /*sets address for grabber ram */
void copyzimg(*,*,*,int,int,int)
  /*copies img0 into zimg0,1 to be filed*/
void zolay0(*,*)
  /*olays zimg0[0ij] onto zimg0[1ij], returns zofs0*/
void zolay1(*,*)
  /*olays zimg1[0ij] onto zimg1[1ij], returns zofs1*/
void makevimg(int,int,int,*,*,*,*)
  /*copies 1 lr of zimg0,1 into vimg[t]*/
double dvolay(*,int,int,int,int,int,int)
  /*olays vimg[t=0] onto vimg[t=1]*/
void showclouds(*,*,*)
  /*makes a file,rcpzimg3.000, to put on monitor*/
```

Annex A4.2 MYLIB.C

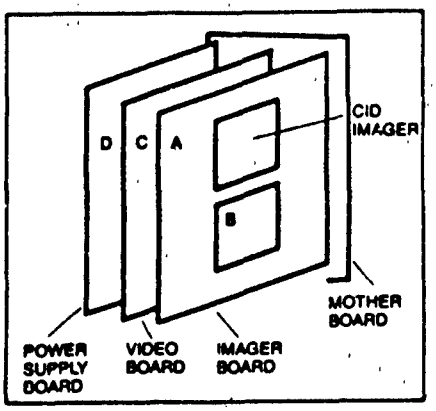
```
#include "stdio.h"
#include "math.h"

void filter(xa,x1a,nfft,e,r,vin) /* by Aubanel: Byte 2.84 */
  double *xa,*x1a; int nfft,e; double *r,*vin; {
  /* enter with xa,ya filled from imgx with size and no.  of freqs e*/
  /* fill fft series into r and vin */
   int j,j1,j2,jp1,jp,n=nfft,k,k1,l;
   int q=0,d=n/10,esq=e*e;
   double pi=3.141592654,twopi=2*pi;
   double twopiovn=twopi/n,s1=0,s2=0,x1,x2,dm,b;
   double g=0,s,c,ul,f1,f2,t,lg2,f;
   double u[92],v[92];
   lg2=log(2.0);
   for (j=0;j(d;j++) {s1+=xa[j]; s2+=xa[n-j-1];}
        /* map to straight line */
   x1=s1/d;x2=s2/d;dm=(x2-x1)/(n-d);b=(x1+x2-dm/n)/2;
   for (j=0;j(n;j++) {xa[j]+=-dm*j-b;g+=xa[j];}
   r[0]=g/n;q=1;
   j2=(n-1)/2; jp1=log((double)(2*j2-1))/lg2; /*r(k) transform */
   for (k=q;k(e;k++) { j1=j2;s=twopiovn*k;c=cos(s);s=sin(s);
     for (j=1;j(=j1;j++) {l=2*j-1;u[j]=xa[l]*c+xa[l+1];v[j]=xa[l]*s;}
     s=2*s*c;c=2*c*c-1;
     for (jp=1;jp(=jp1;jp++) {u[j1+1]=0;v[j1+1]=0;j1=(j1+1)/2;
       for (j=1;j(=j1;j++) {l=2*j-1; ul=u[l]*c-v[l]*s+u[l+1];
         v[j]=u[l]*s+v[l]*c+v[l+1]; u[j]=ul;} /*j*/
       s=2*s*c;c=2*c*c-1;} /*jp*/
     r[k]=(xa[0]+u[1]*c+v[1]*s)/n; } /*k*/
   for (k=q;k(e;k++) { /* vin(k) transform */
     j1=j2;s=twopiovn*k;c=cos(s);s=sin(s);
     for (j=1;j(=j1;j++) {l=2*j-1;u[j]=-xa[l]*s;v[j]=xa[l]*c+xa[l+1];}
     s=2*s*c;c=2*c*c-1;
     for (jp=1;jp(=jp1;jp++) {u[j1+1]=0;v[j1+1]=0;j1=(j1+1)/2;
       for (j=1;j(=j1;j++) {l=2*j-1;ul=u[l]*c-v[l]*s+u[l+1];
```

```
          v[j]=u[l]*s+v[l]*c+v[l+1];u[j]=ul; } /*j*/
        s=2*s*c;c=2*c*c-1;} /*jp*/
        vin[k]=-(u[1]*c+v[1]*s)/n; } /*k*/
    if (e)q) q=e; /*inverse transform */
    for (k=1,f1=0,f2=0;k(e;k++) { t=r[k]; f1+=t; f2+=k*k*t;}
    x1a[0]=r[0]+2*(f1-f2/esq)+b;
    jp1=log((double)(2*e-3))/lg2;
    for (j=1;j(n;j++) {
      for (k=1;k(e;k++) { f=1-k*k/esq; u[k]=r[k]*f; v[k]=-vin[k]*f;}
      k1=e-1;s=twopiovn*j;c=cos(s);s=sin(s);
      for (jp=1;jp(=jp1;jp++) {u[k1+1]=0;v[k1+1]=0;k1=(k1+1)/2;
        for (k=1;k(=k1;k++) {l=2*k-1;ul=u[l]*c-v[l]*s+u[l+1];
          v[k]=u[l]*s+v[l]*c+v[l+1];u[k]=ul; }
        s=2*s*c;c=2*c*c-1;} /*jp*/
      x1a[j]=r[0]+2*(u[1]*c+v[1]*s)+dm*j+b; } /*j*/
  return; }
```

