(NASA-CR-190276) [REPORTS ON COMPUTER GRAPHICS TESTBED TO SIMULATE AND TEST VISION SYSTEMS FOR SPACE APPLICATIONS] Annual Status Reports (Research Inst. for Computing and Information Systems) 126 p G3/61

N92-24540
--THRU--
N92-24543
Unclas
0086905

STATUS REPORTS:

1) March 1988

J.B. Cheatham, Jr., C.K. Wu and Y.H. Lin

2) March 1990

J.B. Cheatham, Jr.

3) July 1991

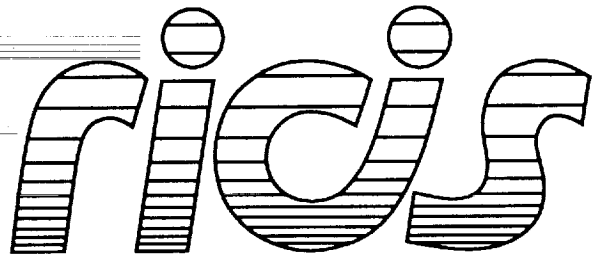
J.B. Cheatham, Jr.

Department of Mechanical Engineering and Materials Science
Rice University

Cooperative Agreement NCC 9-16

Research Activity No. AI.02:
A Computer Graphics Testbed to Simulate and Test Vision System for Space Applications

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

INTERIM REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Dr. John B. Cheatham Jr., Professor of Mechanical Engineering, Rice University. Dr. Terry Feagin served as RICIS research coordinator.

Funding was provided by the Information Systems Directorate, Information Technology Division, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this research activity was Dr. Timothy F. Cleghorn of the Information Technology Division, NASA/JSC.

ANNUAL REPORT

TO

RESEARCH INSTITUTE FOR

COMPUTING AND INFORMATION SYSTEMS

(RICIS)

ON

COMPUTER GRAPHICS TESTBED TO SIMULATE AND TEST VISION SYSTEMS FOR SPACE APPLICATIONS

By

J. B. CHEATHAM, C. K. WU and Y. H. LIN

for

10/15/86 - 10/15/87

GRANT NCC 9-16

MECHANICAL ENGINEERING AND MATERIALS SCIENCE DEPARTMENT

RICE UNIVERSITY

HOUSTON, TX 77251-1892

MARCH 1988

Table of Contents

COMPUTER GRAPHICS TESTBED TO SIMULATE AND TEST VISION SYSTEMS FOR SPACE APPLICATIONS

Principal Investigator: John B. Cheatham, (713) 527-4822
Professor of Mechanical Engineering
Rice University, Houston, TX 77251-1892

Executive Summary

Objectives: The objectives of this project are to develop a system for displaying computer graphics images of space objects and to demonstrate the use of this system as a testbed for evaluating vision systems for space applications.

Overview: In order to evaluate vision systems, it is desirable to be able to control all factors involved in creating the images used for processing by the vision system. Considerable time and expense is involved in building accurate physical models of space objects. Also, precise location of the model relative to the viewer and accurate location of the light source require additional effort. Although the motion of a small satellite model can be controlled by a robotic manipulator, such as a PUMA, this task requires additional equipment and time. On the other hand, the position and motion of a computer graphics generated model can be controlled accurately with precise light location.

As part of this project, graphics models of space objects such as the Solarmax satellite are created such that the user can control the light direction and the relative position of the object and the viewer. The work is also aimed at providing control of hue, shading, noise and shadows for use in demonstrating and testing imaging processing techniques. The simulated camera data can provide XYZ coordinates, pitch, yaw and roll for the models. A physical model is also being used to provide comparison of camera images with the graphics images.

INTRODUCTION

A robotics software simulation testbed is being developed by NASA Johnson Space Center to identify the enabling automation and robotics technologies for space operations into the next century. The project described in this report is aimed at evaluating vision systems for space applications and providing assistance in the overall effort to develop the robotics simulation software testbed. The three complimentary efforts involved in this project are graphics modeling, physical modeling, and vision system evaluation. Research results for each of these efforts is described below.

Initial work on the graphics modeling was done using APL GRAPHPAK software. This permitted rapid conversion of the dataset for a graphics model of the Solarmax satellite to be drawn using a dataset obtained from Johnson Space Center. A FORTRAN program was written to permit conversion of the dataset for use with a Sun workstation working with a network file server with the mechanical engineering department Celerity C1200 computer. Once data had been translated over in a form suitable for use with the Sun III workstation, the testbed was developed in a form containing menus that permit the user interactive control of the graphics modeling. Results using a simple physical model are compared with those obtained with the graphics model and both of these models are utilized in the evaluation of a computer vision system. The 3M VDL vision system is described and examples of its application in image processing are given.

GRAPHICS MODELING USING APL GRAPHPAK

One of the objectives of this research is to create a computer graphics testbed for simulating images of a slowly spinning satellite as viewed by an autonomous robot vehicle vision system. In this work it is necessary to control the relative location and attitude of a satellite with respect to the observer.

In the preliminary work on this project APL was used to generate images of the Solarmax satellite. A brief discussion of image generation using APL is presented and results are given for several example positions of the satellite relative to the viewer. Computer programs, data and additional output images are collected in the Appendices.

Image Generation Using APL

Images were produced for different orientations of a satellite with hidden line removal. Rotation of the reference frame involves matrix manipulation that APL can handle concisely and powerfully.

The two features of APL that are extremely useful are:

1) the use of arrays to store all kinds of data
2) the use of a large number of mathematical functions (each coded as a single symbol) to manipulate arrays.

One main advantage of this approach is the great number of operations that can be performed with very short programs. A second advantage is the fact that most of the operations are very similar to those seen in mathematics. Unfortunately, to be able to use the language, the user must learn:

1) the meaning of quite a few special symbols;
2) ways to organize data into array format.

GRAPHPAK is an APL workspace containing predefined functions that produce drawings on graphics devices. GRAPHPAK provides us with the ability to draw and transform projections of 3-dimensional objects.

A data set was prepared for the Solarmax satellite and programs or functions were written for hidden-line removal to handle data transmission between TSO and APL environments. These functions and data sets are given in the Appendices.

Three-dimensional objects are represented by a 4 column array, the last three are the x, y and z coordinates of a point and the first column contains either a zero or one. APL uses the line method to draw, so a 0 means "move", and a 1 means "draw". The coordinates used are screen coordinates, with positive-x to the right, positive-y up, and positive-z out of the screen.

The original data set "solarmax.vec" listed in the Appendix was modified to permit hidden-line removal. For convenience, we separated the data set into seven groups, which are ANT, BASE, BODY, HOLE, JET, SPAN and TORSO. Also the vertices for each surface must appear in a counter clockwise order looking toward the surface. Function "pickup" can create the required data from raw data by specifying the surface number only (see Appendix).

Finally, we combined all the subpart data, already processed by the function "pickup", into one big data set "solart". The data set "rowse" is created by the starting and ending vertex number of every surface. Both data sets are fed into function "hidden".

APL does not have an edit function. When editing the data set, we can transmit the APL data out to TSO mode, edit those data by the TSO editor and then transmit back to APL mode. The functions "IN" and "OUT", listed in Appendix C, can help us to transmit data back and forth.

It is not easy to identify the 3-D vertices of every surface on a 2-D screen. The function "analyze" draws the picture step by step to permit one to determine the coordinates of any vertex in the picture.

Hidden-Line Removal

A surface is defined by at least three vertices. The identification of vertex 1 is arbitrary, but it is important that the numbering of the remaining vertices of the surface continue in a counterclockwise direction, as viewed from outside the object, facing the surface. We then

identify vector $\vec{u}$, directed from vertex number 1 to vertex number 2, and vector $\vec{v}$, directed from vertex number 1 to vertex number 3. The cross product, $\vec{n} = \vec{u} \times \vec{v}$, will be normal to the face of the object and will be directed outward.

With each surface of an object we also associate a second vector $\vec{w}$, a line-of-sight vector. This is the vector directed from vertex number 1 of the surface to the viewpoint (the location of the eye of the viewer). The dot product of vectors $\vec{w}$ and $\vec{n}$ has the property

$$\vec{w} \cdot \vec{n} = |\vec{w}| \; |\vec{n}| \cos\theta$$

where θ is the angle between $\vec{w}$ and $\vec{n}$. For a visible surface, the angle between $\vec{w}$ and $\vec{n}$ is between 0° and 90°, and $\vec{w} \cdot \vec{n}$ is positive. For a hidden surface, the angle between $\vec{w}$ and $\vec{n}$ is between 90° and 180°, and $\vec{w} \cdot \vec{n}$ is negative.

Function "visible" listed in the Appendix is the visibility filter. Those surfaces which face the viewer will pass through the filter, eventually to be plotted. Those surfaces which do not face the viewer will be rejected.

The method described above permits drawing of the visible portions of a single convex object. If we try to draw the visible portions of several objects, it will be necessary to identify those portions which are hidden by a closer object. We could consider that the hidden portions of an object have been filled, but were covered by the filling of a closer object.

The major part of blackout is to determine the drawing sequence of every visible surface of the whole satellite consisting of seven subparts. The function "hidden" listed in the Appendix plots every visible surface in the order of the magnitude of distance between the surface and the viewpoint.

Graphics Models of Solarmax

The seven component parts for the Solarmax satellite have been drawn with top, front and side views using tne APL threeviews function as shown in Appendix A. These were plotted on a monochrome Tektronic graphics terminal (TEK4010). A plot of the entire Solarmax satellite using the threeview function on the data set SOLART is shown in Fig. 1. The x, y, z axes for the satellite are drawn on this figure as well as the rotations required to show the top and front views. These rotation values are angles in degrees for rotations about the x, y and z axes.

The graphical displays for the satellite with hidden-line removal have been drawn using a TEK4105 color graphics terminal. An example of the results using this procedure are shown in Fig. 2. The allocated data sets are "solart" and "rowse". The execution procedure is

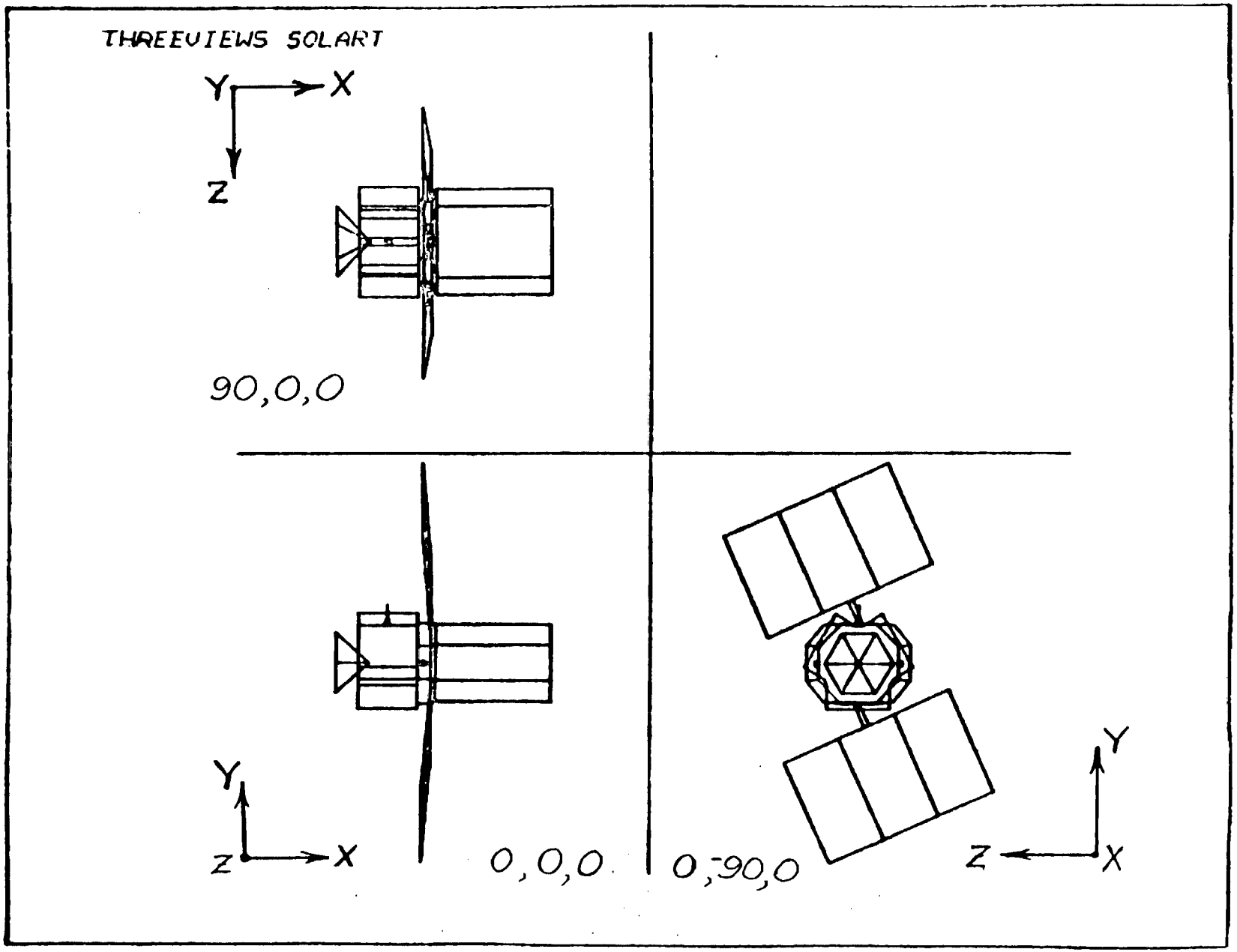


Fig. 1: Threeviews of Satellite

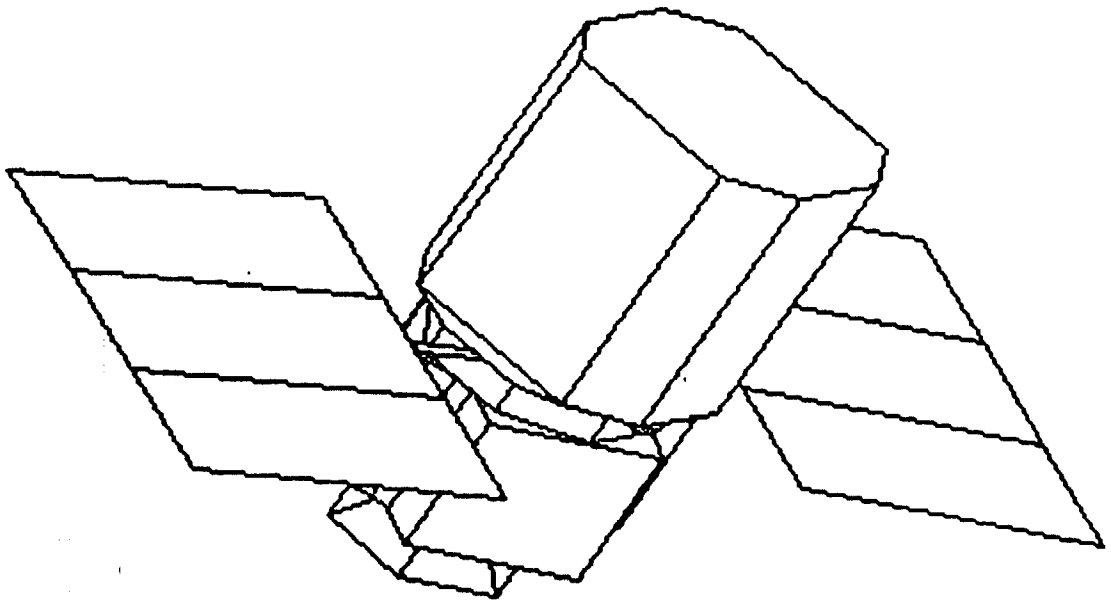


Fig. 2: -60°, 30°, 60° Attitude

θ_x θ_y θ_z hidden solart

where θ_x, θ_y and θ_z are the rotational angles in degrees about the x, y and z axes. It can be seen from these results that it is possible to control the relative attitude of the satellite with respect to the viewer.

Simulation Model

We now advance to simulate a slowly spinning satellite as viewed by an autonomous robot vehicle vision system. Because a perspective projection is needed for this simulation, the function "visible" is changed and listed in the Appendix.

Two different types of simulated vision systems are studied. One of them has the camera view fixed as it moves toward the object along a certain desired path. We call this type the "fixed vision system" which is easily modeled by experiment. Another one, called the "tracking vision system", has the camera view adjusted automatically to focus on the object no matter what the path is. We study these two different vision systems in the following.

Fixed Vision System

In this model, the robot approaches the satellite and follows the path which is drawn as threeviews. This figure, containing an extra oblique projection, is generated by the new function "threeviews" which is modified from the original one and listed in the Appendix. Firstly, we create the data set TRACE, listed in the Appendix, which contains the attitude data from the columns 1 to 3 and 3-D path data from columns 4 to 6. A function MOTION, listed in the Appendix, reads the data from TRACE and generates the images one by one. The results are plotted on one picture to show the changes of attitude and size of satellite when the camera approaches the spinning satellite (see Fig. 3).

Tracking Vision System

In the tracking vision model, the robot camera always focuses on the satellite and is adjusted whenever the path is changed. The path data set HELIX, listed in the Appendix, contains the spin angle θ_y in degrees in column 1, and Y and Z positions in columns 2 and 3. The helix path projection on the X-Z plane is governed by the following formula in polar coordinates

$$\gamma = -\frac{4}{3}\theta_y + 60$$

where θ_y is the spin angle (degree) about the Y axis and γ is the distance between the camera and the satellite.

A new function ROBOT, listed in the Appendix, reads the data from HELIX and generates the images sequentially (see Figures 4 and 5).

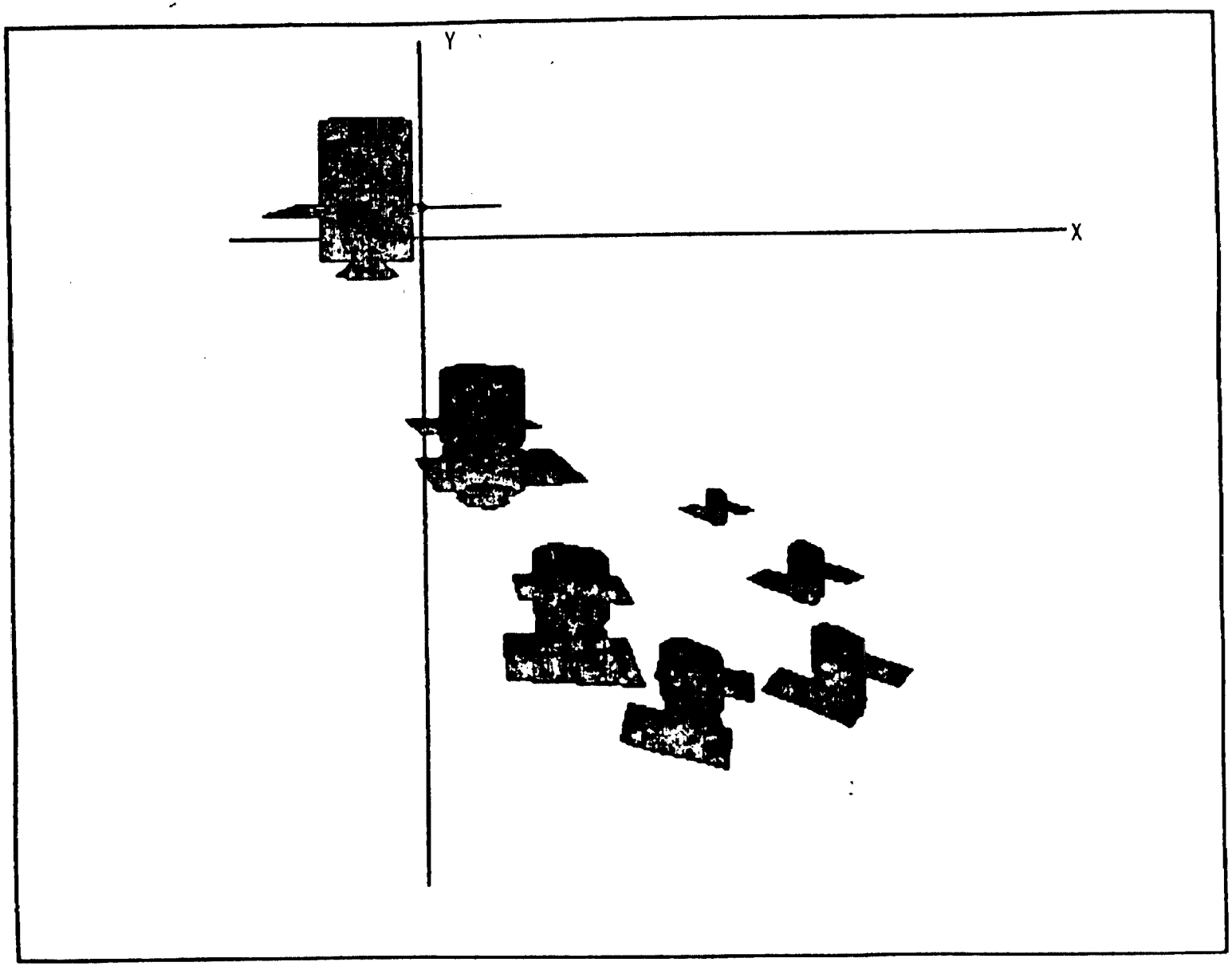


Fig. 3 Results of Fixed Vision System

Fig. 4: Satellite at $y = -180$
$z = -300$
$\theta_y = 270$

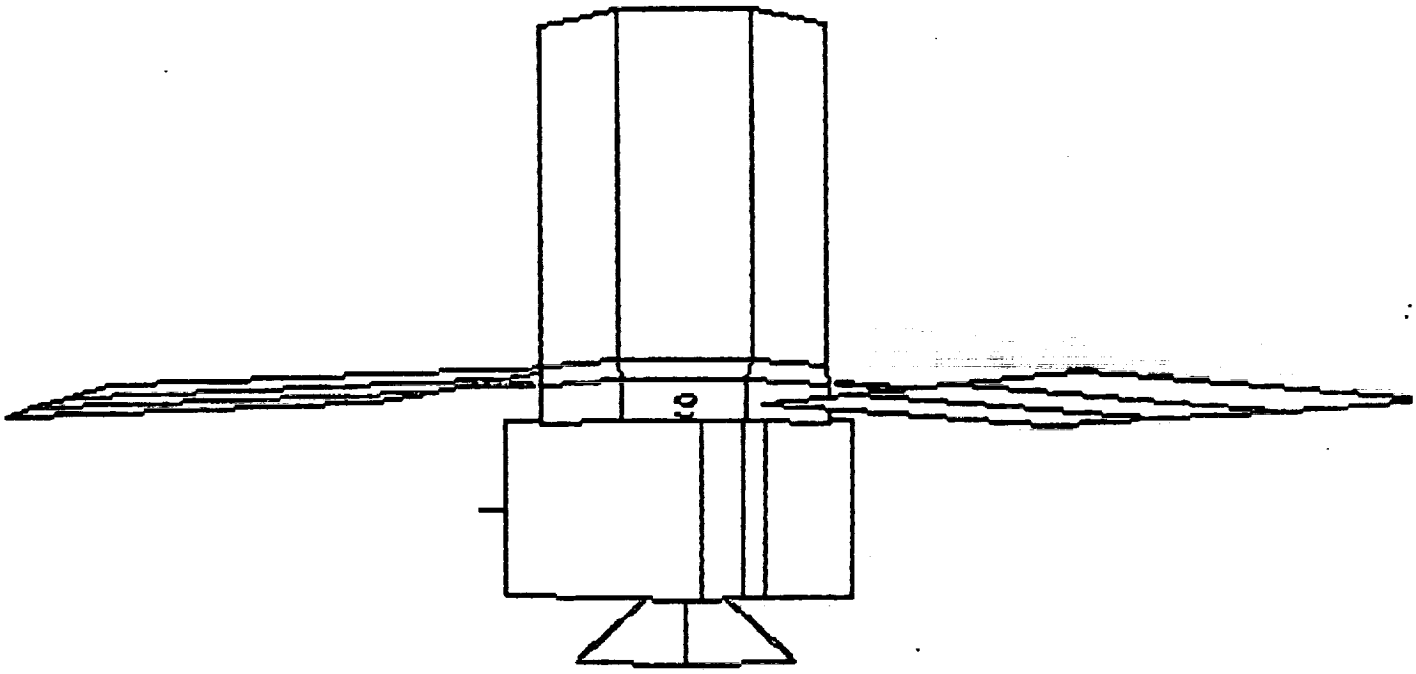


Fig. 5: Satellite at $y = 0$
$z = 60$
$\theta_y = 0°$

Preliminary computer graphics images of the Solarmax satellite as described above were generated using APL on the Rice University AS9000 mainframe computer and TEK4010 and TEK4105 graphics terminals. The next step of the process of developing the computer graphics modeling involved transferring these data to the SUN-III workstation.