# GENERAL INFORMATION

The TN2505 camera series uses LSI/VLSI solid state construction. Advances in this technology have produced a camera that has low power dissipation, and whose performance will not degrade over time. Solid state circuitry has eliminated bulky camera components enabling the camera to be light in weight and compact in size. These cameras are ideal for security, military, automation, medical and scientific applications.

## BOARD CONFIGURATION




**A. IMAGER BOARD** — Contains the CID (Charge Injection Device), LSI scan generator and a hybrid video pre-amplifier.

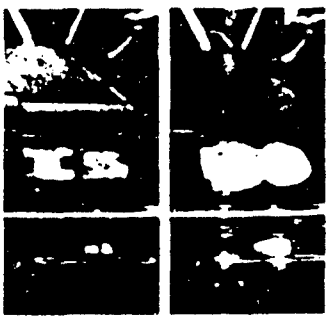
**B. SCAN GENERATOR** — Located on the Imager board, this CMOS scan generator provides the clock signals required by the CID imager, analog video processing circuits and the sync and blanking pulses.

**C. VIDEO BOARD** — The video amplifier on this board receives video from the hybrid pre-amplifier (of the imager board) and provides the following circuit functions: automatic black level, gain, and the addition of sync and blanking.

**D. POWER SUPPLY BOARD** — This board contains the DC power circuit and modular sync component system. Power is supplied to the camera by a separate power supply module. A 10' power supply cable is standard, although up to 50' of cable may be used without affecting quality. This camera can also be operated from other DC power sources with output of +12 to +33v DC @ 1/2 amp peak minimum.

## PERFORMANCE FEATURES

**OPTICAL VERSATILITY** — A Standard C mount permits the use of a wide variety of CCTV lenses, including telephoto, zoom, wide angle close-up and pin hole. The exact application would determine which lens to use. An Automatic Lens Connector provides video and power for an auto iris lens system.



**SENSOR QUALITY** —CID sensors exhibit virtually no lag or blooming. Individual pixel images exhibit minimal blooming into adjacent pixels. There is no post field lag, or build up lag, and bias light compensation is not required.

## I/O CONTROL

**GAIN OPTIONS** — Internal Video gain of +6db or +12db may be implemented via the I/O control. When the camera is not being used with an auto iris lens, an optional external gain switch can be added.

**INJECT INHIBIT** — Inject inhibit allows interruption of camera readout upon command, synchronous stop motion of high speed events, and integration of static low light imagery. When Inject Inhibit is removed, a single field of information may be stored externally for further processing or monitor display.

**TIMING OUTPUTS** — Camera timing signals are available at the I/O connector and may be used to synchronize the operation of the external equipment. The sensor element rate clock signal is provided along with H & V drive signals. However, these signals are not buffered and require care in interfacing.

# TECHNICAL INFORMATION

## SENSOR ARCHITECTURE

The TN 2505 series imagers consist of a matrix of photo sensitive pixels arranged in columns and rows. The readout is controlled by shift registers (or decoders) located at the periphery of the imager. Photon radiation imaged on the sensor surface penetrates the sensor surface where it is collected by the charge storage areas of each pixel. (Fig 1)

The column storage area of each pixel, as well as the interconnection of each column's storage areas for pixels in that column is ordered by a single strip of high conductivity polysilicon, which vertically crosses the thin oxide regions of all pixels in a column of the CID matrix. The row storage area is similarly ordered by a horizontal crossing of the CID matrix. The column and row storage areas are dielectrically separated.

A thin aluminum strip, in direct contact with the upper polysilicon, is designed to improve imager noise performance and row access speed.

The charge collected under a specific pixel's electrode area can be transferred or injected between column and row of the pixel storage area. This function results in an imager with no spacing between pixels and minimum obscuration to incident illumination.

Additionally, the array structure is "tuned"

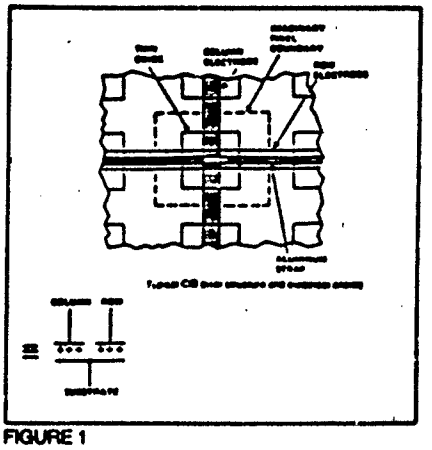


FIGURE 1

to minimize multiple reflection interference. Hence, high quantum efficiencies (45–55%) and responsiveness (approximately 0.2 amps/watt) are obtained.

Figure 2 shows the incorporation of a matrix of four rows and four columns (16 pixels total). These form a differential row read CID imager. Specific columns and rows are addressed through column and row multiplexers by logic "1" output from the

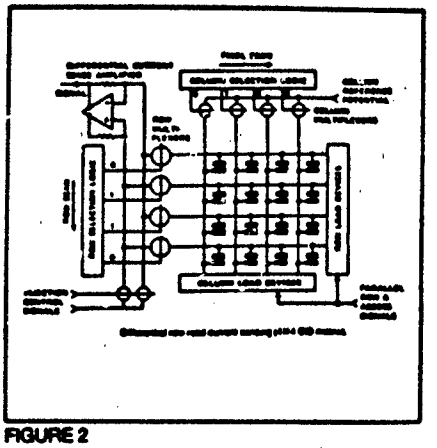
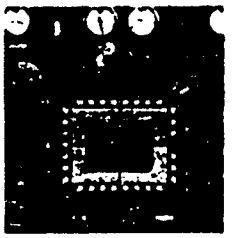


FIGURE 2

column and row selection logic. Simultaneous selection of a column and row causes signal charge collected under the column electrode to be transferred to the row electrode. There, it is sensed and amplified by a high gain, low noise preamplifier. Imager KTC noise is effectively eliminated upon row selection by the current mode amplifier. Column induced fixed pattern noise is removed at the imager output.

**The TN 2505 camera is designed for RS 170 output and the TN 2506 for CCIR standards.**

| | TN 2505 (CID-17) | TN 2506 (CID-18) |
|---|---|---|
| Pixel Array | 248V x 388H | 294V x 416H |
| Active Elements | 242V x 377H | 288V x 400H |
| Pixel Size | 27.3μm x 23.3μm | 22μm x 23μm |



TN2505A, TN2505B

## SYSTEM INTEGRATION

Video switching or image processing requires a high degree of signal synchronization. In order to meet these requirements the TN2505 & TN2506 model versions, A–1 through A–6, are offered with a variety of sync options to meet virtually any situation.

**A-1 LINE LOCK SYNCHRONIZATION** Line Lock Sync locks the vertical blanking interval of the camera to the 60 (or 50) cycle, power line. When several cameras are used Line Lock Sync eliminates vertical roll when switching from one camera to another. Unless specified otherwise, both the TN2505 & TN2506 come with Line Lock Sync.