GRAPHICS MODELING ON THE SUN WORKSTATION

A complete graphics model of the Solarmax satellite has been generated on the SUN-III workstation. The model can be manipulated to show arbitrary views and different shading effects. The SUN-III workstation supports a SunCore package which implements the ACM SIGGRAPH core system that conforms to level 3C (dynamic output with 3D scaling, rotation and translation) of the core specification for output primitives, and to level 2 (complete input) for input primitives.

Data Generation

The satellite data were transferred from the IBM 370 to the Mechanical Engineering Celerity C1200 computer. These data originally were bulky and complicated, since some vertex coordinates which were repeated several times were easily implemented in the APL graphics system. To reduce the size of data, a FORTRAN code (Appendix), "trans.f", was implemented to accomplish this job. There are two input files needed for this code. One is a vertex coordinates data file, "s.s". Another is a file "s.n" containing the index of each vertex for each plane. After running the program, the output file solar.s is generated in a concise form which covers every vertex only once. Based on the data, a program "shad" has again been implemented successfully for displaying graphic models with choice of different shading. The data of the solar plate is entered once as a no-thickness plane, not a solid object. But in an APL system, we enter the data twice to simulate two opposite faces. However, the "shad" program does work for a no-thickness panel very well.

Description of Program

The C program on the SUN workstation is described in this section. The input data sets are allocated as follows:

no. of vertices no. of surfaces
W_{-x} W_x W_{-y} W_y W_{-z} W_z = window coordinates
x y z: coordinates of each vertex

no. of vertices in each surface every vertex index on each surface

The C program "shad.c" originally written for demonstration purposes by Sun Microsystems, has been modified to provide the following menu driven screens:

Screen 1: Enter the desired shading style

1) Wireframe display
2) Gray shading
3) Gourand
4) Phong diffuse [default]
5) Phong specular

Enter your choice (1-5)

Screen 2:

1) Gray
2) Red
3) Green [default]
4) Blue
5) Yellow

Enter your choice (1-5)

Screen 3: Enter the desired display option

1) Still frame
2) Rotate the viewer [default]
3) Rotate the object
4) Rotate the light source
5) Quit

Enter your choice (1-5)

Note: "rotate" means rotation about vertical y axis.

Screen 4: Enter the light source position [default: 0.0, 0.0, -1.0]

x =
y =
z =

Enter the viewer position [default: 4000, 8000, 6000]

x =
y =
z =

Screen 5: Enter noise level

1) White noise
2) Coherent noise
3) No noise [default]

Enter your choice (1-3)

Note: "The light source position" specifies the direction of the light source from the object. The direction is expressed in NDC (normalized device coordinates).

This system permits the viewer to control the shading, color, motion, light source position and viewer position.

Artificial noise can also be introduced. Gaussian white noise or coherent noise can be generated and superimposed on the graphics picture. This operation permits the simulation of a picture transmitted via a noisy channel.

Wireframe displays of the Solarmax satellite model and a hexagonal or hex model are shown in Fig. 6 and the corresponding gray shading models are shown in Fig. 7. The hex model is used to illustrate application of the graphics model to evaluation of a commercial vision system.

PHYSICAL MODELING

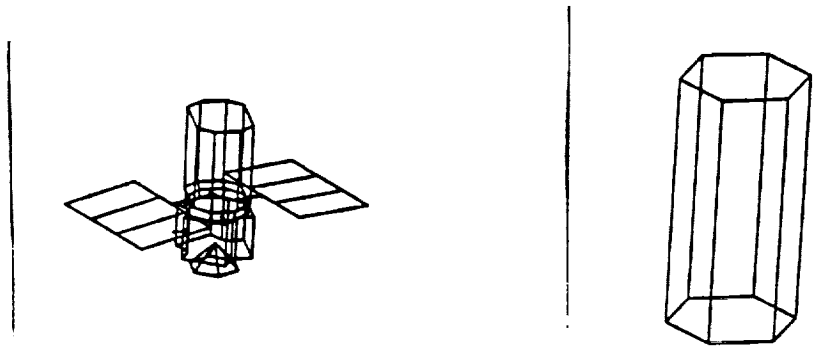
A 6" x 12" (base x height) hexagonal box was assembled to simulate the body of the Solarmax satellite. This model is supported by a thin rod that is attached to the PUMA 560 Robot. The model can be moved by the robot arm to simulate the navigation of a satellite in space.

A communication channel between the PUMA 560 controller and the IBM-PC XT was established and a BASIC program that transfers 6 real numbers (dx, dy, dz, pitch, yaw, and roll) to the PUMA from the PC was implemented and tested successfully. A VAL II program was written which receives the six numbers transferred from the PC and moves the PUMA manipulator arm to the corresponding position. This system permits control of position and orientation of the model. Considerable time and expense is involved in building accurate physical models of space objects and precise location of the camera and light source require additional effort. On the other hand, the position and motion of the computer grahics model can be controlled accurately with precise light location.

Pictures of the hex model in a room with a single bright light source are similar to those produced by the graphics system. Results of the image processing procedures are comparable for either the physical or graphics models.

COMPUTER VISION SYSTEM

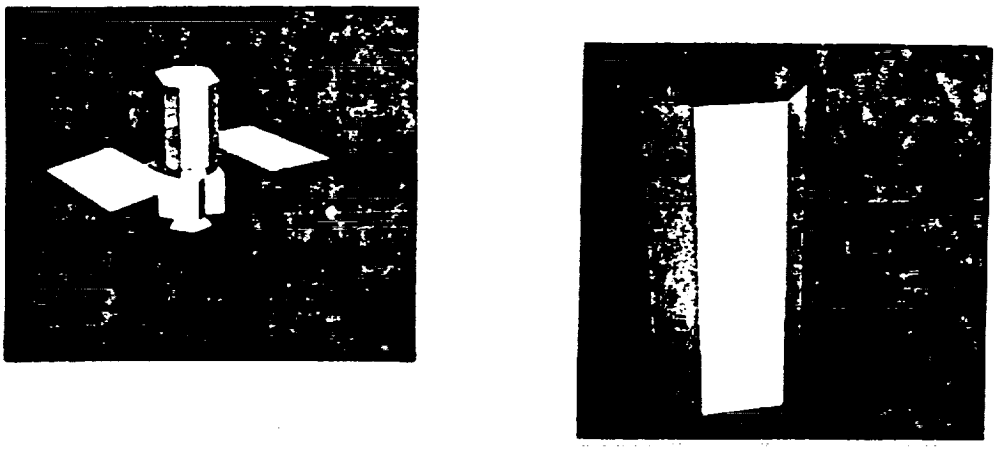
A 3M VDL (vision development language) vision system is used in the work described in this report. This system is a high-speed vision development workstation with a digitizer that provides a resolution of 512 x 512 x 8. This system provides a wide variety of vision algorithms for picture acquisition including filtering, feature enhancement, image segmentation and feature extraction, statistical data manipulation, and object data base matching. These algorithms are encoded into macro forms (also called primitives) that can be used as building blocks in a high-level language program designed for a specific vision application. These macros can be used interactively, in a command interpreter called VDL-BASIC or by a C compiler using VDL software in a C command subroutine library.



a b

Figure 6: Graphics Wireframe Models

(a) Solarmax Satellite, (b) Hex Cylinder



a b

Figure 7: Computer Graphics Gray Shading Models

(a) Solarmax Satellite, (b) Hex Cylinder

Vision Primitives

The upgraded VDL software provides 151 image processing primitives in the areas of control, image I/O, transformation, convolution, graphics, color, dilation/erosion, segmentation, statistics, image arithmetic, threshold, edge detection, transition detection, and neighborhood transformations. All these primitives are implemented such that the images are processed in the spatial domain. These spatial domain implementations have the advantage of fast processing time. Indeed, most of the primitives take only a few seconds of processing time.

Those primitives applicable to the space applications, such as location determinations and object recognition, are especially of interest. They were studied extensively and are discussed below.

Edge Detection

Ten VDL primitives are available for finding edges in a binary or gray image. Nine primitives, (BIN(), KIR(), ROB(), SOB(), LGR(), HGR(), VGR(), VDX(), and VDY()), employ the information of the first order gradient. The existence of an edge is asserted if the gradient exceeds a given threshold value. Five of the first order edge detectors are two dimensional and four are one dimensional. One 2-D second order gradient edge detector, GCR(), finds edges by means of zero-crossing. The processing time is about 5 seconds for an image of size 256 x 256. It can be reduced proportionally by reducing the work window size.

Dilation/Erosion

Dilation and erosion commands (DIL(), ERO(), etc.) are available for both binary and gray images. The maximum number of pixel neighbors that are used in the dilation/erosion procedure is limited to 8 pixels.

Noise Removal

Two commands are implemented for the purpose of relieving noise by averaging the pixel values. The AVN() command averages the pixel values in the neighborhood of 8 pixels. The GAU() primitive performs a weighted averaging process.

Statistics and Transition Detection

Several statistics commands and transition detectors are very useful in determining the position information of an object. The CEN() command returns the center of gravity of a binary image. The MXX() command finds the right most edge of a gray object. The LXX() command detects the right edge along a line, etc. By using the combination of these primitives, position and orientation of simple geometries can be easily determined.

Programming Structure

The VDL software provides a very limited programming structure. To extend the flexibility of the use of the system, a C compiler was added to the system. The VDL software provides a C command subroutine library. With the command library, all 151 vision primitives can be called from a C program. Several C programs, (Appendix), were implemented to examine the performance of the primitives under the C language. The only difference observed is that the processing time is slightly increased.

One C program 'ft.c' was implemented to compute the 2-D Fourier Transformation of an image of size 64 x 64. The image is read from the frame buffer 1 and the results are stored in two data files. The execution time for this program takes about 20 minutes. This implies that if frequency domain processing is required by using this system, a long processing time must be taken into consideration.

Image Processing

Application of the vision system to image processing of the hex model is described in this section. Terminology used is defined in a glossary at the end of the report. A digitized picture after use of the Sobel edge detector is shown in Fig. 8. The noise in this picture can be eliminated by additional image processing.

A histogram for the digitized image is plotted in Fig. 9a. This chart displays the number of pixels (picture elements) at each gray level from -128 for black to 127 for white. There should be five clusters of gray levels for the four sides of the hex and the backgroun, however, there is considerable overlapping of the brightness intensities of the two sides on the white end of the chart. The dark background can be seen as a sharp spike at the left side of the chart representing the darkest pixels. This histogram was segmented interactively by mapping ranges of gray values corresponding to each of the faces and the background into single gray level values using a partial threshold command. Results are shown by the segmented histogram of Fig. 9b.

The following procedure can be used to display the top surface as white and all of the other surfaces and the background as black. the partial threshold command PTH(start-value, end-value, replace-value) maps all pixels with gray levels between start and end values into a single gray levels having replace-value. Thus PTH(-128, -90, -128) maps all values below -90 into black (-128) and PTH(-60, 127, -128) maps the range from -60 to 127 to black. Then PTH(-90, -60, 127) maps the top surface into white (127). ADD() combines the white top surface with the image in the second frame buffer. When the image of Figure 8 is in the second frame buffer one obtains the result shown in Figure 10. Each of the faces can be isolated in this manner.

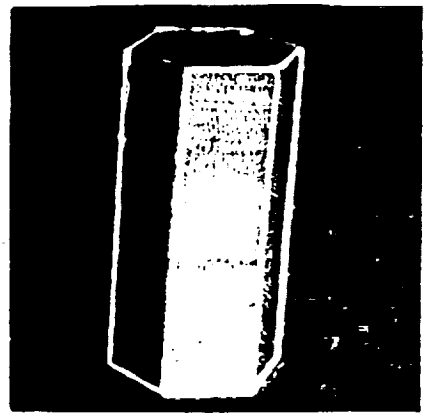
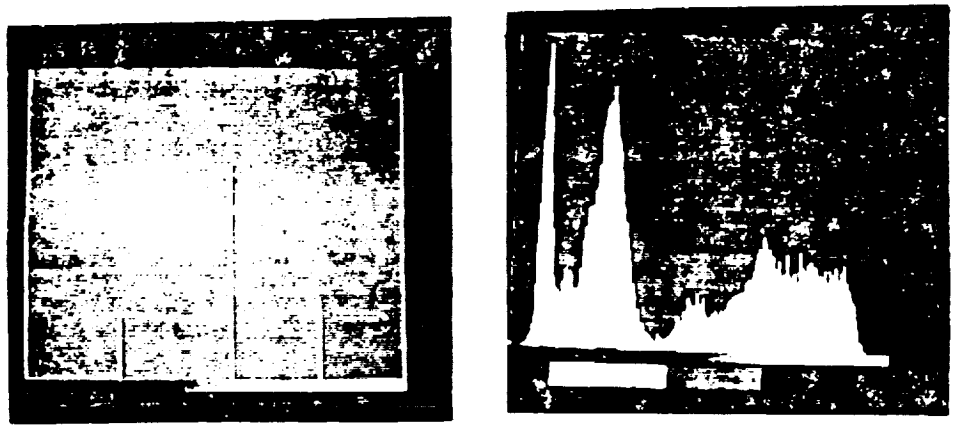


Figure 8: Hex Model - Output of Sobel Edge Detector



a b

Figure 9: (a) Histogram of Hex Model Image,
(b) Histogram after Segmentation

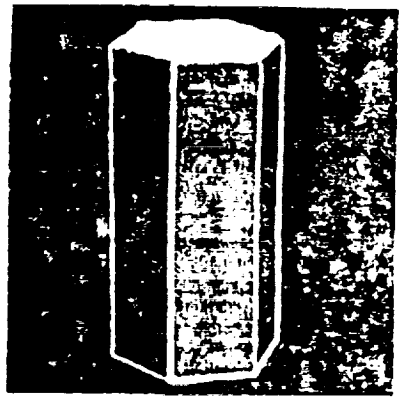


Figure 10 Display of Top Surface as White after Using Partial Threshold Command

By adding all of the surfaces using the segmented histogram of fig. 6b one obtains the image shown in Fig. 11a. Here each of the surfaces contains only a single gray level. Now using the Sobel edge detector on this final image one obtains the wireframe model shown in Fig. 11b. The final result of the image processing procedure is to produce a clean, noise-free model and a wireframe model that is essentially identical to the graphics wireframe model used in the construction of the image. Such a wireframe model can be used by vision algorithms to identify objects from a library of properties of objects stored in computer memory. This exercise illustrates applications of computer graphics modeling to an image processing procedure that utilized several of the macros provided by the vision system.

SUMMARY AND CONCLUSIONS

Computer graphics modeling has been performed using both the APL GRAPHPAK and a C program written for the SUN workstation. These models permit user control of model location and orientation, light location, viewer position, surface shading parameters, color and introduction of artificial noise onto the image. A physical model was constructed and image processing techniques were applied to both the graphics and physical models. It has been found that the graphics models compare favorably with the physical model and offer the advantages of being less expensive and easier to provide control of the important parameters, such as light source and viewer positions.

Preliminary evaluations of the 3M VDL vision development language system indicated that serious image processing could not be accomplished well without the addition of a C compiler. This has been done and examples of applications of the vision system to both the graphically generated models and the physical model have been presented.

GLOSSARY

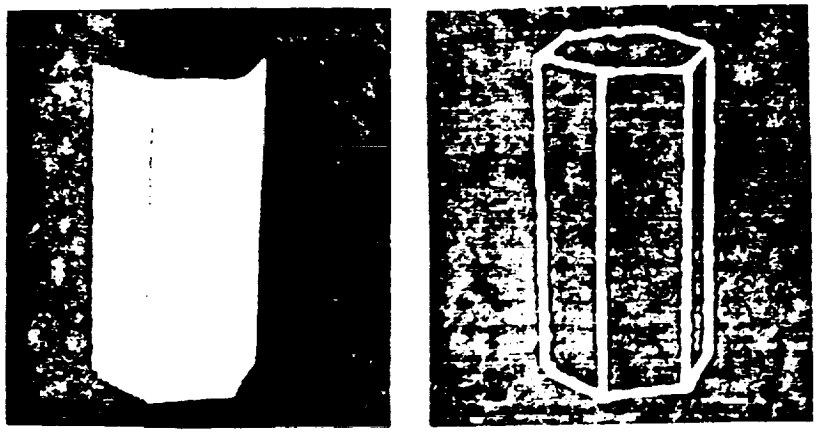
DIGITIZATION: the process by which analog image signals are converted into intensity values and are assigned to pixels of the digitized image.

PIXEL: short for "PI(X)cture ELement." The smallest resolvable area in an image; an individual photosite on a solid state camera.

RESOLUTION: the measure of a system's ability to represent images; the number of bits of accuracy or number of gray levels that can be represented in a pixel. Also, the number of pixel columns and rows in an image raster.

DIGITIZER (FRAME GRABBER): a frame grabber; the device that samples analog video input data, converts them to digital values and stores them in computer memory.

FRAME BUFFER: computer memory specially designed to hold image data and allow for simultaneous video display.



a b

Figure 11: (a) Hex Model Image after Segmentation

(b) Output of Sobel Edge Detector

INTENSITY: the degree of brightness of a pixel, determined by the value stored at the pixel's address in memory.

GRAY LEVEL (GRAY SCALE): the intensity of brightness, stored as a discrete value in each pixel.

HISTOGRAM: a chart showing the frequency of occurence of gray levels in an image.

THRESHOLD: a point at which a pass-fail decision is made. More specifically, the intensity value below which a pixel is forced to black, and equal to or above which a pixel is forced to black, and equal to or above which a pixel is forced to white.

BINARY IMAGE: an image represented in black and white intensities only.

GRADIENT: the rate of change in pixel intensity linearly over the raster.

EDGE: regions of an image where intenstiy levels change rapidly, representing a border between distinct areas.

EDGE ENHANCEMENT: process by which the image intensity changes across an edge are heightened.

EDGE DETECTION: process of locating an edge by inspecting a pixel and its neighbors and finding rapid changes in intensities.

SEGMENTATION: the process of separating objects of interest from their image context or background.

NOISE: meaningless data in an image not related to the image and produced by a variety of sources, such as the camera.

SUMMARY OF 3M VDL MACROS

BIN() = Binary edge detector
KIR() = Kirsch edge detector
ROB() = Roberts edge detector
SOB() = Sobel edge detector
LGR() = Largest gradient edge detector
HGR() = Horizontal gadient edge detector
VGR() = Vertical gradient edge detector
VDY() = Hough vertical gradient
DIL() = Dilate gray image
ERO() = Erode gray image
AVN() = Average neighbor
GAU() = Gaussian (low pass) filter
CEN() = Center of gravity
MXX() = Rightmost edge of an object
LXX() = Rightmost edge pixel along a line
PTH() = Partial thresholding operation
THR() = Thresholding operation

BIBLIOGRAPHY

1. William M. Newman and Robert F. Sproull. Principles of Interactive Computer Graphics, 2nd Edition, McGraw-Hill, New York, 1979.

2. James D. Foley and Andries Van Dam. Fundamentals of Interactive Computer Graphics, Addision Wesley Publishing Company, Reading, MA, 1982.

3. Theo Pavlidis. algorithms for Graphics and Image Processing, Computer Science Press, Rockville, MD, 1982.

4. SUN Reference Manuals, Sun Microsystem, Inc., Mountain View, CA, 1986.

5. Berthold Klaus Paul Horn. Robot Vision, McGraw-Hill, New York, 1986.

6. Alan Pugh, Editor. Robot Vision, Springer-Verlag, Berlin, 1983.

7. Dana H. Ballard and Christopher M. Brown. Computer Vision, Prentice Hall, Englewood Cliffs, N. J., 1982.

8. Rafael C. Gonzales and Paul Wintz. Digital Image Processing, Addison-Wesley, Reading, MA, 1977.

9. OS-9/6800 Operating System User's Manual, Microware Systems Corporation, Des Moines, Iowa, 1985.

10. VDL PC (Vision Development Language) User's Manual, Version 1.1, 3M Vision Systems, Minnesota Mining and Manufacturing Company, McLean, VA, 1986.

11. D. F. Rogers, Procedural Elements for Computer Graphics, McGraw-Hill Book Co., New York, 1985.

12. P. Bergeron, "A General Version of Crow's Shadow Volumes", IEEE CG & A, September 1986, pp. 17-28.

13. A. Appel, "Some Techniques for Shading Machine Renderings of Solids," Proc. SJCC 1968, Thompson Books, Washington, DC, 1968, pp. 39-45.

14. P. Atherton, K. Werten, and D. Greenberg, "Polygon Shadow Generation," Computer Graphics (Proc. SIGG12APH 78), Vol. 12, No. 3, July 1978, pp. 275-281.

15. F. Crow, "Shadow Algorithms for Computer Graphics," Computer Graphics, (Proc. SIGGRAPH 77), Vol. 11, No. 3, July 1977, pp. 242-248.

16. L. Williams, "Casting Curved Shadows on Curved Surfaces," Computer Graphics, (Proc. SIGGRAPH 78), Vol. 12, No. 3, July 1978, pp. 270-274.

17. T. Whitted, "An Improved Illumination Model for Shaded Display," Comm. ACM, Vol. 23, No. 6, June 1980, pp. 343-349.

APPENDIX A: APL GRAPHPAK MODELING

Programs, Data and Examples of Output

Program Description

HIDDEN

[1] problem space window x_{LL} y_{LL} x_{UR} y_{UR}
lower left corner, upper right corner

[2] number of planes

[3] Index origin = 1

[4] map problem space into the scaling viewport

[5] scales data to fit between -1 and 1

[6] rotate the object through angles θ_x, θ_y and θ_z The angle is measured in degrees, and the rotation is counterclockwise.

[7] create a perspective projection of an object onto the X,Y plane, from a vantage point on the Z-axis, perpendicular to the plane of the paper.

[8] perform a transformation from the scaling viewport to a window in problem space.

[9] comment

[11] I = 1

[12] initialize matrix "judge"

[13] create the sequence of integer numbers from rowse [I,1] to rowse [I,2].

[14] Is this plane visible?