**A-2 CRYSTAL CONTROLLED SYNCHRONIZATION** Crystal Controlled Sync provides optimum sync stability of an individual camera operating independently. The clock crystals supplied for these cameras are TN2505A 2–14. 318 MHz, TN2506A 2–15. 375 MHz

**A-3 GENLOCK SYNCHRONIZATION** Genlock Synchronization permits cameras to be locked to each other or locked to a separate source of RS-170 or CCIR video. These models have an easy access Phase Trim Control which brings the Sync Master camera and the other cameras connected to the master into exact phase relationship.

**A-4 H & V DRIVE LOCK SYNCHRONIZATION** This version permits one or more cameras to be locked to a negative going H & V drive of an RS-170 or CCIR video source.

**A-5 COMPULOCK™ SYNCHRONIZATION** The Compulock Sync permits one or more cameras to be locked to an external source of progressively scanned (60/50 frames per second) video such as a CRT controller.

**A-6 COMPULOCK™ SYNCHRONIZATION** This version permits one or more cameras to be locked with H & V drive outputs of a progressively scanned video source. Both the A–5 and the A–6 versions are scanned at 60 frames per second. While video can be displayed on a standard monitor, the picture will not be interlaced. The monitor will lock into a field at random.

# SPECIFICATIONS

## ELECTRICAL

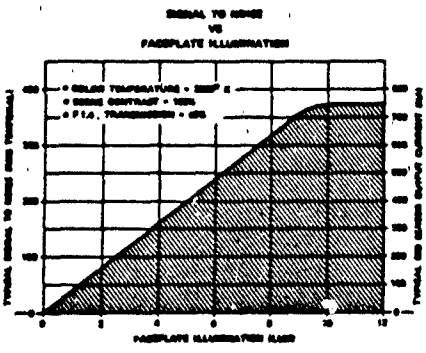
| | |
|---|---|
| Input Voltage | |
| Power Supply Module | 120VAC ± 15% (TN 2505) 50/60Hz<br>220VAC ± 10% (TN 2506) 50Hz |
| Camera | + 12 to + 35 VDC |
| Input Power | 2.5 Watts |
| Sensor Format | 244 Vertical, 388 Horizontal (TN2505)<br>290 Vertical, 416 Horizontal (TN2506) (CCIR) |
| Image Format | 11 Millimeters Diagonal |
| Composite Video | 1 Volt p-p into 75 Ohms RS-170 |
| Sync | Internal RS-170, Linelock Crystal, External H&V, Genlock or Compulock |
| S/N Ratio | 50 dB at Saturation |
| Geometric Distortion | 0% |
| Spectral Response | 350 to 1100 Nanometers |
| ALC Range | 2.6 Million to 1 With ft. 8-360 Auto-Iris Lens |
| Gray Scale | 10 shades of gray, Minimum EIA Standard Chart |
| Modulation Transfer Function | Better than 80% at 250 lines |
| Scan Rate | 525/60 TN 2505<br>625/50 TN 2506 |
| Sensitivity | Full output (1 Volt composite) at .8fc faceplate (0dB gain)<br>.4fc faceplate (+ 6dB gain)<br>Useable video at .2fc faceplate (0dB gain) |
| Outputs | |
| I/O Conn. | Clock (14.3MHz)TN 2505, H&V Drive (Non-buffered) |
| Iris | Gnd, Video, + 10 VDC |
| Video | Composite, 1 Volt p-p RS-170 |
| Inputs | |
| I/O Conn. | + 6 & + 12dB Gain, Sync, Injection Inhibit. |
| Power | + 12 to + 35 VDC |

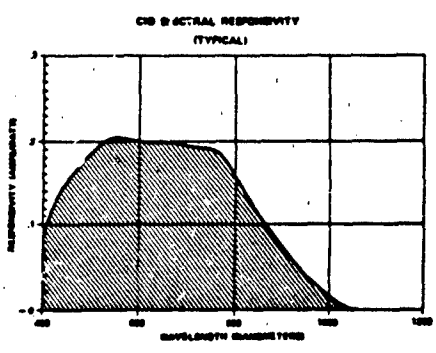
## ENVIRONMENTAL

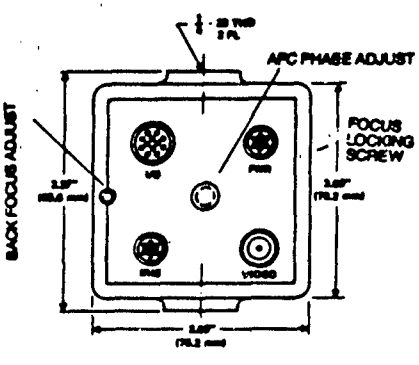
| | |
|---|---|
| Temp. Range | 0 to 50 C. |
| Humidity | 95% Non-condensing |
| Shock | 50G (½ sinewave at 10ms duration) |
| Vibration | 6 G's at 500Hz. For applications requiring greater loading we recommend using the two piece camera configuration. |

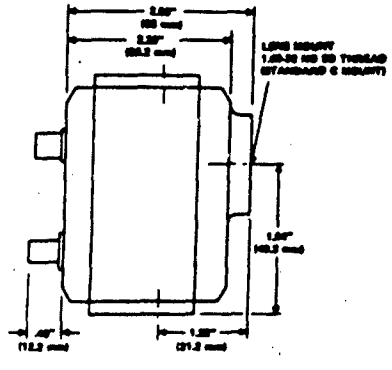
## MECHANICAL

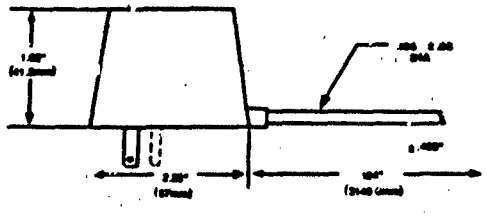
| | |
|---|---|
| Weight: | 13.5oz |
| Camera Dimensions | (3.37" x 3.0" x 3.04") |
| Lens Mount: | Standard C Mount 1.00–32 Thread |
| Camera Mount: (top and bottom) | ¼–20 Thread |
| Video Connectors | BNC |









For further information contact:
Marketing Manager, Electronic Camera Operation
GENERAL ELECTRIC COMPANY
890 7th North Street
Liverpool, New York 13088
(315) 456-2808/2945

We bring good things to life.

GENERAL ELECTRIC

PRINTED IN USA

## I. GENERAL INFORMATION

### Introduction

The TN2505, 6, 7 offers features and performance never before available in a surveillance product. The camera's small size opens a whole realm of exciting new application and installation possibilities. Its low power and wide input voltage range further extend its flexibility. Use of a solid state imager, GE's CID (Charge Injection Device), eliminates the performance drift and ultimate need for replacement experienced with vacuum tube imaging devices. Utilization of Large Scale Integrated circuits (LSI's) makes possible a level of performance, stability and reliability unattainable with discrete components.

The list below outlines the many features which make the TN2505, 6, and 7 a unique surveillance product.

### TN2505, 6, 7 Features List

- Extremely small size (3.37"x3.0"x3.04").
- Light weight - 13.5 oz. Rugged construction.
- Single power source: 12-35 VDC at 2 watts, or 117V or 230 VAC 50/60 cy with power supply module.
- Standard RS170 output format (TN2505, 7).
- Standard CCIR output format (TN2506).
- Available with line lock, or crystal, or sync lock control.
- 2.6 million to 1 ALC with optional automatic iris lens. (f1.4 to f360).
- Selectable 0dB, 6dB, 12dB gain boost.
- Extremely good anti-blooming performance.
- Inject inhibit feature for very low light level and "Snapshot" applications.
- True horizontal aperture correction (enhanced image sharpness).
- Automatic black level circuit.
- Adjustable vernier mechanical focus independent of lens.
- 1/4x20 mounting hole on top and bottom.

## Technical Summary - Camera

The following specifications represent typical camera performance. General Electric Company reserves the right to make changes in design which affect specified performance without notification or obligation.

### Electrical

Input Voltage Range: 12.0 to 35.0 OVDC

Input Power: Slightly over 2 watts within specified input voltage range.