[15] increment I

[16] If I < planeno THEN GO TO [13]
ELSE CONTINUE

[17] comment

[18] I = 1

[19] label

[20] skip the invisible plane

[21] plane number

[22] vertices numbers of the plane

[23] fill with a color

[24] draw the picture edges

[25] comment

[26] increment index I

[27] If I < planeno THEN GO TO [19]

ELSE EXIT

[28] comment

VISIBLE

[1] $\vec{u}$

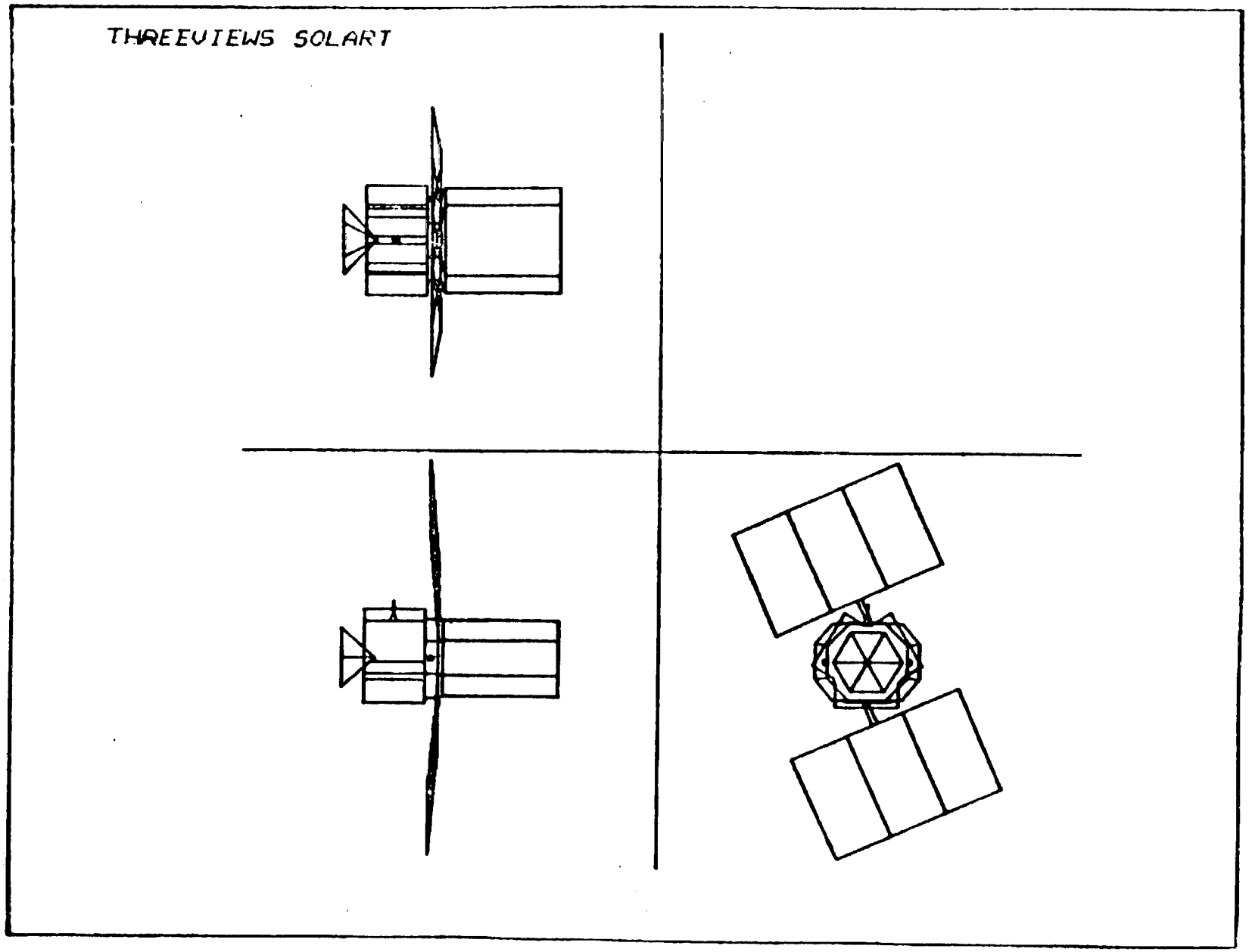
[2] $\vec{v}$

[3] $(\vec{u} \times \vec{v})$ $\vec{k} > 0$? visible θ test

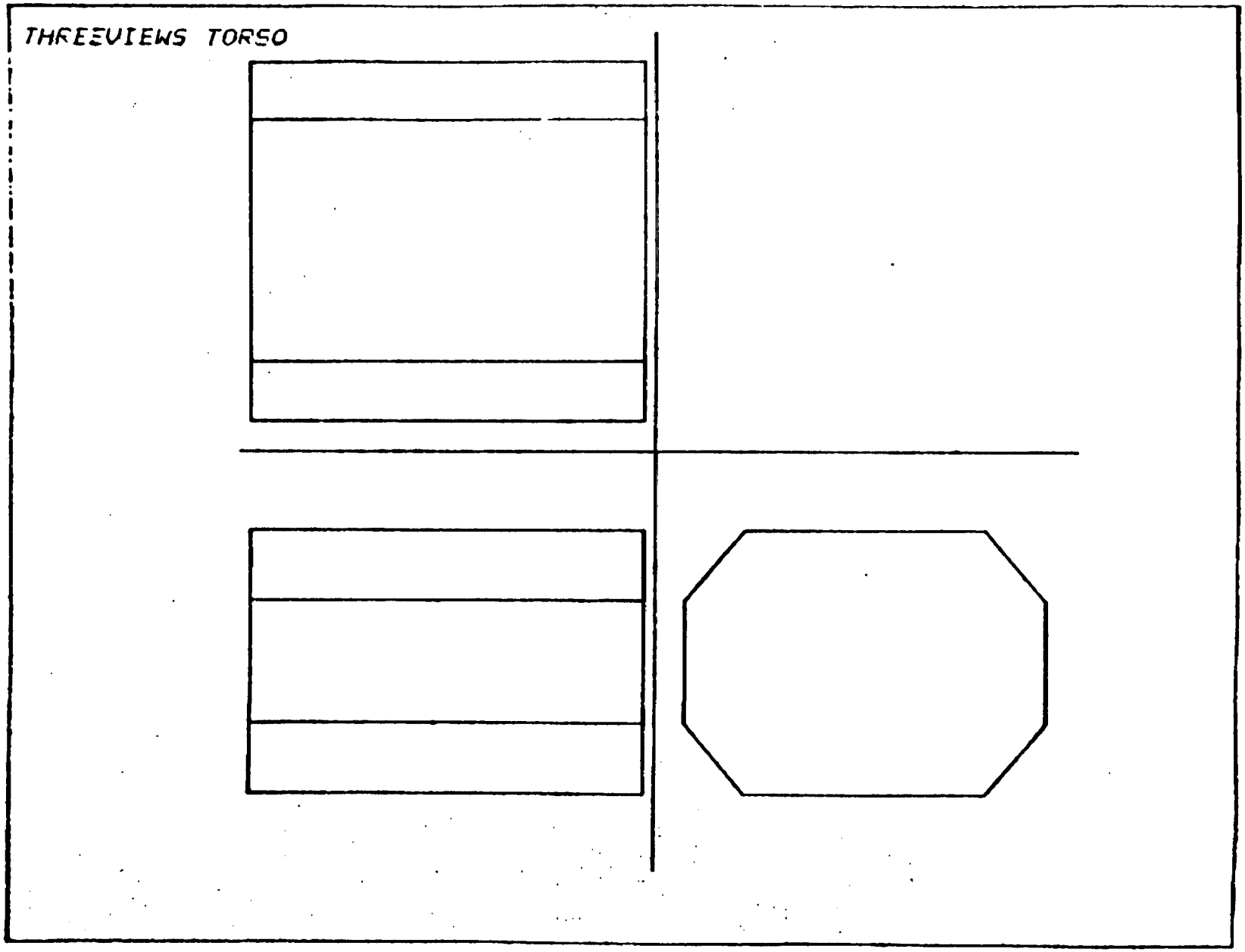
[4] comment

[5] Z coordinate of the last vertex of the plane

[6] export the visible information and maximum Z coordinate of the plane



Threeviews of Satellite.



Threeviews of Module TORSO

Computer Programs and Data

Hidden Line Removal Function

```
      ∇ HIDDEN [⎕] ∇
      ∇ ROT HIDDEN OBJ;DS;OBJDOWN;N;I;ROWR;RT;R;T;J;JUDGE
[1]   ⎕IO←1
[2]   DS←⎕ INTO SVE
[3]   OBJDOWN←¯1 1 SCALE OBJ
[4]   R←ROT ROTATE OBJDOWN
[5]   RT←C[2] PERSPECTIVE R
[6]   T←DS XFM RT[;1 2 3]
[7]   A
[8]   N←⌊900
[9]   I←1
[10]  JUDGE←(PLANENO,3)⍴0
[11] EVAL:ROWR←((ROWSE[I;1]≤N)∧(ROWSE[I;2]≥N))/⍳N
[12]  JUDGE[I;]←(VISIBLE R[ROWR;]),I
[13]  I←I+1
[14]  →(I≤PLANENO)/EVAL
[15] A     JUDGE←JUDGE[(⍋JUDGE[;2]);]
[16]  I←1
[17] PLOTS:
[18]  →(0=JUDGE[I;1])/NEXT
[19]  J←JUDGE[I;3]
[20]  ROWR←((ROWSE[J;1]≤⍳N)∧(ROWSE[J;2]≥⍳N))/⍳N
[21]  FILL T[ROWR;]
[22]  DRAW T[ROWR;] USING COLOR C[1]
[23] A    ←⎕
[24] NEXT:I←I+1
[25]  →(I≤PLANENO)/PLOTS
[26] A
      ∇
```

```
      ∇ VISIBLE [⎕] ∇
      ∇ YN←VISIBLE P;CR1;CR2;VIS;ZZ
[1]   CR1←P[2;1 2 3]-P[1;1 2 3]
[2]   CR2←P[3;1 2 3]-P[1;1 2 3]
[3]   VIS←((CR1[1]×CR2[2])-CR1[2]×CR2[1])≥0
[4]   A
[5]   ZZ←P[(⍋P[;1]);4]
[6]   YN←VIS,ZZ
      ∇
```

Data Transmit Function

```
      ∇ IN [0] ∇
      ∇ AFLV←TSOF IN COL;AFLS;AFLA;ROW
[1]   AFLS←GETFILE TSOF
[2]   AFLA←REPNS AFLS
[3]   ROW←⍴AFLA
[4]   AFLV←(ROW[1],COL)⍴⍷,AFLA
      ∇


      ∇ GETFILE [0] ∇
      ∇ A←GETFILE B;C;D;E;CC
[1]   C←'C'
[2]   TSO 'FREE DD(C)'
[3]   'C' ALLOCATE B
[4]   C←C,' (192 R '
[5]   CC←C,'CTL'
[6]   →(2^.=A←111 ⎕SVO D← 2 2 ⍴'C CC')/L1
[7]   'SHARE FAILURE'
[8]   →0
[9]   L1:→(0∨.≠A←C,CC)/L4
[10]  A← 0 0 ⍴' '
[11]  L2:→(0=⍴E←C)/L4
[12]  →(0≠CC)/L3
[13]  A←A VCAT E
[14]  →L2
[15]  L3:'READ ERROR.  RETURN CODE: ',⍕CC
[16]  L4:⎕SVR D
      ∇


      ∇ REPNS [0] ∇
      ∇ M←REPNS M0;IS;I;J
[1]   M←M0
[2]   IS←⍴M
[3]   I←1
[4]   LP1:J←1
[5]   LP2:→SK1×⍳M[I;J]≠'-'
[6]   M[I;J]←'¯'
[7]   SK1:→SK×⍳M[I;J]≠'+'
[8]   M[I;J]←'0'
[9]   SK:→LP2×⍳IS[2]≥J←J+1
[10]  →LP1×⍳IS[1]≥I←I+1
      ∇
```

```
      ∇ OUT [□] ∇
      ∇ AFLV OUT TSOF;DIM;BLK;RDY
[1]   AFLVA←,AFLV
[2]   DIM←⍴AFLVA
[3]   BLK←(DIM[1],80-DIM[2])⍴' '
[4]   RDY←(DIM[1],80)⍴AFLVA,[2] BLK
[5]   RDY PUTFILE TSOF
      ∇


      ∇ PUTFILE [□] ∇
      ∇ A←B PUTFILE C;F;FC;I;N;R;S;⎕IO
[1]   ⎕IO←1
[2]   B←(¯2↑ 1 1 ,⍴B)⍴B
[3]   R←TSO 'FREE DD(CARDS424)'
[4]   R←TSO 'FREE DD(F)'
[5]   R←TSO 'ATTRIB CARDS424 RECFM(F B) LRECL(80) BLKSIZE(4240)'
[6]   D←' NEW '
[7]   K←' CATALOG '
[8]   C←' BCD '
[9]   S←' SPACE(10,50) BLKSIZE(4240) '
[10]  A←' W '
[11]  U←' (CARDS424) '
[12]  E←111
[13]  D←'()' PREPARSE C
[14]  →(4)1↑⍴D)/LA
[15]  →('()'∨.≠D[2 4 ;1])/LA
[16]  N←(N≠' ')/N←,D[⍳4;]
[17]  D← 5 0 ↓D
[18]  →LB
[19] LA:N←(N≠' ')/N←D[1;]
[20]  D← 1 0 ↓D
[21] L0:→(0=1↑⍴D← 1 0 ↓D)/L4
[22] LB:→(0∈⍴T←OBLANKS(OPTSALCD MEMBER D)≠OPTSALCD)/L10
[23]  D←' ',T,' '
[24] L10:→(0∈⍴T←OBLANKS(OPTSALCK MEMBER D)≠OPTSALCK)/L1
[25]  K←' ',T,' '
[26] L1:→(0∈⍴T←OBLANKS(OPTSCONV MEMBER D)≠OPTSCONV)/L2
[27]  C←' ',T,' '
[28] L2:→(0='USING' MEMBER D)/L3
[29]  U←((T≠' ')/T←,D[((⍳3)+~⎕IO)+D NAMEINDEX 'USING';]),' '
[30] L3:→(0='SPACE' MEMBER D)/L4
[31]  T← 1 2 3 +D NAMEINDEX 'SPACE'
[32]  →('()'∨.≠(W≠' ')/W←,D[(1↑T),2↓T;])/L7
[33]  S←' ',('SPACE',(W≠' ')/W←,D[T;]),' ',OBLANKS D[1+¯1↑T;]
[34]  →(')('∨.≠(¯1↑W),1↑W←(W≠' ')/W←,D[4+T;])/L7
[35]  S←S,' ',W,' '
```

```
[36] L4:→(0=A←TSO 'ALLOCATE DD(F) DA(',N,') ',D,K,S,' USING',U)/L5
[37]   →(0=A←TSO 'ALLOCATE DD(F) DA(',N,') OLD KEEP')/L5
[38]   'ALLOCATE FAILURE,  CODE=',⍕A
[39]   →0
[40] L5:⍎'F←''F(',C,A,''''
[41]   ⍎'FC←''',F,' CTL'''
[42]   →(2^.=E ⎕SVO S← 2 2 ⍴'F FC')/L6
[43]   'SHARE FAILURE, '
[44]   ⋄
[45] L6:→(0^.=T←F,FC)/L7
[46]   'OPEN FAILURE: ',⍕T
[47]   ⋄
[48] L7:N←1↑⍴B
[49]   I←0
[50] L8:→(N<I←I+1)/L9
[51]   F←B[I;]
[52]   →(0=R←FC)/L8
[53]   'WRITE ERROR: ',⍕R
[54]   ⋄
[55] L9:R←⎕SVR S
     ∇



     ∇ TSO [⎕] ∇
     ∇ C←TSO A;B;⎕IO;D
[1]    ⎕IO←0
[2]    →(2=C←⍎ECMD ⎕SVO 'B')/L1
[3]    'TSO SHARE OFFER FAILURE'
[4]    →0
[5]  L1:D←D←A
[6]    →((0≠TYPE C),(1≠⍴,C),1≠⍴⍴C←B)/0
[7]    →(0=C)/0
[8]    →(999<C)/L2
[9]    'TSO RETURN CODE: ',(⍕C),','
[10]   →L3
[11] L2:'TSO ABEND CODE: ','0123456789ABCDEFX'[3↑(6⍴16)⊤C]
[12] L3:'COMMAND: ',⍕D
     ∇
```

Pickup Function

```
      ∇ PICKUP [⎕] ∇
      ∇ NEW←ORDER PICKUP ORG;N;I
[1]   N←⍴ORDER[;1]
[2]   NEW←(N,4)⍴0
[3]   I←1
[4]  RP:NEW[I;]←ORDER[I;1],ORG[ORDER[I;2];2 3 4]
[5]   →(N≥I←I+1)/RP
      ∇
```

```
      OTORSO
   0 14
   1 17
   1  4
   1  3
   1 14
   0 17
   1 20
   1  5
   1  4
   1 17
   0 20
   1 23
   1  6
   1  5
   1 20
   0 23
   1 26
   1  7
   1  6
   1 23
   0 26
   1 29
   1  8
   1  7
   1 26
   0 29
   1 10
   1  1
   1  8
   1 29
   0 10
   1 11
   1  2
   1  1
   1 10
   0 11
   1 14
   1  3
   1  2
   1 11
```

	OTOPSO PICKUP	TOPSO	
0	10.575	-2.4449	2.2383
1	10.575	-2.4449	-2.2383
1	3.175	-2.4449	-2.2383
1	3.175	-2.4449	2.2383
1	10.575	-2.4449	2.2383
0	10.575	-2.4449	-2.2383
1	10.575	-1.1266	-3.3375
1	3.175	-1.1266	-3.3375
1	3.175	-2.4449	-2.2383
1	10.575	-2.4449	-2.2383
0	10.575	-1.1266	-3.3375
1	10.575	1.1266	-3.3375
1	3.175	1.1266	-3.3375
1	3.175	-1.1266	-3.3375
1	10.575	-1.1266	-3.3375
0	10.575	1.1266	-3.3375
1	10.575	2.4449	-2.2383
1	3.175	2.4449	-2.2383
1	3.175	1.1266	-3.3375
1	10.575	1.1266	-3.3375
0	10.575	2.4449	-2.2383
1	10.575	2.4449	2.2383
1	3.175	2.4449	2.2383
1	3.175	2.4449	-2.2383
1	10.575	2.4449	-2.2383
0	10.575	2.4449	2.2383
1	10.575	1.1266	3.3375
1	3.175	1.1266	3.3375
1	3.175	2.4449	2.2383
1	10.575	2.4449	2.2383
0	10.575	1.1266	3.3375
1	10.575	-1.1266	3.3375
1	3.175	-1.1266	3.3375
1	3.175	1.1266	3.3375
1	10.575	1.1266	3.3375
0	10.575	-1.1266	3.3375
1	10.575	-2.4449	2.2383
1	3.175	-2.4449	2.2383
1	3.175	-1.1266	3.3375
1	10.575	-1.1266	3.3375

Graphics Analysis Function

```
      ∇ ANALYZE[⎕] ∇
      ∇ ANALYZE OBJ;N;I;J;B;TEMP
[1]   W← ¯1 ¯1 1 1
[2]   OBJ← ¯0.95 0.95 SCALE OBJ
[3]   RETICLE
[4]   SKETCH OBJ
[5]   B←⎕
[6]   →(B='Q')/0
[7]   RETICLE
[8]   N←⍴OBJ[;1]
[9]   I←1
[10] REP:J←I+1
[11]  TEMP← 2 4 ⍴OBJ[I;],OBJ[J;]
[12]  TEMP[1;1]←0
[13]  SKETCH TEMP
[14]  B←⎕
[15]  →(B='Q')/0
[16]  I←J
[17]  →(I<N)/REP
      ∇
```

```
      ∇ THREEVIEWS [⎕] ∇
      ∇ THREEVIEWS P;W
[1]   ⍝ ORTHOGRAPHIC PROJECTIONS
[2]   W← ¯2 ¯2 2 2
[3]   P← ¯0.95 0.95 SCALE P
[4]   ⍝ FRONT VIEW
[5]   SKETCH ¯1 ¯1 0 TRANSLATE P
[6]   ⍝ SIDE VIEW
[7]   SKETCH 1 ¯1 0 TRANSLATE 0 ¯90 0 ROTATE P
[8]   ⍝ TOP VIEW
[9]   SKETCH ¯1 1 0 TRANSLATE 90 0 0 ROTATE P
      ∇
```

APL Function "visible"

```
      ∇ VISIBLE [⎕] ∇
      ∇ YN←VISIBLE P;SUM;I;VIS;ZZ
[1]   I← 2 3 4 1
[2]   SUM←I CRXDOT P
[3]   I← 3 4 2 0
[4]   SUM←(I CRXDOT P)+SUM
[5]   I← 2 4 3 0
[6]   SUM←SUM-(I CRXDOT P)
[7]   VIS←SUM>0
[8]   ZZ←P[(⍴P[;1]);4]
[9]   YN←VIS,ZZ
      ∇
```

```
      ∇ CRXDOT [⎕] ∇
      ∇ SUM←I CRXDOT P;CR1;CR2;VW
[1]   CR1←P[2;(I[1],I[2])]-P[1;(I[1],I[2])]
[2]   CR2←P[3;(I[1],I[2])]-P[1;(I[1],I[2])]
[3]   VW←0
[4]   →(0=I[4])/CON
[5]   VW←C[2]
[6]  CON:
[7]   SUM←(((CR1[1]×CR2[2])-CR1[2]×CR2[1]))×(VW-P[1;I[3]])
      ∇
```

The Modified Function "threeviews", including an extra oblique projection

```
      ∇ THREEVIEWS [□] ∇
      ∇ THREEVIEWS P;W
[1]   ⍝ ORTHOGRAPHIC PROJECTIONS
[2]    W← ¯2 ¯2 2 2
[3]    P← ¯0.93 0.95 SCALE P
[4]   ⍝ FRONT VIEW
[5]    SKETCH ¯1 ¯1 0 TRANSLATE P
[6]   ⍝ SIDE VIEW
[7]    SKETCH 1 ¯1 0 TRANSLATE 0 ¯90 0 ROTATE P
[8]   ⍝ TOP VIEW
[9]    SKETCH ¯1 1 0 TRANSLATE 90 0 0 ROTATE P
[10]   SKETCH 1.3 1.3 0 TRANSLATE 10 10 10 ROTATE P
[11]   RETICLE
      ∇
```

```
      TEST
  0 60  0  0
  1  0  0  0
  1  0 40  0
  1  0  0  0
  1  0  0 20
```

The Attitude and Path Data

```
   TRACE
0     0     0   130  -130  -200
0    10     0    98   -98   -80
0    20     0    70   -85   -20
0    40     0    40   -75     0
0    60     0    20   -50    20
0    75     0    10   -25    30
0    90     0     0     0    40
```

```
   TRACEAX
0  -130   130   200
1   -98    98    80
1   -70    85    20
1   -40    75     0
1   -20    50   -20
1   -10    25   -30
1     0     0   -40
0     0     0     0
1     0     0     0
```

```
   C
2 100
```

```
   DTOR
0.017453292
```