Scan Rates:

| | CRYSTAL | | LINELOCK | |
|---|---|---|---|---|
| Model | Vertical | Horizontal | Vertical | Horizontal |
| TN2505,7 | 59.94Hz | 15,734Hz | 60Hz | 15,750Hz |
| TN2506 | 50.00Hz | 15,625Hz | 50Hz | 15,625Hz |

Geometry: Unaffected by input voltage, ambient temperature, magnetic fields.

Signal-to-Noise - Peak Signal to RMS Noise: 50 dB* TN2505
49 dB* TN2506
*at Imager Saturation.

ALC (with Optional Lens): 2.6 million to 1

Manual Gain Selection: 0dB, 6dB, 12dB

| | | | |
|---|---|---|---|
| Displayed Resolution: (pixels) | TN2505 | Horizontal - 377<br>Vertical - 242 | CID17B |
| | TN2506 | Horizontal - 400<br>Vertical - 287.5 | CID18B |
| | TN2507 | Horizontal - 377<br>Vertical - 484 | CID20B |

Gray Scale Rendition: 10 shades of gray, minimum (EIA gray scale chart).

Sensitivity: Full output (1V composite) at .8FC faceplate illumination with 0dB boost gain selected.

Full output (1V composite) at .027FC faceplate illumination with injection inhibit period of .5 second.

Spectral Response: .35$\mu$ to 1.1$\mu$ (see Figure 1)

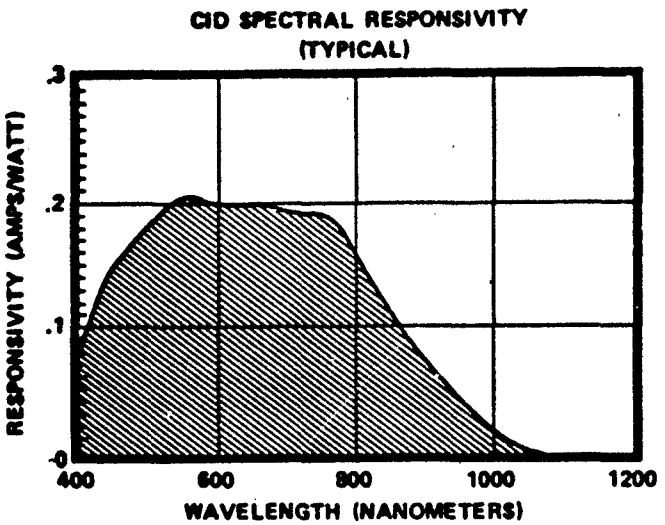



Figure 1. CID Spectral Responsivity

Output Format: 1.0V composite, white positive, RS170 (TN2505), (TN2507), CCIR (TN2506).

## Mechanical

Camera Size: 3.04" long (Less Lens)
3.00" wide
3.37" high

Camera Weight: 13.5 oz.

Lens Mount: Type C

Camera Mounting: 1/4-20 hole top
1/4-20 hole bottom

Mating Connectors: I/O TKP-12-110-TS100T (Viking Industries)
Power TKP-07-110-TS100T(Viking Industries)
Video (BNC) UG 1094/U
Iris TKP-07-110-TS100T (Viking Industries)

Environmental

Temperature: 0°C to 50°C
Relative Humidity: 0-95% non condensing
Maximum Altitude: 20,000 ft.

## Technical Summary - Optional Power Supply Module

Electrical (American) [P/N 232B9116]

| | |
|---|---|
| Input Voltage: | 120V ± 15%<br>50/60 cycle, 2.5 watts |
| Output Voltage: | 24.0 OVDC nominal |
| Output Current: | 180 MA Max. |
| AFC Reference: | 28 VRMS |
| Maximum Current: | 1 MA |

Electrical (European) [P/N 232B9127]

| | |
|---|---|
| Input Voltage: | 220V ± 10%<br>50 cycle, 2.6 watts |
| Output Voltage: | 24.0 OVDC nominal |
| Output Current: | 180 MA Max. |
| AFC Reference: | 28 VRMS |
| Maximum Current: | 1 MA |

Mechanical (American) [P/N 232B9116]

Size: 2.25" long x 2.25" wide x 1.62" high

Cable Length: 124 ± 4"

Weight: 11 oz.

(European) [PN232B9127]

Size: 3.50" long x 2.12" wide x 1.65" high

Cable Length: Input 36 ± 4" Output 124 ± 4"

Weight: 13 oz.

Environmental

Temperature: 0°C to 50°C

Relative Humidity: 95% non condensing

Maximum Altitude: 20,000 ft.

Inspection Prior to Use

Any damage incurred to equipment during shipping should be documented. It is the responsibility of the carrier to insure that equipment arrives in serviceable condition. Note the physical condition of the packing carton, then carefully unpack the contents. Inspect each item for damage. Damage should be reported immediately to the responsible agency. Using the packing slip and the list of equipment supplied below, ascertain that your order is correct. Report any deficiencies to the General Electric Company address indicated on the packing list.

Equipment Supplied: TN2505,6,7 camera without lens

Power supply module and 10 ft interface cable.

I/O connector Viking Industries part number TKP-12-110-TS100T with 12 pins (disassembled).

Power/Lens Connector (disassembled) - Viking Industries part number TKP-07-110-TS100T with 3 pins (A2,A3,A4 models), 5 pins (A1 model).

# 1. INTRODUCTION

Silicon Video allows a PC to digitize, process, display, and transmit video information. Silicon Video will digitize one or several frames from a video camera or other video source, allow the PC to process the image data, display the image data on a B&W or RGB monitor, and allow the PC to transmit the image data via a modem and the telephone network to a similarly equipped PC at a remote location. Video data is digitized and displayed at 8 bits per pixel.

Silicon Video can contain up to 1MB of image memory for multiple image storage or use of interlaced images greater than 540 pixels per line.

The unique features that distinguish the Silicon Video from other video imaging products are:

- Variable sampling,
- 1MB image memory,
- Video memory address registers,
- 752 by 480 maximum resolution,
- Multiple image storage, and
- Extensive software base.

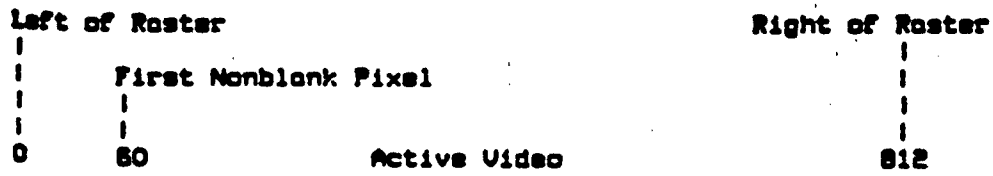
The SVPCIP software (which is provided at no cost for each Silicon Video) allows the user to:

- Select variable resolution sampling parameters,
- Digitize one or several images,
- Display one or several images,
- Digitize and display portions of the raster,
- Read/write images or portions of images to/from disk with or without compression,
- Perform addition, subtraction, differencing, insertion, duplication, and averaging on two images or portions of two images,
- Perform bit-strip, histogram, threshold, exclusive-or, contrast stretch, reduction, print, half-tone, and dither on one image or portion of an image,
- Perform convolution or dynamic thresholding using a 3 by 3 to 7 by 7 pixel kernel on an image or portion of an image,
- Overlay text on an image with up to 83 characters by 25 lines available.

Silicon Video uses conventional dynamic random access memory integrated circuits (DRAM). This allows memory costs to be kept low and allows the basic design to increase in capacity as the capacity of DRAM increases.

Silicon Video has a dual ported image memory that allows access by the PC and by video. While the image memory has dual access, it may not be accessed by video and PC simultaneously. The SVPCIP software accesses the image memory in two modes: when video is enabled it accesses during the vertical interval, and when video is disabled it accesses as required. This allows the user to perform image processing and view the results as the processing is taking place, while providing a higher speed mode when repetitive operations are to be performed and human monitoring of the processing is not required.