Function "motion" for Fixed Vision System

```
      ∇ MOTION[⎕]∇
      ∇ MOTION TRACE;ROW;I;ROT;TRAN
[1]   W← ¯30 ¯100 100 30
[2]   PLANENO←⍴ROWSE[;1]
[3]   ⎕IO←1
[4]   DS←W INTO SVE
[5]   ROW←⍴TRACE[;1]
[6]   I←0
[7]  REP:
[8]   I←I+1
[9]   ROT←TRACE[I; 1 2 3]
[10]  TRAN←TRACE[I; 4 5 6]
[11]  ROT HID TRAN
[12]  →(I<ROW)/REP
      ∇

      ∇ HID[⎕]∇
      ∇ ROT HID TRAN;OBJDOWN;N;I;RT;R;J;JUDGE
[1]   A    OBJDOWN← ¯1 1 SCALE OBJ
[2]    OBJDOWN←OBJ
[3]    R←TRAN TRANSLATE ROT ROTATE OBJDOWN
[4]    RT←C[2] PERSPECTIVE R
[5]    T←DS XFM RT[; 1 2 3]
[6]   A  T←RT[; 1 2 3]
[7]   A
[8]    N←\900
[9]    I←1
[10]   JUDGE←(PLANENO,3)⍴0
[11] EVAL:ROWR←((ROWSE[I;1]≤N)∧(ROWSE[I;2]≥N))/N
[12]   JUDGE[I;]←(VISIBLE R[ROWR;]),I
[13]   I←I+1
[14]   →(I≤PLANENO)/EVAL
[15]  A     JUDGE←JUDGE[(⍋JUDGE[;2]);]
[16]   I←1
[17] PLOTS:
[18]   →(0≠JUDGE[I;1])/NEXT
[19]   J←JUDGE[I;3]
[20]   ROWR←((ROWSE[J;1]≤N)∧(ROWSE[J;2]≥N))/N
[21]   FILL T[ROWR;]
[22]   DRAW T[ROWR;] USING COLOR C[1]
[23]  A    ←⎕
[24] NEXT:I←I+1
[25]   →(I≤PLANENO)/PLOTS
[26]  A
      ∇
```

The HELIX Path Data Set

```
     HELIX
300 -200 -340
270 -180 -300
240 -160 -260
210 -140 -220
180 -120 -180
150 -100 -140
120  -80 -100
 90  -60  -60
 60  -40  -20
 30  -20   20
  0    0   60
```

```
     HELIXAX
0.000000000E0   -2.199999406E2    2.000000000E2    3.810512120E2
1.000000000E0    5.615387607E-5   1.800000000E2    4.000000000E2
1.000000000E0    1.8000000389E2   1.600000000E2    3.1176912229E2
1.000000000E0    2.771281467E2    1.400000000E2    1.5999999697E2
1.000000000E0    2.800000000E2    1.200000000E2   -2.620514218E-5
1.000000000E0    2.078460875E2    1.000000000E2   -1.200000162E2
1.000000000E0    9.999998919E1    8.000000000E1   -1.732050870E2
1.000000000E0   -7.487183495E-6   6.000000000E1   -1.600000000E2
1.000000000E0   -6.000000324E1    4.000000000E1   -1.039230466E2
1.000000000E0   -6.928203293E1    2.000000000E1   -3.999999892E1
1.000000000E0   -4.000000000E1    0.000000000E0    0.000000000E0
0.000000000E0    0.000000000E0    0.000000000E0    0.000000000E0
1.000000000E0    0.000000000E0    0.000000000E0    0.000000000E0
```

Function ROBOT of Tracking Vision System

```
      ▽ ROBOT[□] ▽
      ▽ ROBOT HELIX;ROW;I;ROT;TRAN;R
[1]   W← ¯30 ¯30 30 30
[2]   A  BOXCLS← 5 3 ⍴ 0 50 50 1 ¯50 50 1 ¯50 ¯50 1 50 ¯50 1 50 50
[3]    PLANENO←⍴ROWSE[;1]
[4]    □IO←1
[5]    DS←W INTO SVP
[6]    ROW←⍴HELIX[;1]
[7]    I←0
[8]   REP:
[9]    I←I+1
[10]   R←C[2]-HELIX[I;3]
[11]   ROT←¯57.29578×(¯3○(HELIX[I;2]÷R)),HELIX[I;1],0
[12]   TRAN←0,0,HELIX[I;3]
[13]   ROT HIDHEL TRAN
[14]   ''
[15]   □
[16]  A  FILL(W INTO SVP) XFM BOXCLS
[17]   ERASE
[18]   →(I<ROW)/REP
      ▽

      ▽HIDHEL[□] ▽
      ▽ ROT HIDHEL TRAN;OBJDOWN;N;I;RT;R;J;JUDGE
[1]   A   OBJDOWN← ¯1 1 SCALE OBJ
[2]    OBJDOWN←OBJ
[3]    OBJDOWN←(0,ROT[2],0) ROTATE OBJDOWN
[4]    OBJDOWN←(ROT[1],0,0) ROTATE OBJDOWN
[5]   A    R←(TRAN[1],TRAN[2],0) TRANSLATE OBJDOWN
[6]    R←TRAN TRANSLATE OBJDOWN
[7]    RT←C[2] PERSPECTIVE R
[8]   A  RT← 0 0 0 TRANSLATE RT
[9]    T←DS XFM RT[; 1 2 3]
[10]  A  T←RT[; 1 2 3]
[11]  A
[12]   N←⍳900
[13]   I←1
[14]   JUDGE←(PLANENO,3)⍴0
[15]  EVAL:ROWR←((ROWSE[I;1]≤N)∧(ROWSE[I;2]≥N))/N
[16]   JUDGE[I;]←(VISIBLE R[ROWR;]),I
[17]   I←I+1
[18]   →(I≤PLANENO)/EVAL
[19]  A     JUDGE←JUDGE[(⍋JUDGE[;2]);]
[20]   I←1
[21]  PLOTS:
[22]   →(0≠JUDGE[I;1])/NEXT
[23]   J←JUDGE[I;3]
[24]   ROWR←((ROWSE[J;1]≤N)∧(ROWSE[J;2]≥N))/N
[25]   FILL T[ROWR;]
[26]   DRAW T[ROWR;] USING COLOR C[1]
[27]  A    ←□
[28]  NEXT:I←I+1
[29]   →(I≤PLANENO)/PLOTS
[30]  A
      ▽
```

Original Data Set Solarmax.vec

Provided by NASA/JSC

```
#
#
  266
    0      3.1750      1.1266      3.3375
    1      3.1750     -1.1266      3.3375
    1      3.1750     -2.4449      2.2383
    1      3.1750     -2.4449     -2.2383
    1      3.1750     -1.1266     -3.3375
    1      3.1750      1.1266     -3.3375
    1      3.1750      2.4449     -2.2383
    1      3.1750      2.4449      2.2383
    1      3.1750      1.1266      3.3375
    1     10.5750      1.1266      3.3375
    1     10.5750     -1.1266      3.3375
    1      3.1750     -1.1266      3.3375
    0     10.5750     -1.1266      3.3375
    1     10.5750     -2.4449      2.2383
    1      3.1750     -2.4449      2.2383
    0     10.5750     -2.4449      2.2383
    1     10.5750     -2.4449     -2.2383
    1      3.1750     -2.4449     -2.2383
    0     10.5750     -2.4449     -2.2383
    1     10.5750     -1.1266     -3.3375
    1      3.1750     -1.1266     -3.3375
    0     10.5750     -1.1266     -3.3375
    1     10.5750      1.1266     -3.3375
    1      3.1750      1.1266     -3.3375
    0     10.5750      1.1266     -3.3375
    1     10.5750      2.4449     -2.2383
    1      3.1750      2.4449     -2.2383
    0     10.5750      2.4449     -2.2383
    1     10.5750      2.4449      2.2383
    1      3.1750      2.4449      2.2383
    0     10.5750      2.4449      2.2383
    1     10.5750      1.1266      3.3375
    0      1.9583     -0.2658      3.4825
    1      1.9583     -1.0158      2.2150
    1      1.9583     -1.4025      1.9925
    1      1.9583     -2.8774      1.9925
    1      1.9583     -2.8774     -1.9925
    1      1.9583     -1.4025     -1.9925
    1      1.9583     -1.0158     -2.2150
    1      1.9583     -0.2658     -3.4825
    1      1.9583      3.1442     -1.5000
    1      1.9583      2.4183     -0.2417
    1      1.9583      2.4183      0.2417
    1      1.9583      3.1442      1.5000
    1      1.9583     -0.2658      3.4825
    1     -1.9583     -0.2658      3.4825
    1     -1.9583     -1.0158      2.2150
    1      1.9583     -1.0158      2.2150
    0     -1.9583     -1.0158      2.2150
    1     -1.9583     -1.4025      1.9925
    1      1.9583     -1.4025      1.9925
    0     -1.9583     -1.4025      1.9925
    1     -1.9583     -2.8774      1.9925
```

1	1.9583	-2.8774	1.9925
0	-1.9583	-2.8774	1.9925
1	-1.9583	-2.8774	-1.9925
1	1.9583	-2.8774	-1.9925
0	-1.9583	-2.8774	-1.9925
1	-1.9583	-1.4025	-1.9925
1	1.9583	-1.4025	-1.9925
0	-1.9583	-1.4025	-1.9925
1	-1.9583	-1.0158	-2.2150
1	1.9583	-1.0158	-2.2150
0	-1.9583	-1.0158	-2.2150
1	-1.9583	-0.2658	-3.4825
1	1.9583	-0.2658	-3.4825
0	-1.9583	-0.2658	-3.4825
1	-1.9583	3.1442	-1.5000
1	1.9583	3.1442	-1.5000
0	-1.9583	3.1442	-1.5000
1	-1.9583	2.4183	-0.2417
1	1.9583	2.4183	-0.2417
0	-1.9583	2.4183	-0.2417
1	-1.9583	2.4183	0.2417
1	1.9583	2.4183	0.2417
0	-1.9583	2.4183	0.2417
1	-1.9583	3.1442	1.5000
1	1.9583	3.1442	1.5000
0	-1.9583	3.1442	1.5000
1	-1.9583	-0.2658	3.4825
0	2.8233	-6.3191	4.7183
1	2.3267	-12.8115	1.8275
1	2.3267	-8.1265	-8.6941
1	2.8233	-1.6342	-5.8032
1	2.8233	-6.3191	4.7183
0	2.8233	-4.7574	1.2108
1	2.3267	-11.2500	-1.6800
0	2.3267	-9.6882	-5.1866
1	2.8233	-3.1958	-2.2958
0	2.8233	-3.9091	-0.6950
1	2.9166	-2.7583	-0.1667
1	2.9166	-2.7583	0.1667
1	2.8233	-4.0441	-0.3900
0	2.8233	1.6342	5.8032
1	2.8233	6.3191	-4.7183
1	2.3267	12.8115	-1.8275
1	2.3267	8.1265	8.6941
1	2.8233	1.6342	5.8032
0	2.8233	3.1958	2.2958
1	2.3267	9.6882	5.1866
0	2.3267	11.2500	1.6800
1	2.8233	4.7574	-1.2108
0	2.8233	4.0441	0.3900
1	2.9166	2.7583	-0.1667
1	2.9166	2.7583	0.1667
1	2.8233	3.9091	0.6950
0	-3.4583	0.0000	2.2083
1	-2.4166	0.0000	0.0000
1	-3.4583	1.9125	1.1042

1	-3.4583	1.9125	-1.1042
1	-2.4166	0.0000	0.0000
1	-3.4583	0.0000	-2.2083
1	-3.4583	-1.9125	-1.1042
1	-2.4166	0.0000	0.0000
1	-3.4583	-1.9125	1.1042
1	-2.2083	0.0000	0.0000
1	-3.4583	0.0000	2.2083
1	-3.4583	1.9125	1.1042
1	-2.2083	0.0000	0.0000
1	-3.4583	1.9125	-1.1042
1	-3.4583	0.0000	-2.2083
1	-2.2083	0.0000	0.0000
1	-3.4583	-1.9125	-1.1042
1	-3.4583	-1.9125	1.1042
1	-3.4583	0.0000	2.2083
1	-1.2333	0.0000	0.0000
1	-3.4583	1.9125	1.1042
0	-1.2333	0.0000	0.0000
1	-3.4583	1.9125	-1.1042
0	-1.2333	0.0000	0.0000
1	-3.4583	0.0000	-2.2083
0	-1.2333	0.0000	0.0000
1	-3.4583	-1.9125	-1.1042
0	-1.2333	0.0000	0.0000
1	-3.4583	-1.9125	1.1042
0	-1.2333	0.0000	0.0000
1	-2.2083	0.0000	0.0000
0	-3.4583	0.0208	0.0000
1	-2.3750	0.0208	0.0000
1	-2.3750	0.0000	0.0208
1	-3.4583	0.0000	0.0208
1	-3.4583	-0.0208	0.0000
1	-2.3750	-0.0208	0.0000
1	-2.3750	0.0000	-0.0208
1	-3.4583	0.0000	-0.0208
1	-3.4583	0.0208	0.0000
1	-3.4583	0.0000	0.0208
0	-3.4583	-0.0208	0.0000
1	-3.4583	0.0000	-0.0208
0	-2.3750	0.0000	-0.0208
1	-2.3750	0.0208	0.0000
0	-2.3750	0.0000	0.0208
1	-2.3750	0.0000	-0.0208
0	2.8083	1.0591	2.5575
1	1.9583	1.0591	2.5575
1	1.9583	-1.0591	2.5575
1	2.8083	-1.0591	2.5575
1	2.8083	1.0591	2.5575
1	3.1750	1.1266	3.3375
1	3.1750	-1.1266	3.3375
1	2.8083	-1.0591	2.5575
0	1.9583	-1.0591	2.5575
1	1.9583	-2.5575	1.0591
1	2.8083	-2.5575	1.0591
1	2.8083	-1.0591	2.5575

0	3.1750	-1.1266	3.3375
1	3.1750	-2.4449	2.2383
1	2.8083	-2.5575	1.0591
0	1.9583	-2.5575	1.0591
1	1.9583	-2.5575	-1.0591
1	2.8083	-2.5575	-1.0591
1	2.8083	-2.5575	1.0591
0	3.1750	-2.4449	2.2383
1	3.1750	-2.4449	-2.2383
1	2.8083	-2.5575	-1.0591
0	1.9583	-2.5575	-1.0591
1	1.9583	-1.0591	-2.5575
1	2.8083	-1.0591	-2.5575
1	2.8083	-2.5575	-1.0591
0	3.1750	-2.4449	-2.2383
1	3.1750	-1.1266	-3.3375
1	2.8083	-1.0591	-2.5575
0	1.9583	-1.0591	-2.5575
1	1.9583	1.0591	-2.5575
1	2.8083	1.0591	-2.5575
1	2.8083	-1.0591	-2.5575
0	3.1750	-1.1266	-3.3375
1	3.1750	1.1266	-3.3375
1	2.8083	1.0591	-2.5575
0	1.9583	1.0591	-2.5575
1	1.9583	2.5575	-1.0591
1	2.8083	2.5575	-1.0591
1	2.8083	1.0591	-2.5575
0	3.1750	1.1266	-3.3375
1	3.1750	2.4449	-2.2383
1	2.8083	2.5575	-1.0591
0	1.9583	2.5575	-1.0591
1	1.9583	2.5575	1.0591
1	2.8083	2.5575	1.0591
1	2.8083	2.5575	-1.0591
0	3.1750	2.4449	-2.2383
1	3.1750	2.4449	2.2383
1	2.8083	2.5575	1.0591
0	1.9583	2.5575	1.0591
1	1.9583	1.0591	2.5575
0	2.8083	1.0591	2.5575
1	2.8083	2.5575	1.0591
0	3.1750	2.4449	2.2383
1	3.1750	1.1266	3.3375
0	0.1400	2.4183	0.2417
1	0.1400	2.4183	-0.2417
1	-0.2792	2.4183	0.0000
1	0.1400	2.4183	0.2417
1	0.0000	2.9166	0.0000
1	0.1400	2.4183	-0.2417
0	-0.2792	2.4183	0.0000
1	0.0000	2.9166	0.0000
1	0.0000	3.7333	0.0000
0	2.3833	0.1500	2.7800
1	2.5133	0.0750	2.7800
1	2.5133	-0.0750	2.7800

1	2.3833	-0.1500	2.7800
1	2.2533	-0.0750	2.7800
1	2.2533	0.0750	2.7800
1	2.3833	0.1500	2.7800
1	2.3833	0.1500	2.5575
1	2.5133	0.0750	2.5575
1	2.5133	0.0750	2.7800
0	2.5133	0.0750	2.5575
1	2.5133	-0.0750	2.5575
1	2.5133	-0.0750	2.7800
0	2.5133	-0.0750	2.5575
1	2.3833	-0.1500	2.5575
1	2.3833	-0.1500	2.7800
0	2.3833	-0.1500	2.5575
1	2.2533	-0.0750	2.5575
1	2.2533	-0.0750	2.7800
0	2.2533	-0.0750	2.5575
1	2.2533	0.0750	2.5575
1	2.2533	0.0750	2.7800
0	2.2533	0.0750	2.5575
1	2.3833	0.1500	2.5575
0	2.3833	0.1500	-2.7800
1	2.5133	0.0750	-2.7800
1	2.5133	-0.0750	-2.7800
1	2.3833	-0.1500	-2.7800
1	2.2533	-0.0750	-2.7800
1	2.2533	0.0750	-2.7800
1	2.3833	0.1500	-2.7800
1	2.3833	0.1500	-2.5575
1	2.5133	0.0750	-2.5575
1	2.5133	0.0750	-2.7800
0	2.5133	0.0750	-2.5575
1	2.5133	-0.0750	-2.5575
1	2.5133	-0.0750	-2.7800
0	2.5133	-0.0750	-2.5575
1	2.3833	-0.1500	-2.5575
1	2.3833	-0.1500	-2.7800
0	2.3833	-0.1500	-2.5575
1	2.2533	-0.0750	-2.5575
1	2.2533	-0.0750	-2.7800
0	2.2533	-0.0750	-2.5575
1	2.2533	0.0750	-2.5575
1	2.2533	0.0750	-2.7800
0	2.2533	0.0750	-2.5575
1	2.3833	0.1500	-2.5575

APPENDIX B: FORTAN PROGRAM TO TRANSFER SOLARMAX DATA FROM IBM 370 TO CELERITY C1200

FORTRAN program trans.f


```
      program main
      implicit real*8(a-h,o-z)
      dimension x(500),y(500),z(500),ibf(500),node(50)
      open (2,file='s.n')
      open (1,file='s.s')
      k = 1
      i = 1
   10 read (1,*,err=30,end=30) igar, gar, x(i), y(i), z(i)
      do 20 j=1,k
         if ((x(i) .eq. x(j)) .and. (y(i) .eq. y(j))
     .                        .and. (z(i) .eq. z(j))) then
            if (i .eq. 1) then
               xs = x(1)
               xl = x(1)
               ys = y(1)
               yl = y(1)
               zs = z(1)
               zl = z(1)
               ibf(1) = 1
            endif
            ibf(i) = j
            go to 25
         endif
   20 continue
      k = k+1
      ibf(i) = k
      x(k) = x(i)
      y(k) = y(i)
      z(k) = z(i)
      if (x(k) .lt. xs)  xs = x(k)
      if (x(k) .gt. xl)  xl = x(k)
      if (y(k) .lt. ys)  ys = y(k)
      if (y(k) .gt. yl)  yl = y(k)
      if (z(k) .lt. zs)  zs = z(k)
      if (z(k) .gt. zl)  zl = z(k)
   25 i = i+1
      go to 10
   30 write (*,50) xs, xl, ys, yl, zs, zl
   50 format(6g13.6)
      do 60 j = 1,k
         write (*,50) x(j), y(j), z(j)
   60 continue
      kv = k
      k = 0
   70 read (2,*,err=99,end=99) n, (node(j),j=1,n)
      k = k+1
      do 80 j=1,n
         node(j)=ibf(node(j))
   80 continue
      write (*,100) n, (node(j),j=1,n)
  100 format(50i4)
      go to 70
   99 close(1)
      close(2)
      write (*,*) kv,k,'    <<<<<===== move this line to first row'
      end
```

Coordinates vertices (before reduction)

Mar 4 13:57 1987 s.s Page 1

1	0.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
2	1.000000000e0	1.912500000e0	-3.458300000e0	-1.104200000e0
3	1.000000000e0	1.912500000e0	-3.458300000e0	1.104200000e0
4	1.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
5	0.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
6	1.000000000e0	-6.031047813e-16	-3.458300000e0	-2.208300000e0
7	1.000000000e0	1.912500000e0	-3.458300000e0	-1.104200000e0
8	1.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
9	0.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
10	1.000000000e0	-1.912500000e0	-3.458300000e0	-1.104200000e0
11	1.000000000e0	-6.031047813e-16	-3.458300000e0	-2.208300000e0
12	1.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
13	0.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
14	1.000000000e0	-1.912500000e0	-3.458300000e0	1.104200000e0
15	1.000000000e0	-1.912500000e0	-3.458300000e0	-1.104200000e0
16	1.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
17	0.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
18	1.000000000e0	-6.031047813e-16	-3.458300000e0	2.208300000e0
19	1.000000000e0	-1.912500000e0	-3.458300000e0	1.104200000e0
20	1.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
21	0.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
22	1.000000000e0	1.912500000e0	-3.458300000e0	1.104200000e0
23	1.000000000e0	-6.031047813e-16	-3.458300000e0	2.208300000e0
24	1.000000000e0	-2.150794109e-16	-1.233300000e0	0.000000000e0
25	0.000000000e0	1.015800000e0	1.958300000e0	2.215000000e0
26	1.000000000e0	1.402500000e0	1.958300000e0	1.992500000e0
27	1.000000000e0	1.402500000e0	-1.958300000e0	1.992500000e0
28	1.000000000e0	1.015800000e0	-1.958300000e0	2.215000000e0
29	1.000000000e0	1.015800000e0	1.958300000e0	2.215000000e0
30	0.000000000e0	1.402500000e0	1.958300000e0	-1.992500000e0
31	1.000000000e0	1.015800000e0	1.958300000e0	-2.215000000e0
32	1.000000000e0	1.015800000e0	-1.958300000e0	-2.215000000e0
33	1.000000000e0	1.402500000e0	-1.958300000e0	-1.992500000e0
34	1.000000000e0	1.402500000e0	1.958300000e0	-1.992500000e0
35	0.000000000e0	-2.418300000e0	1.958300000e0	-2.417000000e-1
36	1.000000000e0	-2.418300000e0	1.958300000e0	2.417000000e-1
37	1.000000000e0	-2.418300000e0	-1.958300000e0	2.417000000e-1
38	1.000000000e0	-2.418300000e0	-1.958300000e0	-2.417000000e-1
39	1.000000000e0	-2.418300000e0	1.958300000e0	-2.417000000e-1
40	0.000000000e0	1.015800000e0	1.958300000e0	2.215000000e0
41	1.000000000e0	1.015800000e0	-1.958300000e0	2.215000000e0
42	1.000000000e0	2.658000000e-1	-1.958300000e0	3.482500000e0
43	1.000000000e0	2.658000000e-1	1.958300000e0	3.482500000e0
44	1.000000000e0	1.015800000e0	1.958300000e0	2.215000000e0
45	0.000000000e0	1.402500000e0	1.958300000e0	1.992500000e0
46	1.000000000e0	2.877400000e0	1.958300000e0	1.992500000e0
47	1.000000000e0	2.877400000e0	-1.958300000e0	1.992500000e0
48	1.000000000e0	1.402500000e0	-1.958300000e0	1.992500000e0
49	1.000000000e0	1.402500000e0	1.958300000e0	1.992500000e0
50	0.000000000e0	1.402500000e0	1.958300000e0	-1.992500000e0
51	1.000000000e0	1.402500000e0	-1.958300000e0	-1.992500000e0
52	1.000000000e0	2.877400000e0	-1.958300000e0	-1.992500000e0
53	1.000000000e0	2.877400000e0	1.958300000e0	-1.992500000e0
54	1.000000000e0	1.402500000e0	1.958300000e0	-1.992500000e0
55	0.000000000e0	1.015800000e0	1.958300000e0	-2.215000000e0
56	1.000000000e0	2.658000000e-1	1.958300000e0	-3.482500000e0