The sampling and display of pixels is controlled by a micro sequencer. This allows specific pixels to be digitized or displayed. It also allows pixels to be both digitized and displayed on the same raster. The programmable portion of the micro sequencer is called the XSAM (X Sample Memory). It is constucted of a pixel counter that provides the address to a high speed memory and register. At the start of each horizontal line the counter is cleared to zero. At the trailing edge of horizontal drive (the left of the raster) the counter is incremented by each pixel clock. The outputs of the memory are clocked into a register that controls the sampling and display of pixels and other functions. The XSAM is disabled during vertical blanking or when the PC loads the bit in the control register that disables the XSAM. The XSAM addresses that correspond to the video raster are shown below:




Note that there is a maximum of 752 pixels that are active video.

The XSAM may be programmed to digitize less than 752 pixels on each active line of video. The video may be digitized as one frame of information (two fields) or one field of information. Each field of information results in 240 lines of information. The highest resolution is obtained using a frame of video and 752 pixels per line. However, this requires the 1MB image memory option. The maximum resolution image that the 256KB image memory will support for a frame of video is 540 pixels by the 480 active lines. Some of the possible number of images that the 1MB version supports are:

2 images at 752 pixels by 480 lines,
3 images at 640 pixels by 480 lines,
4 images at 512 pixels by 480 lines,
6 images at 640 pixels by 240 lines,
8 images at 512 pixels by 240 lines, or
17 images at 256 pixels by 240 lines.

Note that these are a very small number of the possibilities.

## Some Advantages of the 286s

Your PC's Limited 286[10] or 286[12] offer a number of advanced features which place them at the forefront of powerful, adaptable yet easy-to-use personal computers. Here are just a few of those features:

- The 286s work at both the original IBM PC-AT's 6 MHz speed, and also at either 10 MHz or 12 MHz, depending upon which model you've purchased. This isn't a matter of just changing a clock-crystal, but rather the result of a fresh approach to computer design.

  This increase in speed translates into much faster processing of data in your work. By most popular benchmarks for operating speed in personal computers, the 286[10] and 286[12] are up to ten times faster than the original IBM PC.

  The actual increase in total system performance you'll find when working at your 286 (or "throughput," in computer lingo) will depend, on the programs you use and the speed of connected peripherals, such as modems and printers.

- Your PC's Limited 286 is dramatically smaller than earlier designs. By saving as much as 25% of the usual "footprint," or space occupied by the computer's cabinet, you regain space on your desk for other items.

- The six 16-bit (PC-AT-style) expansion slots and two 8-bit (PC-style) expansion slots in the PC's Limited 286s give you more for your money by allowing you to expand your computer with add-in modems, video cards, multiple I/O ports, expanded-memory cards, speech-recognition systems and many other exciting products. Unlike other "AT compatibles," the PC's Limited 286s don't cripple your ability to expand your computer to meet your changing needs.

- As you know, your PC's Limited 286 costs less—often, much less—than older, slower designs with fewer features.

# Appendix J
# Specifications

*size, large chassis*
*.....6.5 H x 21 W x 16.5 D in./16.5 x 53.3 x 41.9 cm*
*size, small chassis*
*... 6.5 H x 18.75 W x 16.5 D in./16.5 x 47.6 x 41.9 cm*
*weight.....55 lbs/25 kilos*
*microprocessor.....80286*
*clock speed (user-selectable, from keyboard)*
*.....6 MHz, 10 MHz ($286^{10}$)*
*.....6 MHz, 12 MHz ($286^{12}$)*
*installed RAM.....1024K*
*RAM division option (user-selectable)*
*.....512K for DOS programs, 512K for VDISK*
*.....640K for DOS programs, 384K for VDISK*
*.....256K for DOS programs*
*.....512K for DOS programs*
*.....640K for DOS programs*
*number of drives installable*
*.....1 full-height, 3 half-height, large chassis*
*.....3 half-height, small chassis*
*expansion slots*
*.....6 IBM PC-AT-compatible (16-bit)*
*..... 2 IBM PC-compatible (8-bit)*
*voltage (user-selectable .....110v/220v, 50/60 Hz*
*power supply.....192 watts*
*typical power consumption at 110v, with*
*..... 1.2 Mb floppy-disk drive, 15 watts*
*.....1.2 Mb, 360 Kb floppy disk drives, 18 watts*
*..... 1.2 Mb floppy, 40 Mb hard-disk drive, 24 watts*
*.....1.2 Mb, 40 Mb, 60 Mb tape drive, 33 watts*