57	1.000000000e0	2.658000000e-1	-1.958300000e0	-3.482500000e0
58	1.000000000e0	1.015800000e0	-1.958300000e0	-2.215000000e0
59	1.000000000e0	1.015800000e0	1.958300000e0	-2.215000000e0
60	0.000000000e0	-2.418300000e0	1.958300000e0	-2.417000000e-1
61	1.000000000e0	-2.418300000e0	-1.958300000e0	-2.417000000e-1
62	1.000000000e0	-3.144200000e0	-1.958300000e0	-1.500000000e0
63	1.000000000e0	-3.144200000e0	1.958300000e0	-1.500000000e0
64	1.000000000e0	-2.418300000e0	1.958300000e0	-2.417000000e-1
65	0.000000000e0	-2.418300000e0	1.958300000e0	2.417000000e-1
66	1.000000000e0	-3.144200000e0	1.958300000e0	1.500000000e0
67	1.000000000e0	-3.144200000e0	-1.958300000e0	1.500000000e0
68	1.000000000e0	-2.418300000e0	-1.958300000e0	2.417000000e-1
69	1.000000000e0	-2.418300000e0	1.958300000e0	2.417000000e-1
70	0.000000000e0	-2.418300000e0	1.400000000e-1	2.417000000e-1
71	1.000000000e0	-2.418300000e0	1.400000000e-1	-2.417000000e-1
72	1.000000000e0	-2.418300000e0	-2.792000000e-1	0.000000000e0
73	1.000000000e0	-2.418300000e0	1.400000000e-1	2.417000000e-1
74	1.000000000e0	-2.916600000e0	5.086358631e-16	0.000000000e0
75	1.000000000e0	-2.418300000e0	1.400000000e-1	-2.417000000e-1
76	1.000000000e0	-2.916600000e0	5.086358631e-16	0.000000000e0
77	1.000000000e0	-2.418300000e0	-2.792000000e-1	0.000000000e0
78	1.000000000e0	-2.916600000e0	5.086358631e-16	0.000000000e0
79	1.000000000e0	-3.733300000e0	6.510629732e-16	0.000000000e0
80	1.000000000e0	-2.916600000e0	5.086358631e-16	0.000000000e0
81	1.000000000e0	-2.418300000e0	-2.792000000e-1	0.000000000e0
82	1.000000000e0	-2.916600000e0	5.086358631e-16	0.000000000e0
83	1.000000000e0	-2.418300000e0	1.400000000e-1	-2.417000000e-1
84	1.000000000e0	-2.916600000e0	5.086358631e-16	0.000000000e0
85	1.000000000e0	-2.418300000e0	1.400000000e-1	2.417000000e-1
86	1.000000000e0	-2.418300000e0	-2.792000000e-1	0.000000000e0
87	1.000000000e0	-2.418300000e0	1.400000000e-1	-2.417000000e-1
88	1.000000000e0	-2.418300000e0	1.400000000e-1	2.417000000e-1
89	0.000000000e0	2.658000000e-1	1.958300000e0	3.482500000e0
90	1.000000000e0	2.658000000e-1	-1.958300000e0	3.482500000e0
91	1.000000000e0	-3.144200000e0	-1.958300000e0	1.500000000e0
92	1.000000000e0	-3.144200000e0	1.958300000e0	1.500000000e0
93	1.000000000e0	2.658000000e-1	1.958300000e0	3.482500000e0
94	0.000000000e0	2.877400000e0	1.958300000e0	1.992500000e0
95	1.000000000e0	2.877400000e0	1.958300000e0	-1.992500000e0
96	1.000000000e0	2.877400000e0	-1.958300000e0	-1.992500000e0
97	1.000000000e0	2.877400000e0	-1.958300000e0	1.992500000e0
98	1.000000000e0	2.877400000e0	1.958300000e0	1.992500000e0
99	0.000000000e0	2.658000000e-1	1.958300000e0	-3.482500000e0
100	1.000000000e0	-3.144200000e0	1.958300000e0	-1.500000000e0
101	1.000000000e0	-3.144200000e0	-1.958300000e0	-1.500000000e0
102	1.000000000e0	2.658000000e-1	-1.958300000e0	-3.482500000e0
103	1.000000000e0	2.658000000e-1	1.958300000e0	-3.482500000e0
104	0.000000000e0	2.658000000e-1	1.958300000e0	3.482500000e0
105	1.000000000e0	-3.144200000e0	1.958300000e0	1.500000000e0
106	1.000000000e0	-2.418300000e0	1.958300000e0	2.417000000e-1
107	1.000000000e0	-2.418300000e0	1.958300000e0	-2.417000000e-1
108	1.000000000e0	-3.144200000e0	1.958300000e0	-1.500000000e0
109	1.000000000e0	2.658000000e-1	1.958300000e0	-3.482500000e0
110	1.000000000e0	1.015800000e0	1.958300000e0	-2.215000000e0
111	1.000000000e0	1.402500000e0	1.958300000e0	-1.992500000e0
112	1.000000000e0	2.877400000e0	1.958300000e0	-1.992500000e0

Vertex indices for each plane


```
4     1 2 3 4
4     5 6 7 8
4     9 10 11 12
4     13 14 15 16
4     17 18 19 20
4     21 22 23 24
5     25 26 27 28 29
5     30 31 32 33 34
5     35 36 37 38 39
5     40 41 42 43 44
5     45 46 47 48 49
5     50 51 52 53 54
5     55 56 57 58 59
5     60 61 62 63 64
5     65 66 67 68 69
19    70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
5     89 90 91 92 93
5     94 95 96 97 98
5     99 100 101 102 103
13    104 105 106 107 108 109 110 111 112 113 114 115 116
5     117 118 119 120 121
5     122 123 124 125 126
5     127 128 129 130 131
5     132 133 134 135 136
5     137 138 139 140 141
5     142 143 144 145 146
5     147 148 149 150 151
7     152 153 154 155 156 157 158
5     159 160 161 162 163
5     164 165 166 167 168
5     169 170 171 172 173
5     174 175 176 177 178
5     179 180 181 182 183
5     184 185 186 187 188
5     189 190 191 192 193
7     194 195 196 197 198 199 200
5     201 202 203 204 205
5     206 207 208 209 210
5     211 212 213 214 215
5     216 217 218 219 220
5     221 222 223 224 225
5     226 227 228 229 230
5     231 232 233 234 235
5     236 237 238 239 240
5     241 242 243 244 245
5     246 247 248 249 250
5     251 252 253 254 255
5     256 257 258 259 260
5     261 262 263 264 265
5     266 267 268 269 270
5     271 271 273 274 275
5     276 277 278 279 280
5     281 282 283 284 285
5     286 287 288 289 290
5     291 292 293 294 295
5     296 297 298 299 300
```

```
5      301 302 303 304 305
5      306 307 308 309 310
5      311 312 313 314 315
5      316 317 318 319 320
5      321 322 323 324 325
5      326 327 328 329 330
5      331 332 333 334 335
5      336 337 338 339 340
5      341 342 343 344 345
5      346 347 348 349 350
5      351 352 353 354 355
5      356 357 358 359 360
5      361 362 363 364 365
5      366 367 368 369 370
5      371 372 373 374 375
5      376 377 378 379 380
5      381 382 383 384 385
5      386 387 388 389 390
9      391 392 393 394 395 396 397 398 399
end
```

Sattelite data after reduction

Mar 8 00:13 1987 solar.s Page 1

```
116  68
 -15.0   15.0  -15.0 15.0    -15.0 15.0
-0.215079E-15 -1.23330      0.000000E+00
  1.91250      -3.45830      -1.10420
  1.91250      -3.45830       1.10420
-0.603105E-15 -3.45830      -2.20830
 -1.91250      -3.45830      -1.10420
 -1.91250      -3.45830       1.10420
-0.603105E-15 -3.45830       2.20830
  1.01580       1.95830       2.21500
  1.40250       1.95830       1.99250
  1.40250      -1.95830       1.99250
  1.01580      -1.95830       2.21500
  1.40250       1.95830      -1.99250
  1.01580       1.95830      -2.21500
  1.01580      -1.95830      -2.21500
  1.40250      -1.95830      -1.99250
 -2.41830       1.95830     -0.241700
 -2.41830       1.95830      0.241700
 -2.41830      -1.95830      0.241700
 -2.41830      -1.95830     -0.241700
 0.265800      -1.95830       3.48250
 0.265800       1.95830       3.48250
  2.87740       1.95830       1.99250
  2.87740      -1.95830       1.99250
  2.87740      -1.95830      -1.99250
  2.87740       1.95830      -1.99250
 0.265800       1.95830      -3.48250
 0.265800      -1.95830      -3.48250
 -3.14420      -1.95830      -1.50000
 -3.14420       1.95830      -1.50000
 -3.14420       1.95830       1.50000
 -3.14420      -1.95830       1.50000
 -2.41830      0.140000      0.241700
 -2.41830      0.140000     -0.241700
 -2.41830     -0.279200      0.000000E+00
 -2.91660      0.508636E-15  0.000000E+00
 -3.73330      0.651063E-15  0.000000E+00
 -1.05910       2.80830       2.55750
  1.05910       2.80830       2.55750
  1.05910       1.95830       2.55750
 -1.05910       1.95830       2.55750
-0.750000E-01   2.51330       2.55750
 0.750000E-01   2.51330       2.55750
 0.750000E-01   2.51330       2.78000
-0.750000E-01   2.51330       2.78000
 0.150000       2.38330       2.55750
 0.150000       2.38330       2.78000
 0.750000E-01   2.25330       2.55750
 0.750000E-01   2.25330       2.78000
-0.750000E-01   2.25330       2.55750
-0.750000E-01   2.25330       2.78000
-0.150000       2.38330       2.55750
-0.150000       2.38330       2.78000
  1.05910       2.80830      -2.55750
 -1.05910       2.80830      -2.55750
```

```
 -1.05910        1.95830      -2.55750
  1.05910        1.95830      -2.55750
-0.750000E-01    2.51330      -2.78000
 0.750000E-01    2.51330      -2.78000
 0.750000E-01    2.51330      -2.55750
-0.750000E-01    2.51330      -2.55750
 0.150000        2.38330      -2.78000
 0.150000        2.38330      -2.55750
 0.750000E-01    2.25330      -2.78000
 0.750000E-01    2.25330      -2.55750
-0.750000E-01    2.25330      -2.78000
-0.750000E-01    2.25330      -2.55750
-0.150000        2.38330      -2.78000
-0.150000        2.38330      -2.55750
  2.55750        2.80830       1.05910
  2.55750        2.80830      -1.05910
  2.55750        1.95830      -1.05910
  2.55750        1.95830       1.05910
 -2.55750        2.80830      -1.05910
 -2.55750        1.95830      -1.05910
 -2.55750        2.80830       1.05910
 -2.55750        1.95830       1.05910
  1.63420        2.82330      -5.80320
  8.12650        2.32670      -8.69410
  9.68820        2.32670      -5.18660
  3.19580        2.82330      -2.29580
  11.2500        2.32670      -1.68000
  4.75740        2.82330       1.21080
  12.8115        2.32670       1.82750
  6.31910        2.82330       4.71830
  2.75830        2.91660     -0.166700
  3.90910        2.82330     -0.695000
  4.04410        2.82330     -0.390000
  2.75830        2.91660      0.166700
 -12.8115        2.32670      -1.82750
 -6.31910        2.82330      -4.71830
 -4.75740        2.82330      -1.21080
 -11.2500        2.32670       1.68000
 -3.19580        2.82330       2.29580
 -9.68820        2.32670       5.18660
 -1.63420        2.82330       5.80320
 -8.12650        2.32670       8.69410
 -4.04410        2.82330      0.390000
 -2.75830        2.91660     -0.166700
 -2.75830        2.91660      0.166700
 -3.90910        2.82330      0.695000
  2.44490        3.17500       2.23830
  2.44490        3.17500      -2.23830
  1.12660        3.17500      -3.33750
 -1.12660        3.17500      -3.33750
 -2.44490        3.17500      -2.23830
 -2.44490        3.17500       2.23830
 -1.12660        3.17500       3.33750
  1.12660        3.17500       3.33750
  2.44490        10.5750       2.23830
  2.44490        10.5750      -2.23830
```

```
  1.12660        10.5750       -3.33750
 -1.12660        10.5750       -3.33750
 -2.44490        10.5750       -2.23830
 -2.44490        10.5750        2.23830
 -1.12660        10.5750        3.33750
  1.12660        10.5750        3.33750
    4    1    2    3    1
    4    1    4    2    1
    4    1    5    4    1
    4    1    6    5    1
    4    1    7    6    1
    4    1    3    7    1
    5    8    9   10   11    8
    5   12   13   14   15   12
    5   16   17   18   19   16
    5    8   11   20   21    8
    5    9   22   23   10    9
    5   12   15   24   25   12
    5   13   26   27   14   13
    5   16   19   28   29   16
    5   17   30   31   18   17
    5   32   35   36   35   32
    5   33   35   34   35   33
    5   21   20   31   30   21
    5   22   25   24   23   22
    5   26   29   28   27   26
   13   21   30   17   16   29   26   13   12   25   22    9    8   21
    5   37   38   39   40   37
    5   41   42   43   44   41
    5   42   45   46   43   42
    5   45   47   48   46   45
    5   47   49   50   48   47
    5   49   51   52   50   49
    5   51   41   44   52   51
    7   52   44   43   46   48   50   52
    5   53   54   55   56   53
    5   57   58   59   60   57
    5   58   61   62   59   58
    5   61   63   64   62   61
    5   63   65   66   64   63
    5   65   67   68   66   65
    5   67   57   60   68   67
    7   67   65   63   61   58   57   67
    5   69   70   71   72   69
    5   70   53   56   71   70
    5   54   73   74   55   54
    5   73   75   76   74   73
    5   75   37   40   76   75
    5   38   69   72   39   38
    5   89   90   91   92   89
    5   92   91   93   94   92
    5   94   93   95   96   94
    5   97   98   99  100   97
    5   77   80   79   78   77
    5   80   82   81   79   80
    5   82   84   83   81   82
```

```
    5  85  88  87  86  85
    5 101 102  70  69 101
    5 102 103  53  70 102
    5 103 104  54  53 103
    5 104 105  73  54 104
    5 105 106  75  73 105
    5 106 107  37  75 106
    5 107 108  38  37 107
    5 108 101  69  38 108
    5 109 110 102 101 109
    5 110 111 103 102 110
    5 111 112 104 103 111
    5 112 113 105 104 112
    5 113 114 106 105 113
    5 114 115 107 106 114
    5 115 116 108 107 115
    5 116 109 101 108 116
    9 110 109 116 115 114 113 112 111 110
```

APPENDIX C: C PROGRAM FOR GRAPHICS MODELING

ON THE SUN WORKSTATION

```
/*          shad.c                                          */

#include <usercore.h>
#include <sun/fbio.h>
#include <math.h>
#include <stdio.h>
#include "demolib.h"

#define MAXVERT 800
#define MAXPOLY 800
#define MAXPVERT 6400
static int nvert, npoly;
static float *wldx, *wldy;
static float *ndcx, *ndcy;
static char string[81];
static float bbox[3][2];
static float planeq[MAXPOLY][4];
static float vertices[MAXVERT][3];
static float normal[MAXVERT][3];
static int cindex[MAXVERT];
static short normalcount[MAXVERT];
static int npvert[MAXPOLY];
static short *pvertptr[MAXPOLY];
static short pvert[MAXPVERT];
static float xlist[10], ylist[10], zlist[10];
static float dxlist[10], dylist[10], dzlist[10];
static int  indxlist[10];
static float red[256], grn[256], blu[256];
static float dred[256], dgrn[256], dblu[256];
static int renderstyle=1;
static int renderhue=0;
static float xcut[2]={0.,1.}, zcut0[2]={0.,0.}, zcut[2]={0.,1.};


main(argc, argv)
int argc;
char *argv[];
{
        int i, disopt, length = 0, visible();
        static char str[] = "Enter your choice (1-5) ?";
        float lx,ly,lz,vx,vy,vz,x, y, z, lxr, lyr, lzr;
        float c0,s0,theta,temp1,temp2;

        if (argc < 2) { printf("Usage: shadeobj objfile\n"); exit(1); }
        if (getobjdat(argv[1])) exit(2);
        initvw();
        get_view_surface(our_surface,argv);
        start_up_core();
        setvwpv();
cycle:
        style_select();
        new_frame();
        create_temporary_segment();
        move_abs_2(300.,700.);
        text("Enter the desired display option");
        move_abs_2(330.,650.);
        text("1) Still frame");
        move_abs_2(330.,620.);
        text("2) Potate the viewer [default]");
        move_abs_2(330.,590.);
```

```
        text("3) Rotate the object");
        move_abs_2(330.,560.);
        text("4) Rotate the light source");
        move_abs_2(330.,530.);
        text("5) Quit");
        move_abs_2(300.,480.);
        text(str);
/*        inquire_text_extent_2(str,wldx,wldy);
        map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
        set_echo_position(KEYBOARD,1,*ndcx,*ndcy); */
        set_echo_position(KEYBOARD,1,0.5,0.47);
        await_keyboard(1000000000, 1, string, &length);
        disopt = atoi(string);
        close_temporary_segment();
        if (disopt == 0) disopt = 2;
        length = 0;
        switch (disopt) {
            case 1:
                new_frame();
                getxyz(&lx,&ly,&lz,&vx,&vy,&vz);
                set_light_direction(lx ,ly ,lz );
                setvwpo(vx,vy,vz);
                new_frame();
                create_temporary_segment();
                    set_primitive_attributes( &PRIMATTS);
                    set_polygon_interior_style( SHADED);
                    drawobj();
                close_temporary_segment();
                do {
                        await_keyboard(0, 1, string, &length);
                        }
                while (length == 0);
                break;
             case 2:
                new_frame();
                getxyz(&lx,&ly,&lz,&vx,&vy,&vz);
                 set_light_direction(lx ,ly ,lz );
                theta = 0.;
                do {
                    new_frame();
                    c0 = cos(theta);
                    s0 = sin(theta);
                    x  = vx * c0 + vz * s0;
                    y  = vy;
                    z  = -s0 * vx + vz * c0;
                    setvwpo(x,y,z);
                    create_temporary_segment();
                    set_primitive_attributes( &PRIMATTS);
                    set_polygon_interior_style( SHADED);
                    drawobj();
                    close_temporary_segment();
                    theta = theta + .17453 ;
                    await_keyboard(3000000, 1, string, &length);
                    }
                while (length == 0);
                break;
        case 3:
                new_frame();
                getxyz(&lx,&ly,&lz,&vx,&vy,&vz);
                set_light_direction(lx ,ly ,lz );
```

```
            setvwpo(vx,vy,vz);
            theta = .17453;
            do {
                new_frame();
                create_temporary_segment();
                set_primitive_attributes( &PRIMATTS);
                set_polygon_interior_style( SHADED);
                drawobj();
                close_temporary_segment();
                c0 = cos(theta);
                s0 = sin(theta);
                x  = vx * c0 + vz * s0;
                y  = vy;
                z  = -s0 * vx + vz * c0;
                setvwpo(x,y,z);
                lxr = lx * c0 + lz * s0;
                lyr = ly;
                lzr = -s0 * lx + lz * c0;
                set_light_direction(lxr ,lyr ,lzr );
                for (i = 0; i < nvert; i++) {
                    temp1 = c0 * vertices[i][0] + s0 * vertices[i][2];
                    temp2 = -s0 * vertices[i][0] + c0 * vertices[i][2];
                    vertices[i][0] = temp1;
                    vertices[i][2] = temp2;
                    }
                await_keybboard(3000000, 1, string, &length);
                theta = theta + .17453 ;
                }
            while (length == 0);
            break;
    case 4:
            new_frame();
            getxyz(&lx,&ly,&lz,&vx,&vy,&vz);
            setvwpo(vx,vy,vz);
            theta = 0.;
            do {
                new_frame();
                c0 = cos(theta);
                s0 = sin(theta);
                x  = lx * c0 + lz * s0;
                y  = ly;
                z  = -s0 * lx + lz * c0;
                set_light_direction(x ,y ,z );
                create_temporary_segment();
                set_primitive_attributes( &PRIMATTS);
                set_polygon_interior_style( SHADED);
                drawobj();
                close_temporary_segment();
                theta = theta + .17453 ;
                await_keyboard(3000000, 1, string, &length);
                }
            while (length == 0);
            break;
    case 5:
            shut_down_core();
            exit();
    default:
            goto cycle;
    }
goto cycle;
```

```
}

style_select()
{
    static char str[] = "Enter your choice (1-5) ?";
    static char str1[] = "Enter your choice (1-5) ?";
    int done, segnam, pickid, butnum;
    int hue,length;

    new_frame();
        setvwpv();
        create_temporary_segment();
        move_abs_2(300.,700.);
        text("Enter the desired shading style");
        move_abs_2(330.,650.);
        text("1) Wireframe display");
        move_abs_2(330.,620.);
        text("2) Gray shading");
        move_abs_2(330.,590.);
        text("3) Gouraud");
        move_abs_2(330.,560.);
        text("4) Phong diffuse [default]");
        move_abs_2(330.,530.);
        text("5) Phong specular");
        move_abs_2(300.,480.);
        text(str);
/*      inquire_text_extent_2(str,wldx,wldy);
        map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
        set_echo_position(KEYBOARD,1,*ndcx,*ndcy);*/
        set_echo_position(KEYBOARD,1,0.5,0.47);
        await_keyboard(1000000000, 1, string, &length);
        close_temporary_segment();
        renderstyle = atoi(string) -1;
        if (renderstyle == -1) renderstyle = 3;
        new_frame();
        create_temporary_segment();
        move_abs_2(300.,700.);
        text("Enter the desired shading color");
        move_abs_2(330.,650.);
        text("1) Gray");
        move_abs_2(330.,620.);
        text("2) Red");
        move_abs_2(330.,590.);
        text("3) Green [default]");
        move_abs_2(330.,560.);
        text("4) Blue");
        move_abs_2(330.,530.);
        text("5) Yellow");
        move_abs_2(300.,480.);
        text(str1);
/*        inquire_text_extent_2(str1,wldx,wldy);
        map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
        set_echo_position(KEYBOARD,1,*ndcx,*ndcy);*/
        set_echo_position(KEYBOARD,1,0.52,0.47);
        await_keyboard(1000000000, 1, string, &length);
        renderhue = atoi(string) - 1;
        if (renderhue == -1) renderhue = 2;
        close_temporary_segment();
    if (renderhue == 0) define_color_indices( our_surface,0,255,red,grn,blu);
    else define_color_indices( our_surface,0,255,dred,dgrn,dblu);
```

```
                    /* ambient,diffuse,specular,flood,bump,hue,style */
    hue = renderhue;
    switch (renderstyle) {
    case 1: set_shading_parameters( .01, .96, .0, 0., 7.,hue,0); break;
    case 2: set_shading_parameters( .01, .96, .0, 0., 7.,hue,1); break;
    case 3: set_shading_parameters( .01, .95, .0, 0., 7.,hue,2); break;
    case 4: set_shading_parameters( .05, .50, .40, 0., 7.,hue,2); break;
    default: break;
    }
}

getxyz(lx,ly,lz,vx,vy,vz)
float *lx,*ly,*lz,*vx,*vy,*vz;
{
static char str1[] = "x= ";
static char str2[] = "y= ";
static char str3[] = "z= ";
static char str4[] = "x= ";
static char str5[] = "y= ";
static char str6[] = "z= ";
int length;

setvwpv();
set_text_index(1);
new_frame();
create_temporary_segment();
move_abs_2(300.,700.); text("enter the  light source position [default : 0.0,0.0,-1.0]"
/* set_echo_position(KEYBOARD,1,.3,.4);*/
move_abs_2(300.,680.);
        text(str1);
/*        inquire_text_extent_2(str1,wldx,wldy);
        map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
        set_echo_position(KEYBOARD,1,*ndcx,*ndcy); */
        set_echo_position(KEYBOARD,1,0.31,0.665);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
   *lx = 0.0;
else
   *lx = atof(string);
move_abs_2(300.,660.);
        text(str2);
/*        inquire_text_extent_2(str2,wldx,wldy);
        map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
        set_echo_position(KEYBOARD,1,*ndcx,*ndcy);*/
        set_echo_position(KEYBOARD,1,0.31,0.645);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
   *ly = 0.0;
else
   *ly = atof(string);
move_abs_2(300.,640.);
        text(str3);
/*        inquire_text_extent_2(str3,wldx,wldy);
        map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
        set_echo_position(KEYBOARD,1,*ndcx,*ndcy);*/
        set_echo_position(KEYBOARD,1,0.31,0.625);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
   *lz = 0.0;
else
```

```
    *lz = atof(string);
if ((*lx == 0.0) && (*ly == 0.0) && (*lz == 0.0)) {
     *lx = 0.0;
     *ly = 0.0;
     *lz = -1.0;
}
move_abs_2(300.,600.); text("enter the viewer position [default : 4000, 5000, 6000]");
move_abs_2(300.,580.);
          text(str4);
/*          inquire_text_extent_2(str4,wldx,wldy);
          map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
          set_echo_position(KEYBOARD,1,*ndcx,*ndcy);*/
          set_echo_position(KEYBOARD,1,0.31,0.565);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *vx = 0;
else
    *vx = atof(string);
move_abs_2(300.,560.);
          text(str5);
/*          inquire_text_extent_2(str5,wldx,wldy);
          map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
          set_echo_position(KEYBOARD,1,*ndcx,*ndcy);*/
          set_echo_position(KEYBOARD,1,0.31,0.548);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *vy = 0.0;
else
    *vy = atof(string);
move_abs_2(300.,540.);
          text(str6);
/*          inquire_text_extent_2(str6,wldx,wldy);
          map_world_to_ndc_2(*wldx,*wldy,ndcx,ndcy);
          set_echo_position(KEYBOARD,1,*ndcx,*ndcy);*/
          set_echo_position(KEYBOARD,1,0.31,0.527);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *vz = 0.0;
else
    *vz = atof(string);
if ((*vx == 0.0) && (*vy == 0.0) && (*vz == 0.0)) {
    *vx = 4000.0;
    *vy = 5000.0;
    *vz = 6000.0;
}
close_temporary_segment();
}

start_up_core()
{
          int i;
          float x,y,z;

          initialize_core(BASIC, SYNCHRONOUS, THREED);
          our_surface->cmapsize = 256;
          our_surface->cmapname[0] = '\0';
          if(initialize_view_surface(our_surface, TRUE)) exit(1);
          select_view_surface(our_surface);
          inquire_color_indices(our_surface,0,255,dred,dgrn,dblu);
          for (i=1; i<=255; i++) {                    /* load color LUT */
```

```
                red[i] = (float)i * 0.003921568;
                grn[i] = (float)i * 0.003921568;
                blu[i] = (float)i * 0.003921568;;
                }
        red[0] = 0.; grn[0] = .7; blu[0] = 0.;

        define_color_indices( our_surface,0,255,red,grn,blu);
                        /* ambient,diffuse,specular,flood,bump,hue,style */
        set_shading_parameters( .01, .96, .0, 0., 7.,0,0);

        initialize_device(KEYBOARD, 1);
        set_echo( KEYBOARD , 1 , 1 );
        set_echo_surface( KEYBOARD, 1, our_surface);
        set_keyboard(1, 80, "", 1);
        setvwpv();
        set_font(1);
        }

shut_down_core()
{
        terminate_device(KEYBOARD, 1);
        deselect_view_surface(our_surface);
        terminate_view_surface(our_surface);
        terminate_core();
        }

int getobjdat(filename)
char *filename;
        {
        int i, j, k;
        short vtmp, v1, v2, v3;
        float ftmp, maxd, scale, offset[3];
        float x,y,z,x0,y0,z0,length;
        FILE *fptr;

        if ((fptr = fopen(filename, "r")) == NULL) {
                printf("Can't open file: %s\n", filename);
                return(1);
                }
        fscanf(fptr, "%d%d", &nvert, &npoly);
        if ((nvert > MAXVERT) || (npoly > MAXPOLY)) {
                printf("Too many object vertices or polygons\n");
                return(2);
                }
        fscanf(fptr, "%f%f%f%f%f%f", &bbox[0][0], &bbox[0][1], &bbox[1][0],
                &bbox[1][1], &bbox[2][0], &bbox[2][1]);
        maxd = 0.0;
        for (i = 0; i < 3; i++) {
                offset[i] = (bbox[i][0] + bbox[i][1]) / 2.0;
                bbox[i][0] -= offset[i];
                bbox[i][1] -= offset[i];
                if (bbox[i][0] > bbox[i][1]) {
                        ftmp = bbox[i][0];
                        bbox[i][0] = bbox[i][1];
                        bbox[i][1] = ftmp;
                        }
                if (maxd < bbox[i][1])
                        maxd = bbox[i][1];
                }
        scale = 1000.0 / maxd;
```

```
        for (i = 0; i < 3; i++) {
                bbox[i][0] *= scale;
                bbox[i][1] *= scale;
                }
        for (i = 0; i < nvert; i++) {
                fscanf(fptr, "%f%f%f", &vertices[i][0], &vertices[i][1],
                        &vertices[i][2]);
                vertices[i][0] = (vertices[i][0] - offset[0]) * scale;
                vertices[i][1] = (vertices[i][1] - offset[1]) * scale;
                vertices[i][2] = (vertices[i][2] - offset[2]) * scale;
                normal[i][0] = 0.0; normal[i][1] = 0.0; normal[i][2] = 0.0;
                normalcount[i] = 0;
                }
        k = 0;
        for (i = 0; i < npoly; i++) {
                fscanf(fptr, "%d", &npvert[i]);
                if ((k + npvert[i]) > MAXPVERT) {
                        printf("Too many polygon vertices\n");
                        return(3);
                        }
                pvertptr[i] = &pvert[k];
                for (j = 0; j < npvert[i]; j++) {
                        fscanf(fptr, "%hd", &vtmp);
                        pvert[k++] = vtmp - 1;
                        }
                planeq[i][0] = planeq[i][1] = planeq[i][2] = planeq[i][3] = 0.0;
                v1 = pvert[k - 1]; v2 = pvert[k - 2]; v3 = pvert[k - 3];
                for (j = 0; j < 3; j++) {
                        planeq[i][0] += vertices[v1][1] *
                                        (vertices[v2][2] - vertices[v3][2]);
                        planeq[i][1] += vertices[v1][0] *
                                        (vertices[v3][2] - vertices[v2][2]);
                        planeq[i][2] += vertices[v1][0] *
                                        (vertices[v2][1] - vertices[v3][1]);
                        planeq[i][3] += vertices[v1][0] *
                                        ((vertices[v3][1] * vertices[v2][2]) -
                                         (vertices[v2][1] * vertices[v3][2]));
                        vtmp = v1; v1 = v2; v2 = v3; v3 = vtmp;
                        }
                if (planeq[i][3] > 0.0)
                        for (j = 0; j <= 3; j++) planeq[i][j] = -planeq[i][j];
*/
                for (j = 1; j <= npvert[i]; j++) {      /* accum normls */
                        vtmp = pvert[k-j];
                        x = planeq[i][0]; y = planeq[i][1]; z = planeq[i][2];
                        length = sqrt( x*x + y*y + z*z);
                        normal[vtmp][0] += x/length;
                        normal[vtmp][1] += y/length;
                        normal[vtmp][2] += z/length;
                        normalcount[vtmp]++;
                        }
                }
        for (i = 0; i < nvert; i++) {
                normal[i][0] /= normalcount[i];
                normal[i][1] /= normalcount[i];
                normal[i][2] /= normalcount[i];
                }
        fclose(fptr);
        return(0);
        }
```

```
setvwpo(vx, vy, vz)
float vx, vy, vz;
        {
         int i;
        float diag, del, objdist, near;

        set_view_reference_point(vx, vy, vz);
        set_view_plane_normal(-vx, -vy, -vz);
        set_projection(PERSPECTIVE, 0., 0., 0.);
        set_view_plane_distance(256.0);
        if ((vx == 0.0) && (vz == 0.0))
                set_view_up_3(0.0, 0.0, vy);
        else
                set_view_up_3(0.0, 1.0, 0.0);
        set_window(-80.0, 80.0, -80.0, 80.0);
        diag = 0.0;
        for (i = 0; i < 3; i++) {
                del = bbox[i][1] - bbox[i][0];
                diag += del * del;
                }
        diag = sqrt(diag) / 2.0;
        objdist = sqrt( vx*vx + vy*vy + vz*vz);
        near = (diag >= objdist) ? objdist/2.0 : objdist-diag;
        set_view_depth( near, objdist + diag);
        set_window_clipping(TRUE);
        set_front_plane_clipping(TRUE);
        set_back_plane_clipping(TRUE);
        set_viewport_3(.125, .874, 0., .749, 0.0, 1.0);
        }

static float invxform[4][4];

drawobj()
        {
        int i;
        float x,y,z,x0,y0,z0,length;
        char ch;

        if (renderstyle && renderstyle<3)
                map_ndc_to_world_3( -348., 348., -870., &x,&y,&z);
                map_ndc_to_world_3( 0.0, 0.0, 0.0, &x0,&y0,&z0);
                x -= x0; y -= y0; z -= z0;
                length = sqrt( x*x + y*y + z*z);
                if (length != 0.0) {
                        x /= length; y /= length; z /= length;
                        }
            for (i = 0; i < nvert; i++) {
                cindex[i] =
                fabs(normal[i][0]*x +normal[i][1]*y +normal[i][2]*z) * 254.;
                if (cindex[i] < 4) cindex[i] = 4;
                if (cindex[i] > 248) cindex[i] = 248;
                }
        /* inquire_inverse_composite_matrix(invxform); */
        if (renderstyle == 3) set_zbuffer_cut( our_surface, xcut, zcut, 2);
        else set_zbuffer_cut( our_surface, xcut, zcut0, 2);
        for (i = 0; i < npoly; i++) {
                /* if (visible(planeq[i])) */
                drawface(i);
```

```
                }
        }

int visible(pln)
float pln[];
        {
        int i;
        float c;

        c = 0.0;
        for (i = 0; i < 4; i++)
                c += invxform[2][i] * pln[i];
        return(c < 0.0);
        }

drawface(p) int p;
{
        int i, j, k;
        short *ptr;

        ptr = pvertptr[p];
        for (i = 0; i < npvert[p]; i++) {
                j = *ptr++;
                xlist[i] = vertices[j][0]; ylist[i] = vertices[j][1];
                zlist[i] = vertices[j][2];
                if (renderstyle < 3) {
                    indxlist[i] = cindex[j];
                }else {
                    dxlist[i] = normal[j][0]; dylist[i] = normal[j][1];
                    dzlist[i] = normal[j][2];
                    }
                }
        if (renderstyle < 2) {
                if (renderhue) {
                    j = indxlist[0]>>2; if (j > 62) j = 62;
                    set_fill_index( j + renderhue*64 -63);
                    }
                else set_fill_index( indxlist[0]);
                }
        else if (renderstyle == 2) {set_vertex_indices( indxlist, npvert[p]);}
        else {set_vertex_normals(dxlist,dylist,dzlist,npvert[p]);}
        if (renderstyle == 0) polyline_abs_3( xlist,ylist,zlist,npvert[p]);
        else polygon_abs_3( xlist, ylist, zlist, npvert[p]);
}

static float maxvw, vwpp, maxvdim;
static float vlx, vby, vfz, vdx, vdy, vdz;
static float minleft, maxright, minbot, maxtop, minback, maxfront;

initvw()
        {
        int i;
        float ftmp;

        ftmp = bbox[0][1];
        for (i = 1; i < 3; i++)
                if (bbox[i][1] > ftmp)
                        ftmp = bbox[i][1];
        maxvw = 16.0 * ftmp;
        vwpp = 2.0 * maxvw / 480.0;
```

```
        maxvdim = maxvw - ceil(vwpp);
        vlx = (bbox[0][0] + maxvw) / vwpp;
        vby = 100.0 + (bbox[1][0] + maxvw) / vwpp;
        vfz = 580.0 - (bbox[2][1] + maxvw) / vwpp;
        vdx = (bbox[0][1] - bbox[0][0]) / vwpp;
        vdy = (bbox[1][1] - bbox[1][0]) / vwpp;
        vdz = (bbox[2][1] - bbox[2][0]) / vwpp;
        minleft = bbox[0][0] - 5.0;
        maxright = bbox[0][1] + 5.0;
        minbot = bbox[1][0] - 5.0;
        maxtop = bbox[1][1] + 5.0;
        minback = bbox[2][0] - 5.0;
        maxfront = bbox[2][1] + 5.0;
        }

setvwpv()
        {
        set_view_reference_point(0.0, 0.0, 0.0);
        set_view_plane_normal(0.0, 0.0, -1.0);
        set_view_plane_distance(0.0);
        set_projection(PARALLEL, 0.0, 0.0, 1.0);
        set_view_up_3(0.0, 1.0, 0.0);
        set_window(0.0, 1023.0, 0.0, 767.0);
        set_view_depth( 0.0, 1.0);
        set_window_clipping(FALSE);
        set_viewport_3(0.0, 1., 0.0, .75, 0.0, 1.);
        }

int insideobj(x, y, z)
float x, y, z;
        {
        if ((x < minleft) || (x > maxright))
                return(0);
        if ((y < minbot) || (y > maxtop))
                return(0);
        if ((z < minback) || (z > maxfront))
                return(0);
        return(1);
        }
```

Data "torso.s"

```
49 10
-3 3 3.0 11.0 -4 4
    2.444900000e0     1.057500000e1     2.238300000e0
    2.444900000e0     1.057500000e1    -2.238300000e0
    2.444900000e0     3.175000000e0    -2.238300000e0
    2.444900000e0     3.175000000e0     2.238300000e0
    2.444900000e0     1.057500000e1     2.238300000e0
    2.444900000e0     1.057500000e1    -2.238300000e0
    1.126600000e0     1.057500000e1    -3.337500000e0
    1.126600000e0     3.175000000e0    -3.337500000e0
    2.444900000e0     3.175000000e0    -2.238300000e0
    2.444900000e0     1.057500000e1    -2.238300000e0
    1.126600000e0     1.057500000e1    -3.337500000e0
   -1.126600000e0     1.057500000e1    -3.337500000e0
   -1.126600000e0     3.175000000e0    -3.337500000e0
    1.126600000e0     3.175000000e0    -3.337500000e0
    1.126600000e0     1.057500000e1    -3.337500000e0
   -1.126600000e0     1.057500000e1    -3.337500000e0
   -2.444900000e0     1.057500000e1    -2.238300000e0
   -2.444900000e0     3.175000000e0    -2.238300000e0
   -1.126600000e0     3.175000000e0    -3.337500000e0
   -1.126600000e0     1.057500000e1    -3.337500000e0
   -2.444900000e0     1.057500000e1    -2.238300000e0
   -2.444900000e0     1.057500000e1     2.238300000e0
   -2.444900000e0     3.175000000e0     2.238300000e0
   -2.444900000e0     3.175000000e0    -2.238300000e0
   -2.444900000e0     1.057500000e1    -2.238300000e0
   -2.444900000e0     1.057500000e1     2.238300000e0
   -1.126600000e0     1.057500000e1     3.337500000e0
   -1.126600000e0     3.175000000e0     3.337500000e0
   -2.444900000e0     3.175000000e0     2.238300000e0
   -2.444900000e0     1.057500000e1     2.238300000e0
   -1.126600000e0     1.057500000e1     3.337500000e0
    1.126600000e0     1.057500000e1     3.337500000e0
    1.126600000e0     3.175000000e0     3.337500000e0
   -1.126600000e0     3.175000000e0     3.337500000e0
   -1.126600000e0     1.057500000e1     3.337500000e0
    1.126600000e0     1.057500000e1     3.337500000e0
    2.444900000e0     1.057500000e1     2.238300000e0
    2.444900000e0     3.175000000e0     2.238300000e0
    1.126600000e0     3.175000000e0     3.337500000e0
    1.126600000e0     1.057500000e1     3.337500000e0
    2.444900000e0     1.057500000e1    -2.238300000e0
    2.444900000e0     1.057500000e1     2.238300000e0
    1.126600000e0     1.057500000e1     3.337500000e0
   -1.126600000e0     1.057500000e1     3.337500000e0
   -2.444900000e0     1.057500000e1     2.238300000e0
   -2.444900000e0     1.057500000e1    -2.238300000e0
   -1.126600000e0     1.057500000e1    -3.337500000e0
    1.126600000e0     1.057500000e1    -3.337500000e0
    2.444900000e0     1.057500000e1    -2.238300000e0
5    1  2  3  4  5
5    6  7  8  9  10
5    11  12  13  14  15
5    16  17  18  19  20
5    21  22  23  24  25
5    26  27  28  29  30
5    31  32  33  34  35
5    36  37  38  39  40
9    41  42  43  44  45  46  47  48  49
```

Noise Program: temp.c

```
*****************************************************************************
*   Thu Aug 13 12:08:59 1987                                                *
*   /julius/j2/.yhl/nasa/temp.c                                             *
*****************************************************************************/
/*************************************************************************/
/* temp.c                                                                */
/* alias cscr tem                                                        */
/* f77 -f68881 -o !* !*.c -lfortd -lcore -lsunwindow -lpixrect -lm       */
/* width = 462(7 sin cycles), height = 325                               */
/* Output is to 19-inches bit-mapped color display having                */
/* 1152 by 900 pixels                                                    */
/*************************************************************************/
#include <usercore.h>
#include "demolib.h"
#include <sun/fbio.h>
#include <math.h>
#include <stdio.h>
#include <sys/file.h>
#include <pixrect/pixrect_hs.h>
#define COLOR_VWSURF(ddname) {"","",0,ddname,0,256,"",0,0};
struct pixrect *screen;

#define MAXVERT 500
#define MAXPOLY 500
#define MAXPVERT 3000

static int      nvert, npoly;
static float   *wldx, *wldy;
static float   *ndcx, *ndcy;
static char     string[81];
static float    lx, nlx, ly, nly, lz, nlz;
static float    bbox[3][2];
static float    planeq[MAXPOLY][4];
static float    vertices[MAXVERT][3];
static float    normal[MAXVERT][3];
static int      cindex[MAXVERT], cpindex[MAXPOLY];
static short    normalcount[MAXVERT];
static int      npvert[MAXPOLY];
static short   *pvertptr[MAXPOLY];
static short    pvert[MAXPVERT];
static float    xlist[MAXVERT], ylist[MAXVERT], zlist[MAXVERT];
static float    dxlist[MAXVERT], dylist[MAXVERT], dzlist[MAXVERT];
static int      indxlist[MAXVERT];
static float    bred[256], bgrn[256], bblu[256];
static float    cred[256], cgrn[256], cblu[256];
static float    ambient, diffuse, specular, flood, bump;

static int      renderstyle = 1;
static int      renderhue = 0;
static float    xcut[2] = {
                0., 1.
},              zcut0[2] = {
                 0., 1.
```

```
},              zcut[2] = {
                0., 1.

};
int             bw2dd();
int             pixel[560][440], width, height;
struct vwsurf   vwsurf =
COLOR_VWSURF(bw2dd);
struct suncore_raster raster;
int             autodraw, debug, noise;
void            normlig();

MAIN_()
{
    void            blkscr();
    double          dseed, dvalue;
    double          unit, amp = 14.;
    float           ggnqf_(), fvalue;
    float           factor1, factor2;
    int             cycles = 70;
    int             i, j;
    int             ax0, ay0, ax, ay;
    short           temp, vsin, vdsin, vnormal;
    int             disopt, length = 0, visible();

    float           angle = 0.785398163;  /* 45 degree */
    static char     str[] = "Enter your choice (1-5) ?";
    float           vx, vy, vz, x, y, z, lxr, nlx0, lyr, nly0, lzr, nlz0;
    float           c0, cc0, s0, cs0, theta, ctheta, temp1, temp2;
    char            argv[], yn;

    printf("objfile : ");
    scanf("%s", argv);
    printf("noise ? (1/0): ");
    scanf("%d", &noise);
    printf("autodraw ? (1/0): ");
    scanf("%d", &autodraw);
    printf("debug ? (1/0): ");
    scanf("%d", &debug);

    getobjdat(argv);
    start_up_core();
    set_primitive_attributes(&PRIMATTS);
    inquire_color_indices(&vwsurf, 0, 255, cred, cgrn, cblu);
    for (i = 1; i <= 255; i++) {  /* load color LUT */
/*      bred[i] = (float) i *0.003921568;
        bgrn[i] = (float) i *0.003921568;
        bblu[i] = (float) i *0.003921568; */
        bred[i] = (float) i *0.0033064207 + 0.15686272;
        bgrn[i] = (float) i *0.0033064207 + 0.15686272;
        bblu[i] = (float) i *0.0033064207 + 0.15686272;
    }

cycle:
    cred[0] = 0.6;
```

```
cgrn[0] = 0.0;
cblu[0] = 0.6;
define_color_indices(&vwsurf, 0, 255, cred, cgrn, cblu);
set_text_index(255);
/*
 * bred[0] = 60; bgrn[0] = 60; bblu[0] = 60; bred[255] = 40;
 * bgrn[255] = 40; bblu[255] = 40; define_color_indices (&vwsurf, 0,
 * 255, bred, bgrn, bblu);
 * ambient,diffuse,specular,flood,bump,hue,style
 * set_shading_parameters (.01,.96,.0, 0., 7., 0, 0);
 */
style_select();
new_frame();
create_temporary_segment();
move_abs_2(300., 700.);
text("Enter the desired display option");
move_abs_2(330., 650.);
text("1) Still frame");
move_abs_2(330., 620.);
text("2) Rotate the viewer");
move_abs_2(330., 590.);
text("3) Rotate the object [default]");
move_abs_2(330., 560.);
text("4) Rotate the light source");
move_abs_2(330., 530.);
text("5) Quit");
move_abs_2(300., 480.);
text(str);
set_echo_position(KEYBOARD, 1, 0.5, 0.47);
await_keyboard(1000000000, 1, string, &length);
disopt = atoi(string);
close_temporary_segment();
if (disopt == 0)
    disopt = 3;  /* default @@@@@ */
length = 0;
if (debug) {
    printf("\n\n**********************************",
            "****************************************\n");
}
switch (disopt) {
case 1:      /* Still Frame */
    new_frame();
    getxyz(&lx, &ly, &lz, &vx, &vy, &vz);
    set_light_direction(nlx, nly, -nlz);
    setvwpo(vx, vy, vz);
    if (debug) {
        printf("Still frame\n");
        printf("view : %f,%f,%f\n", vx, vy, vz);
        printf("light : %f,%f,%f\n", nlx, nly, nlz);
    }
    create_temporary_segment();
```

```
/* sets all the primitive attributes to their default values */
set_polygon_interior_style(SHADED);
if (!noise) {
    new_frame();
    drawobj();
} else {
    screen = pr_open("/dev/bwtwo1");
    blkscr();
    drawobj();
    width = 560;
    height = 440;
    ax0 = (1152 - width) / 2;
    ay0 = (900 - height) / 2;

    for (j = 0; j < height; j++) {
        for (i = 0; i < width; i = i++) {
            ax = ax0 + i;
            ay = ay0 + j;
            pixel[i][j] = pr_get(screen, ax, ay);
            pr_put(screen, ax, ay, 0);
        }
    }

    blkscr();
    unit = 6.283185307 / width * cycles;
    dseed = 123457;

    ax0 = 8;
    ay0 = 5;

    for (j = 0; j < height; j++) {
        for (i = 0; i < width; i = i++) {
            temp = pixel[i][j];
            vsin = temp + (short) (amp *
                                   sin(unit * (double) (i)));
            vdsin = temp + (short) (amp *
                                    sin((double) i * unit) *
                                    sin((double) j * unit));
            dvalue = ggnqf_(&dseed);
            fvalue = *((float *) &dvalue);
            vnormal = temp + (short) (amp * fvalue);
            ax = ax0 + i;
            ay = ay0 + j;
            pr_put(screen, ax + 576, ay, temp);
            pr_put(screen, ax, ay, vsin);
            pr_put(screen, ax, ay + 450, vdsin);
            pr_put(screen, ax + 576, ay + 450, vnormal);
        }

    }
    scanf("%d", &i);
    pr_close(screen);
    free_raster(&raster);
```

```
        }
        close_temporary_segment();
        do {
            await_keyboard(0, 1, string, &length);
        }
        while (length == 0);
        break;
    case 2:     /* rotate the viewer */
        new_frame();
        getxyz(&lxr, &lyr, &lzr, &vx, &vy, &vz);
        set_light_direction(nlx, nly, -nlz);
        if (debug) {
            printf("light : %f,%f,%f\n", nlx, nly, nlz);
        }
        setvwpo(vx, vy, vz);
        theta = 0.;
        lx = nlx;
        ly = nly;
        lz = nlz;
        x = vx;
        y = vy;
        z = vz;
        if (debug) {
            printf("rotate the viewer\n");
        }
        do {
            new_frame();
            if (debug) {
                printf("view : %f,%f,%f\n", x, y, z);
                printf("light : %f,%f,%f\n", lx, ly, lz);
            }
            create_temporary_segment();
            set_polygon_interior_style(SHADED);
            drawobj();
            close_temporary_segment();
            theta = theta + angle;
            c0 = cos(theta);
            s0 = sin(theta);
            x = vx * c0 + vz * s0;
            y = vy;
            z = -s0 * vx + vz * c0;
            setvwpo(x, y, z);
            lx = nlx * c0 - nlz * s0;
            ly = nly;
            lz = -s0 * nlx - nlz * c0;
            set_light_direction(lx, ly, lz);
            /*
             * for (i = 0; i < nvert; i++) { temp1 = c0 *
             * vertices[i][0] + s0 * vertices[i][2]; temp2 = -s0 *
             * vertices[i][0] + c0 * vertices[i][2];
             * vertices[i][0] = temp1; vertices[i][2] = temp2; }
```

```
            */
            await_keyboard(3000000, 1, string, &length);
        }
        while (length == 0);
        break;
    case 3:      /* rotate the object */
        getxyz(&lxr, &lyr, &lzr, &vx, &vy, &vz);
        set_light_direction(nlx, nly, -nlz);
        if (debug) {
            printf("light : %f,%f,%f\n", nlx, nly, nlz);
        }
        theta = 0.;
        ctheta = 0.;
        nlx0 = nlx;
        nly0 = nly;
        nlz0 = nlz;
        if (debug) {
            printf("rotate the object\n");
        }
        do {
            new_frame();
            c0 = cos(theta);
            s0 = sin(theta);
            cc0 = cos(ctheta);
            cs0 = sin(ctheta);
            x = vx * c0 + vz * s0;
            y = vy;
            z = -s0 * vx + vz * c0;
            setvwpo(x, y, z);
            nlx = nlx0 * cc0 + nlz0 * cs0;
            nly = nly0;
            nlz = -cs0 * nlx0 + nlz0 * cc0;
            /* set_light_direction (nlx, nly, -nlz); */
            if (debug) {
                printf("light : %f,%f,%f\n", nlx, nly, nlz);
                printf("view : %f,%f,%f\n", x, y, z);
            }
            create_temporary_segment();
            set_polygon_interior_style(SHADED);
            drawobj();
            close_temporary_segment();
            theta = theta + angle;
            ctheta = ctheta + angle;
            await_keyboard(3000000, 1, string, &length);
        }
        while (length == 0);
        break;
    case 4:      /* rotate the light source */
        new_frame();
```

```
            getxyz(&lxr, &lyr, &lzr, &vx, &vy, &vz);
            setvwpo(vx, vy, vz);
            if (debug) {
                printf("rotate the light source\n");
                printf("view : %f,%f,%f\n", vx, vy, vz);
            }
            theta = 0.;
            nlx0 = nlx;
            nly0 = nly;
            nlz0 = nlz;

            do {
                new_frame();
                c0 = cos(theta);
                s0 = sin(theta);
                nlx = nlx0 * c0 + nlz0 * s0;
                nly = nly0;
                nlz = -s0 * nlx0 + nlz0 * c0;
                if (debug) {
                    printf("vertices[%d] : (%f,%f,%f) :
                            indxlist = %d\n", i, xlist[i], ylist[i], zlist[i],
                                    indxlist[i]);
                }
                create_temporary_segment();
                set_light_direction(nlx, nly, -nlz);
                set_polygon_interior_style(SHADED);
                drawobj();
                close_temporary_segment();
                theta = theta + angle;
                await_keyboard(3000000, 1, string, &length);
            }
            while (length == 0);
            break;
        case 5:
            shut_down_core();
            exit();
        default:
            goto cycle;
        }
        goto cycle;
    }

    style_select()
    {
        static char     str[] = "Enter your choice (1-6) ?";
        static char     str1[] = "Enter your choice (1-5) ?";
        int             done, segnam, pickid, butnum;
        int             hue, length;

        new_frame();
        setvwpv();
        create_temporary_segment();
        set_text_index(255);
```

```
move_abs_2(300., 700.);
text("Enter the desired shading style");
move_abs_2(330., 650.);
text("1) Wireframe display");
move_abs_2(330., 620.);
text("2) Gray shading [default]");
move_abs_2(330., 590.);
text("3) Gouraud");
move_abs_2(330., 560.);
text("4) Phong diffuse");
move_abs_2(330., 530.);
text("5) Phong specular");
move_abs_2(330., 500.);
text("6) Quit");
move_abs_2(300., 480.);
text(str);
set_echo_position(KEYBOARD, 1, 0.5, 0.47);
await_keyboard(1000000000, 1, string, &length);
close_temporary_segment();
renderstyle = atoi(string) - 1;
/* default */
if (renderstyle == -1)
    renderstyle = 1;
if (renderstyle >= 5) {
    shut_down_core();
    exit();
}
new_frame();
create_temporary_segment();
move_abs_2(300., 700.);
text("Enter the desired shading color");
move_abs_2(330., 650.);
text("1) Gray [default]");
move_abs_2(330., 620.);
text("2) Red");
move_abs_2(330., 590.);
text("3) Green");
move_abs_2(330., 560.);
text("4) Blue");
move_abs_2(330., 530.);
text("5) Yellow");
move_abs_2(300., 480.);
text(str1);
set_echo_position(KEYBOARD, 1, 0.52, 0.47);
await_keyboard(1000000000, 1, string, &length);
move_abs_2(30., 500.);
text("ambient(0.00),diffuse(1.00),specular(0),flood(0),bump(7) ?");
scanf("%f%f%f%f%f", &ambient, &diffuse, &specular, &flood, &bump);
close_temporary_segment();
renderhue = atoi(string) - 1;
if (renderhue == -1)
```

```
        renderhue = 0;   /* default @@@@@ */
    /* ambient,diffuse,specular,flood,bump,hue,style */
    hue = renderhue;
    if (ambient == 0 && diffuse == 0 && specular == 0
          && flood == 0 && bump == 0) {
        ambient = 0.00;
        diffuse = 1.00;
        specular = 0.0;
        flood = 0.0;
        bump = 7.0;
    }
    switch (renderstyle) {
    case 1:
        set_shading_parameters(ambient, diffuse, specular, flood,
                                bump, hue, 0);
        break;
    case 2:
        set_shading_parameters(ambient, diffuse, specular, flood,
                                bump, hue, 1);
        break;
    case 3:
        set_shading_parameters(ambient, diffuse, specular, flood,
                                bump, hue, 2);
        break;
    case 4:
        set_shading_parameters(ambient, diffuse, specular, flood,
                                bump, hue, 2);
        break;
    default:
        break;
    }
}

getxyz(lgtx, lgty, lgtz, vx, vy, vz)
    float          *lgtx, *lgty, *lgtz, *vx, *vy, *vz;
{
    static char     str1[] = "x= ";
    static char     str2[] = "y= ";
    static char     str3[] = "z= ";
    static char     str4[] = "x= ";
    static char     str5[] = "y= ";
    static char     str6[] = "z= ";
    int             length;

    setvwpv();

    new_frame();
    create_temporary_segment();
    move_abs_2(300., 700.);
    text("enter the light source position [default : 0.0,1.0,1.0]");
    move_abs_2(300., 680.);
    text(str1);
    set_echo_position(KEYBOARD, 1, 0.31, 0.665);
```

```
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *lgtx = 0.0;
else
    *lgtx = atof(string);
move_abs_2(300., 660.);
text(str2);
set_echo_position(KEYBOARD, 1, 0.31, 0.645);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *lgty = 0.0;
else
    *lgty = atof(string);
move_abs_2(300., 640.);
text(str3);
set_echo_position(KEYBOARD, 1, 0.31, 0.625);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *lgtz = 0.0;
else
    *lgtz = atof(string);

/* default light direction */
if ((*lgtx == 0.0) && (*lgty == 0.0) && (*lgtz == 0.0)) {
    *lgtx = 0.0;
    *lgty = 1.0;
    *lgtz = 1.0;
}
normlig(*lgtx, *lgty, *lgtz);

move_abs_2(300., 600.);
text("enter the viewer position [default : -1000, 2200, 5000]");
move_abs_2(300., 580.);
text(str4);
set_echo_position(KEYBOARD, 1, 0.31, 0.565);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *vx = 0;
else
    *vx = atof(string);
move_abs_2(300., 560.);
text(str5);
set_echo_position(KEYBOARD, 1, 0.31, 0.548);
await_keyboard(1000000000, 1, string, &length);
if (length == 0)
    *vy = 0.0;
else
    *vy = atof(string);
move_abs_2(300., 540.);
text(str6);
set_echo_position(KEYBOARD, 1, 0.31, 0.527);
await_keyboard(1000000000, 1, string, &length);
```

```
    if (length == 0)
        *vz = 0.0;
    else
        *vz = atof(string);

    /* default view direction */
    if ((*vx == 0.0) && (*vy == 0.0) && (*vz == 0.0)) {
        *vx = -1000.0;
        *vy = 2200.0;
        *vz = 5000.0;
    }
    close_temporary_segment();
}

start_up_core()
{
    int             i;
    float           x, y, z;

    if (initialize_core(DYNAMICC, SYNCHRONOUS, THREED))
        exit(1);

    if (initialize_view_surface(&vwsurf, TRUE))
        exit(2);

    if (select_view_surface(&vwsurf))
        exit(3);

    initialize_device(KEYBOARD, 1);
    set_echo(KEYBOARD, 1, 1);
    set_echo_surface(KEYBOARD, 1, &vwsurf);
    set_keyboard(1, 80, "", 1);
    set_font(1);
}

shut_down_core()
{
    bred[0] = 0.8;
    bgrn[0] = 0.8;
    bblu[0] = 0.8;
    define_color_indices(&vwsurf, 0, 255, bred, bgrn, bblu);
    terminate_device(KEYBOARD, 1);
    deselect_view_surface(&vwsurf);
    terminate_view_surface(&vwsurf);
    terminate_core();
}

int
getobjdat(filename)
    char            *filename;
{
    int             i, j, k;
    short           vtmp, v1, v2, v3;
    float           ftmp, maxd, scale, offset[3];
    float           x, y, z, x0, y0, z0, length;
```

```
FILE            *fptr;
if ((fptr = fopen(filename, "r")) == NULL) {
    printf("Can't open file: %s\n", filename);
    return (1);
}
fscanf(fptr, "%d%d", &nvert, &npoly);
if ((nvert > MAXVERT) || (npoly > MAXPOLY)) {
    printf("Too many object vertices or polygons\n");
    return (2);
}
fscanf(fptr, "%f%f%f%f%f%f", &bbox[0][0], &bbox[0][1], &bbox[1][0],
       &bbox[1][1], &bbox[2][0], &bbox[2][1]);
maxd = 0.0;
for (i = 0; i < 3; i++) {
    offset[i] = (bbox[i][0] + bbox[i][1]) / 2.0;
    bbox[i][0] -= offset[i];
    bbox[i][1] -= offset[i];
    if (bbox[i][0] > bbox[i][1]) {
        ftmp = bbox[i][0];
        bbox[i][0] = bbox[i][1];
        bbox[i][1] = ftmp;
    }
    if (maxd < bbox[i][1])
        maxd = bbox[i][1];
}
scale = 1000.0 / maxd;
for (i = 0; i < 3; i++) {
    bbox[i][0] *= scale;
    bbox[i][1] *= scale;
}
for (i = 0; i < nvert; i++) {
    fscanf(fptr, "%f%f%f", &vertices[i][0], &vertices[i][1],
           &vertices[i][2]);
    vertices[i][0] = (vertices[i][0] - offset[0]) * scale;
    vertices[i][1] = (vertices[i][1] - offset[1]) * scale;
    vertices[i][2] = (vertices[i][2] - offset[2]) * scale;
    normal[i][0] = 0.0;
    normal[i][1] = 0.0;
    normal[i][2] = 0.0;
    normalcount[i] = 0;
}
k = 0;
for (i = 0; i < npoly; i++) {
    fscanf(fptr, "%d", &npvert[i]);
    if ((k + npvert[i]) > MAXPVERT) {
        printf("Too many polygon vertices\n");
        return (3);
    }
    pvertptr[i] = &pvert[k];
    for (j = 0; j < npvert[i]; j++) {
        fscanf(fptr, "%hd", &vtmp);
```

```
            pvert[k++] = vtmp - 1;
        }
        planeq[i][0] = planeq[i][1] = planeq[i][2] = planeq[i][3] = 0.0;
        v1 = pvert[k - 1];
        v2 = pvert[k - 2];
        v3 = pvert[k - 3];
        for (j = 0; j < 3; j++) {
            planeq[i][0] += -vertices[v1][1] *
                (vertices[v2][2] - vertices[v3][2]);
            planeq[i][1] += -vertices[v1][0] *
                (vertices[v3][2] - vertices[v2][2]);
            planeq[i][2] += -vertices[v1][0] *
                (vertices[v2][1] - vertices[v3][1]);
            planeq[i][3] += -vertices[v1][0] *
                ((vertices[v3][1] * vertices[v2][2]) -
                 (vertices[v2][1] * vertices[v3][2]));
            vtmp = v1;
            v1 = v2;
            v2 = v3;
            v3 = vtmp;
        }
/* The normal of this face points to the center of the object. */
/*       if (planeq[i][3] > 0.0)
                    for (j = 0; j < 3; j++)
                planeq[i][j] = -planeq[i][j]; */
        x = planeq[i][0];
        y = planeq[i][1];
        z = planeq[i][2];
        length = sqrt(x * x + y * y + z * z);
        planeq[i][0] /= length;
        planeq[i][1] /= length;
        planeq[i][2] /= length;
               planeq[i][3] /= length;

        if (debug) {
            printf("planeq[%d] : %f,%f,%f,%f\n", i, planeq[i][0],
                        planeq[i][1], planeq[i][2], planeq[i][3]);
        }
        for (j = 1; j <= npvert[i]; j++) {    /* accum normls */
            vtmp = pvert[k - j];
            normal[vtmp][0] += planeq[i][0];
            normal[vtmp][1] += planeq[i][1];
            normal[vtmp][2] += planeq[i][2];
            normalcount[vtmp]++;
        }
    }
    for (i = 0; i < nvert; i++) {
        normal[i][0] /= normalcount[i];
        normal[i][1] /= normalcount[i];
        normal[i][2] /= normalcount[i];
```

```
            if (debug) {
                printf("normal[%d] : %f,%f,%f\n", i, normal[i][0],
                            normal[i][1], normal[i][2]);
            }
        }
        fclose(fptr);
        return (0);
}

setvwpo(vx, vy, vz)
        float           vx, vy, vz;
{
        int             i;
        float           diag, del, objdist, near;

        set_view_reference_point(vx, vy, vz);
        set_view_plane_normal(-vx, -vy, -vz);
        set_projection(PERSPECTIVE, 0., 0., 0.);
        set_view_plane_distance(256.0);
        if ((vx == 0.0) && (vz == 0.0))
            set_view_up_3(0.0, 0.0, vy);
        else
            set_view_up_3(0.0, 1.0, 0.0);
        set_window(-80.0, 80.0, -80.0, 80.0);
        diag = 0.0;
        for (i = 0; i < 3; i++) {
            del = bbox[i][1] - bbox[i][0];
            diag += del * del;
        }
        diag = sqrt(diag) / 2.0;
        objdist = sqrt(vx * vx + vy * vy + vz * vz);
        near = (diag >= objdist) ? objdist / 2.0 : objdist - diag;
        set_view_depth(near, objdist + diag);
        /*
         * set_window_clipping (FALSE); set_front_plane_clipping (FALSE);
         * set_back_plane_clipping (FALSE);
         */
        set_viewport_3(.125,.874, 0.,.749, 0.0, 1.0);
}

static float    invxform[4][4];

drawobj()
{
        int             i;
        float           x, y, z, x0, y0, z0, length;
        char            ch;

        bred[0] = 0.6;
        bgrn[0] = 0.0;
        bblu[0] = 0.6;
        cred[0] = 0.6;
        cgrn[0] = 0.0;
```

```
    cblu[0] = 0.6;
    if (renderhue == 0) {
        set_text_index(1);
        define_color_indices(&vwsurf, 0, 255, bred, bgrn, bblu);
    } else {
        set_text_index(1);
        define_color_indices(&vwsurf, 0, 255, cred, cgrn, cblu);
    }
    /* if (renderstyle && renderstyle < 3) */
    for (i = 0; i < npoly; i++) {
        cpindex[i] =
            (ambient + diffuse * (planeq[i][0] * nlx + planeq[i][1] *
                  nly + planeq[i][2] * nlz)) * 254.;

    }
    for (i = 0; i < nvert; i++) {
        cindex[i] =
            (normal[i][0] * nlx + normal[i][1] * nly +
                 normal[i][2] * nlz) * 254.;
        if (cindex[i] < 4)
            cindex[i] = 4;
        if (cindex[i] > 248)
            cindex[i] =
                248;
    }
    /* inquire_inverse_composite_matrix(invxform); */
/*  if (renderstyle == 3)
        set_zbuffer_cut(&vwsurf, xcut, zcut, 2);
    else
        set_zbuffer_cut(&vwsurf, xcut, zcut0, 2); */
    for (i = 0; i < npoly; i++) {
        /* if ((visible(planeq[i])) || (renderstyle == 0)) */
        drawface(i);

    }
}

void
normlig(lgtx, lgty, lgtz)
    float           lgtx, lgty, lgtz;

{
    float           x0, y0, z0, length;

/*  map_ndc_to_world_3(lgtx, lgty, lgtz, &nlx, &nly, &nlz);
    map_ndc_to_world_3(0.0, 0.0, 0.0, &x0, &y0, &z0);
    printf("lgtx = %f : nlx = %f\n", lgtx, nlx - x0);
    printf("lgty = %f : nly = %f\n", lgty, nly - y0);
    printf("lgtz = %f : nlz = %f\n", lgtz, nlz - z0); */
    x0 = 0.0;
    y0 = 0.0;
    z0 = 0.0;
```

```
    lgtx -= x0;
    lgty -= y0;
    lgtz -= z0;
    length = sqrt(lgtx * lgtx + lgty * lgty + lgtz * lgtz);
    if (length != 0.0) {
        nlx = lgtx / length;
        nly = lgty / length;
        nlz = lgtz / length;
    }
}

int
visible(pln)
    float           pln[];
{
    int             i;
    float           c;

    c = 0.0;
    for (i = 0; i < 4; i++)
        c += invxform[2][i] * pln[i];
    return (c > 0.0);
}

drawface(p) int
    p;
{
    int             i, j, k;
    short          *ptr;
    float           para = 215.0 / 255.0;

    if (debug) {
        printf("face : %d\n", p);
    }
    ptr = pvertptr[p];
    for (i = 0; i < npvert[p]; i++) {
        j = *ptr++;
        xlist[i] = vertices[j][0];
        ylist[i] = vertices[j][1];
        zlist[i] = vertices[j][2];
        if (renderstyle < 3) {
            indxlist[i] = cindex[j];
            if (debug) {
                printf("vertices[%d] : (%f,%f,%f) : indxlist = %d\n",
                                i, xlist[i], ylist[i], zlist[i], indxlist[i]);
            }
        } else {
            dxlist[i] = normal[j][0];
            dylist[i] = normal[j][1];     /* Phong */
            dzlist[i] = normal[j][2];
        }
    }
```

```
    if (renderstyle < 2) {
        j = (cpindex[p] > 0) ? cpindex[p] : 1;
        j = (j > 255) ? 255 : j;
/*      j = (int) (para * (float) j) + 40; */
        if (debug) {
            printf("indxlist[%d] : %d\n", p, j);
        }
        if (renderhue) {
            j = j >> 2;  /* if (j > 62) j = 62; other colars */
            if (!autodraw) {
                if (debug) {
                    scanf("%d", &j);
                } else
                    do {
                        await_keyboard(0, 1, string, &i);
                    }
                    while (i == 0);
            }
            set_fill_index(j + renderhue * 64 - 63);
        } else {
            if (!autodraw) {
                if (debug) {
                    scanf("%d", &j);
                } else
                    do {
                        await_keyboard(0, 1, string, &i);
                    }
                    while (i == 0);
            }
            set_fill_index(j);   /* gray colar */
        }
    } else if (renderstyle == 2) {
        set_vertex_indices(indxlist, npvert[p]);
    }
     /* Gouraud */
    else {
        set_vertex_normals(dxlist, dylist, dzlist, npvert[p]);
    }           /* Phong */
    if (renderstyle == 0)
        polyline_abs_3(xlist, ylist, zlist, npvert[p]);   /* wireframe */
    else
        polygon_abs_3(xlist, ylist, zlist, npvert[p]);
}

static float    maxvw, vwpp, maxvdim;
static float    vlx, vby, vfz, vdx, vdy, vdz;
static float    minleft, maxright, minbot, maxtop, minback, maxfront;

setvwpv()
{
    set_view_reference_point(0.0, 0.0, 0.0);
    set_view_plane_normal(0.0, 0.0, -1.0);
```

```
    set_view_plane_distance(0.0);
    set_projection(PARALLEL, 0.0, 0.0, 1.0);
    set_view_up_3(0.0, 1.0, 0.0);
    set_window(0.0, 1023.0, 0.0, 767.0);
    set_view_depth(0.0, 1.0);
    set_window_clipping(FALSE);
    set_viewport_3(0.0, 1., 0.0,.75, 0.0, 1.);
    set_text_index(255);
}

void
blkscr()
{
    int             i, j;

    static float    x[4] = {
                80., -80., -80., 80.
    };
    static float    y[4] = {
                80., 80., -80., -80.
    };
    static float    z[4] = {
                0., 0., 0., 0.
    };
    static int      n = 4;

    for (j = 0; j < 900; j++) {
        for (i = 0; i < 1152; i = i++) {
            pr_put(screen, i, j, 1);
        }
    }
    set_line_index(1);
    set_fill_index(1);
    set_linewidth(0.);
    polygon_abs_2(x, y, n);
}
```

APPENDIX D: C PROGRAMS FOR IMAGE PROCESSING USING THE 3M VDL VISION SYSTEM

```
/* PROGRAM FT.C - This program calculates the Fourier Transformation
        of the input image.

    Writtern by Chirs K. Wu, Jun, 1987.
*/

#include <stdio.h>
#include <math.h>
#include <strings.h>
#include <ci.h>
#define SIZE 64
#define M 6

extern vdlinit(),vdlterm();
extern long sci();

main(ac,av)
int ac;char *av[];
{
FILE *fopen(),*fp1,*fp2,*fp3,*fp4,*fp5;
char cmd[50],filere[50],fileim[50];
int ulx1,uly1;
int i,j,ti,tj,nt;
float rebuf[SIZE],imbuf[SIZE];

if(ac!= 2) {
     printf("Usage: %s filename\n",av[0]);
     printf("This program calculates the FT of the input\n");
     printf("image. The results are stored in the following\n");
     printf("files:\n");
     printf("re_filename - the real part of the FT\n");
     printf("im_filename - the imaginary part of the FT\n");
     exit(-1);
     }
if (vdlinit() < 0) {
     printf("unable to initialize VDL\n");
     exit(-1);
     }
sci(pinp,(COORD)1,0);
sci(pcop,(COORD)2,0,1);
sci(pbuf,(COORD)2,-2,1);
sci(pwin,(COORD)1,-1);
sci(pwin,(COORD)1,0);
sci(pbuf,(COORD)2,-2,0);
printf("Due to the stack size limitation, the size of the image to be\n");
printf("processed is restricted to 64x64. \n\n");
printf("Enter the coordinates of the upper left corner of picture : ");
scanf("%d %d",&ulx1,&uly1);
fp1=fopen("fttemp.re","w");
fp2=fopen("fttemp.im","w");
fp5=fopen("wu.pic","w");

/*perform FFT to each row*/

for (i=uly1;(ti=i-uly1)<SIZE;i++) {
     for (j=ulx1;(tj=j-ulx1)<SIZE;j++) {
          nt=ti+tj;
          imbuf[tj]=0;
          rebuf[tj]=(float)pw(-1,nt)
               *(float)(sci(pget,(COORD)3,(long)j,(long)i,1)+128);
```

```
        }
    fwrite(rebuf,4,SIZE,fp5);
    fft2(rebuf,imbuf,SIZE,M);
    fwrite(rebuf,4,SIZE,fp1);
    fwrite(imbuf,4,SIZE,fp2);
    }
fclose(fp1);fclose(fp2);fclose(fp5);
sprintf(cmd,"/d0/user/flipreal fttemp.re %d",SIZE);
system(cmd);
sprintf(cmd,"/d0/user/flipreal fttemp.im %d",SIZE);
system(cmd);

/* perform FFT to each column */

fp1=fopen("fttemp.re","r");
fp2=fopen("fttemp.im","r");
sprintf(filere,"re_");
strncat(filere,av[1],10);
fp3=fopen(filere,"w");
sprintf(fileim,"im_");
strncat(fileim,av[1],10);
fp4=fopen(fileim,"w");
for (i=0;i<SIZE;i++) {
    fread(rebuf,4,SIZE,fp1);
    fread(imbuf,4,SIZE,fp2);
    fft2(rebuf,imbuf,SIZE,M);
    fwrite(rebuf,4,SIZE,fp3);
    fwrite(imbuf,4,SIZE,fp4);
    }
fclose(fp1);fclose(fp2);fclose(fp3);fclose(fp4);
sprintf(cmd,"/d0/user/flipreal %s %d",filere,SIZE);
system(cmd);
sprintf(cmd,"/d0/user/flipreal %s %d",fileim,SIZE);
system(cmd);
unlink("fttemp.re");
unlink("fttemp.im");
if (vdlterm()<0) {
    printf("unable tp terminate VDL\n");
    exit(-1);
    }
}

pw(x,n)
int x,n;
{
int i,p=1;
if (n==0) return(1);
else {
    for (i=0;i<n;i++) p=p*x;
    return(p);
    }
}
```

```
/*SUBROUTINE FFT2 - Cooley-Turkey radix-2 FFT algorithm
Chris K. Wu.
*/
#include <stdio.h>
#include <math.h>
fft2(x,y,n,m)
float *x,*y; int n,m;
{
int i,ie,ia,j,k,l,l1,n1,n2;
float c,s,xt,yt;
double pi,wr[512],wi[512];
pi=6.28319/(double)n;
for(i=0;i<n;i++) {
     wr[i]=cos(pi*(double)i);
     wi[i]=sin(pi*(double)i);
     }
n2=n;
for(i=0;i<m;i++) {
     n1=n2;
     n2=n2/2;
     ie=n/n1;
     ia=0;
     for(j=0;j<n2;j++) {
          c=(float)wr[ia];
          s=(float)wi[ia];
          ia=ia+ie;
          for(k=j;k<n;k=k+n1) {
               l=k+n2;
               xt=x[k]-x[l];
               x[k]=x[k]+x[l];
               yt=y[k]-y[l];
               y[k]=y[k]+y[l];
               x[l]=c*xt+s*yt;
               y[l]=c*yt-s*xt;
               }
          }
     }
 /*digit reverse counter*/
 j=1;
 for(i=1;i<n;i++) {
      if (j>i) {
           l=i-1;
           l1=j-1;
           xt=x[l1];
           x[l1]=x[l];
           x[l]=xt;
           xt=y[l1];
           y[l1]=y[l];
           y[l]=xt;
           }
      k=n/2;
      while (j>k) {
           j=j-k;
           k=k/2;
           }
      j=j+k;
      }
 return(0);
 }
```

```
/*
SUBROUTINE FLIPREAL-This routine flips the input floating point
                    raster along the diagonal axis for the 2-D
                    FFT algorithm.
written by Chris K. Wu
*/

#include <stdio.h>

main(ac,av)
int ac;char *av[];
{
char temph[20],cmd[80];
int size;
if (ac!=3) {
     printf("Usage: %s raster size\n",av[0]);
     exit(-1);
     }
size=atoi(av[2])/2;
sprintf(temph,"frtemp");
dorotate(av[1],temph,size);
unlink(av[1]);
sprintf(cmd,"rename frtemp %s",av[1]);
system(cmd);
exit(0);
}

/*
SUBROUTINE DOROTATE() -- Handle rotations of the raster file
*/
dorotate(raster,temph,size)
char *raster,*temph; int size;
{
int i,j;
float buf1[32][64],buf2[32][32],temp[64],dt;
FILE *fopen(),*fp,*fp1;
fp=fopen(raster,"r");
for (i=0;i<size;i++) {
     fread(buf1[i],4,size,fp);
     fread(buf2[i],4,size,fp);
     }
fp1=fopen("rotemp","w");
for(i=0;i<size;i++)
     for(j=i+1;j<size;j++) {
          dt=buf1[i][j];
          buf1[i][j]=buf1[j][i];
          buf1[j][i]=dt;
          }
for (i=0;i<size;i++) {
     fwrite(buf1[i],4,size,fp1);
     fwrite(buf2[i],4,size,fp1);
     }
for (i=0;i<size;i++) {
     fread(buf1[i],4,size,fp);
     fread(buf2[i],4,size,fp);
     }
fclose(fp);
for(i=0;i<size,i++)
     for(j=i+1;j<size,j++) {
          dt=buf2[i][j];
```

```
            buf2[i][j]=buf2[j][i];
            buf2[j][i]=dt;
            }
    for (i=0;i<size;i++) {
         fwrite(buf1[i],4,size,fp1);
         fwrite(buf2[i],4,size,fp1);
         }
    fclose(fp1);
    fp=fopen("rotemp","r");
    for(i=0;i<size;i++) {
         fread(temp,4,size,fp);
         fread(buf2[i],4,size,fp);
         }
    for(i=0;i<size;i++)
         for(j=0;j<size;j++) {
              dt=buf1[i][j];
              buf1[i][j]=buf2[j][i];
              buf2[j][i]=dt;
              }
    fclose(fp);
    fp=fopen("rotemp","r");
    fp1=fopen(temph,"w");
    for(i=0;i<size;i++) {
         fread(temp,8,size,fp);
         fwrite(temp,4,size,fp1);
         fwrite(buf2[i],4,size,fp1);
         }
    for(i=0;i<size;i++) {
         fread(temp,8,size,fp);
         for (j=0;j<size;j++) temp[j]=buf1[i][j];
         fwrite(temp,8,size,fp1);
         }
    fclose(fp);
    fclose(fp1);
    unlink("rotemp");
    return(0);
    }
```

```
/* PROGRAM SPECTRUM.C - This program takes the outputs from
     the FT.C program and creats a data file containing the
     spectrum of the image.

Writtern by Chris K  Wu, Jun, 1987.
*/

#include <stdio.h>
#include <math.h>
#include <strings.h>
#define SIZE 64
main(ac,av)
int ac;char *av[];
{
char filere[50],fileim[50];
FILE *fopen(),*fp1,*fp2,*fp3;
int i,j;
long iad[SIZE];
float scale,fmax;
float rd[SIZE],id[SIZE],fad[SIZE];
double dad;
if (ac!=2) {
     printf("Usage: %s filename \n",av[0]);
     printf("filename-output data filename from the FT program.\n");
     printf("The output data is stored in file sp_filename.\n");
     exit(-1);
     }
fmax=(-128.);
sprintf(filere,"re_");
strncat(filere,av[1]);
sprintf(fileim,"im_");
strncat(fileim,av[1]);
fp1=fopen(filere,"r");
fp2=fopen(fileim,"r");
fp3=fopen("sptemp","w");
for(i=0;i<SIZE;i++) {
     fread(rd,4,SIZE,fp1);
     fread(id,4,SIZE,fp2);
     for(j=0;j<SIZE;j++) {
          dad=(double)(rd[j]*rd[j]+id[j]*id[j]);
          fad[j]=(float)sqrt(dad);
          if(fad[j]>fmax) fmax=fad[j];
          }
     fwrite(fad,4,SIZE,fp3);
     }
dad=(double)fmax+1.;
scale=256./(float)log(dad);
fclose(fp1);fclose(fp2);fclose(fp3);
sprintf(filere,"sp_");
strncat(filere,av[1]);
fp1=fopen("sptemp","r");
fp2=fopen(filere,"w");
for(i=0;i<SIZE;i++) {
     fread(fad,4,SIZE,fp1);
     for(j=0;j<SIZE;j++) {
          dad=(double)fad[j]+1.;
          iad[j]=(long)(scale*(float)log(dad))-128;
          }
     fwrite(iad,4,SIZE,fp2);
     }
```

```
unlink("sptemp");
exit(0);
}
```

```
/* PROGRAM CREATPIC.C - This program plots the spectrum of the
     image on the monitor
Writtern by Chris K. Wu  Jun, 1987
*/

#include <ci.h>
#include <stdio.h>
#define SIZE 64

extern vdlinit();
extern long sci();
extern vdlterm();

main(argc,argv,envp)
int argc;
char *argv[],*envp[];
{
int i,j,ti,tj,xs,ys;
long buf[SIZE];
FILE *fopen(),*fp;
if (argc != 2) {
     printf("Usage: %s sp_filename\n",argv[0]);
     exit(-1);
     }
if (vdlinit() < 0) {
     fprintf(stderr,"unable to initialize VDL\n");
     return(-1);
     }
xs=121;
ys=121;
sci(pwin,(COORD)1,0);
sci(pcle,(COORD)1,0);
sci(pbuf,(COORD)2,-2,0);
fp=fopen(argv[1],"r");
for(j=ys;(tj=(j-ys)/2)<SIZE;j=j+2) {
     fread(buf,4,SIZE,fp);
     for(i=xs;(ti=(i-xs)/2)<SIZE;i=i+2) {
          if(buf[ti]>127) buf[ti]=127;
          if(buf[ti]<-128) buf[ti]=(-128);
          sci(pput,(COORD)4,(long)i,(long)j,buf[ti],1);
          sci(pput,(COORD)4,(long)i,(long)j+1,buf[ti],1);
          sci(pput,(COORD)4,(long)i+1,(long)j,buf[ti],1);
          sci(pput,(COORD)4,(long)i+1,(long)j+1,buf[ti],1);
          }
     }
fclose(fp);
sci(pbuf,(COORD)2,-2,0);
sci(pwin,(COORD)4,100,100,263,263);
sci(pout,(COORD)1,0);
if (vdlterm() < 0) {
     fprintf(stderr,"unable to terminate VDL\n");
     return(-1);
     }
}
```

```
/* PROGRAM PATTERN.C - This program identify a 2-D geometry from
        a library of geometries. The geometric properties, area,
        perimeter, and maximum radius, of the viewed geometry are
        measured and used to check against the library set.

   Writtern by Chris K. Wu, May, 1987.
*/

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

extern vdlinit(),vdlterm();
extern long sci();

int window[4],threshold,count;
int area_arr[10],peri_arr[10],mr_arr[10];
char name_arr[10][50];

main()
{

long v;
int area,peri,mr,c;
int cx,cy,mx,my,temp;
double distsq;

if (vdlinit() < 0) {
    printf("unable to initialize VDL");
    exit(-1);
    }
printf("place any object inside view area to ");
printf("determine the view window\n");
sci(pdig,(COORD)1,1);
sci(pwin,(COORD)1,-1);
printf("perform learning process (y/n)? ");
while ( (c = getchar()) != 'y' && c != 'Y' && c != 'n' && c != 'N' )
    ;
getchar();
if ( c == 'y' || c == 'Y' )
    learn_sub();
else
    getdata();
printf("place the unknown geometry inside the view window\n");
start:
sci(pdig,(COORD)1,1);
sci(pthr,(COORD)1,(long)threshold);
sci(pfrq,(COORD)0);
sci(prng,(COORD)2,-128,0);
v = sci(prpc,(COORD)0);
area= (int) v;
printf("the area of the viewed geometry is %d",area);
printf(" pixels\n\n");
sci(pbin,(COORD)0);
sci(pfrq,(COORD)0);
sci(prng,(COORD)2,0,127);
v = sci(pcen,(COORD)0);
cy = (int)v / 512;
cx = (int)v - (cy*512);
v = sci(prpc,(COORD)0);
```

```
peri = (int) v;
printf("the perimeter of the viewed geometry is %d",peri);
printf(" pixels\n\n");
v = sci(pmnx,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
mr = (int) sqrt(distsq);
v = sci(pmny,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
temp = (int) sqrt(distsq);
if (temp > mr) mr=temp;
v = sci(pmxx,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
temp = (int) sqrt(distsq);
if (temp > mr) mr=temp;
v = sci(pmxy,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
temp = (int) sqrt(distsq);
if (temp > mr) mr=temp;
printf("the maximum radius of the viewed geometry is %d",mr);
printf(" pixels\n\n");
recog(area,peri,mr);
printf("identify next unknown geometry (y/n)? ");
while ( (c = getchar()) != 'y' && c != 'Y' && c != 'n' && c != 'N' )
    ;
getchar();
if ( c == 'y' || c == 'Y' )
    goto start;
else
    if (vdlterm() < 0) {
        printf("unable to terminate VDL\n");
        exit(-1);
        }
}

/* subroutine to learn the geometric properties of
   the target objects */

learn_sub()
{

int i,c,cx,cy,mx,my,temp;
long v;
double distsq;

printf("use the numerical key pad to move the ");
printf("cursor to determint the threshold value\n");
printf("press <return> key to exit\n");
sci(pcrs,(COORD)0);
printf("enter the threshold value ");
scanf("%d",&threshold);
printf("threshold value is set to %d\n",threshold);
count=1;
```

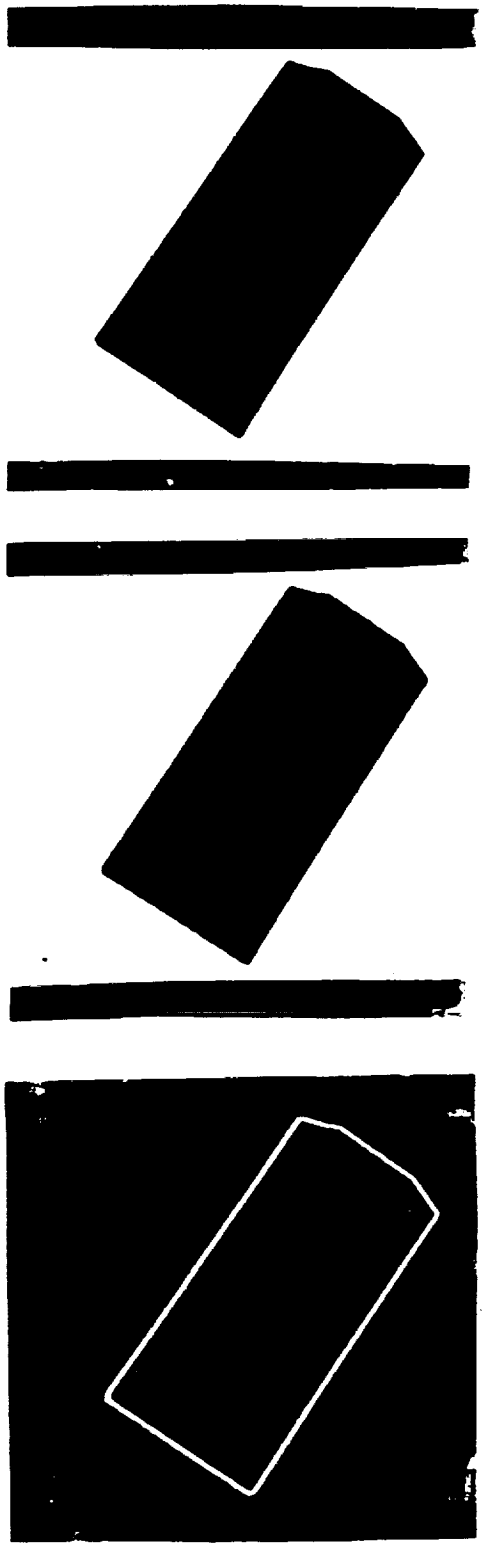
```
loop
printf("place the geometry inside the view window and\n");
printf("input the given name of the viewed geometry\n");
scanf("%s",name_arr[count]);
sci(pdig,(COORD)1,1);
sci(pthr,(COORD)1,(long)threshold);
sci(pfrq,(COORD)0);
sci(prng,(COORD)2,-128,0);
v = sci(prpc,(COORD)0);
area_arr[count] = (int) v;
printf("the area of the viewed geometry is %d",area_arr[count]);
printf(" pixels\n\n");
sci(pbin,(COORD)0);
sci(pfrq,(COORD)0);
sci(prng,(COORD)2,0,127);
v = sci(pcen,(COORD)0);
cy = (int)v / 512;
cx = (int)v - (cy*512);
v = sci(prpc,(COORD)0);
peri_arr[count] = (int) v;
printf("the perimeter of the viewed geometry is %d",peri_arr[count]);
printf(" pixels\n\n");
v = sci(pmnx,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
mr_arr[count] = (int) sqrt(distsq);
v = sci(pmny,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
temp = (int) sqrt(distsq);
if (temp > mr_arr[count]) mr_arr[count]=temp;
v = sci(pmxx,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
temp = (int) sqrt(distsq);
if (temp > mr_arr[count]) mr_arr[count]=temp;
v = sci(pmxy,(COORD)0);
my = (int)v/512;
mx = (int)v - my*512;
distsq = (cx-mx)*(cx-mx) + (cy-my)*(cy-my);
temp = (int) sqrt(distsq);
if (temp > mr_arr[count]) mr_arr[count]=temp;
printf("the maximum radius of the viewed geometry is %d",mr_arr[count]);
printf(" pixels\n\n");
printf("next geometry (y/n)? ");
while ( (c = getchar()) != 'y' && c != 'Y' && c != 'n' && c != 'N' )
    ;
getchar();
if ( c == 'y' || c == 'Y' ) {
    ++count;
    goto loop;
    }
else
    savedata();
return(0);
}
```

```
/* subroutine to save data to file "pattern dat" */
savedata()
{

int i;
FILE *fopen(),*fp;

fp = fopen("/d0/user/pattern.dat","w");
fprintf(fp,"%d\n",threshold);
fprintf(fp,"%d\n",count);
for ( i=1;i<count+1,i++) {
     fprintf(fp,"%s\n",name_arr[i]);
     fprintf(fp,"%d %d %d\n",area_arr[i],peri_arr[i],mr_arr[i]);
     }
fclose(fp);
}

/* subroutine to read the geometric properties of the
   learned objects from the data file */

getdata()
{

int i;
FILE *fopen(),*fp;

fp = fopen("/d0/user/pattern.dat","r");
fscanf(fp,"%d",&threshold);
fscanf(fp,"%d",&count);
for ( i=1;i<count+1,i++) {
     fscanf(fp,"%s",name_arr[i]);
     fscanf(fp,"%d %d %d",&area_arr[i],&peri_arr[i],&mr_arr[i]);
     }
fclose(fp);
}

/* subroutine to identify the viewed geometry against
   the library geometry */

recog(area,peri,mr)
int area,peri,mr;
{

int i,c,rec_flag=0;
char *rec_name1,*rec_name2;
double cdsq,cd,err,rec_err1=1,rec_err2;

for (i=1;i<count+1;i++) {
     cdsq = (double)area/(double)area_arr[i];
     cd = sqrt(cdsq);
     err = 0.5 * fabs(peri-cd*peri_arr[i])/(cd*peri_arr[i]);
     err = err + 0.5 * fabs(mr-mr_arr[i]*cd)/(mr_arr[i]*cd);
     printf("match the viewed geometry to %s\n",name_arr[i]);
     printf("   %7.2f percent in difference\n\n",err*100.);
     if ( err < 0.1 )
          if ( rec_flag != 0 ) {
               if (err < rec_err1) {
                    rec_err2 = rec_err1;
                    rec_name2 = rec_name1;
                    rec_err1 = err;
```

```
                    rec_name1 = name_arr[i];
                    }
                else {
                    rec_err2 = err;
                    rec_name2 = name_arr[i];
                    }
                rec_flag = 2;
                }
            else {
                rec_err1 = err;
                rec_name1 = name_arr[i];
                rec_flag = 1;
                }
        }
    if ( rec_flag != 0 ) {
        printf("\n\nthe geometry is recognized as \n\n *** %s ***",rec_name1),
        printf("    with confidance of %5.1f percent\n\n",(1.-rec_err1)*100);
        if ( rec_flag == 2 )
            printf(" *** %s ***    with confidance of %5.1f percent\n\n",
            rec_name2,(1.-rec_err2)*100);
        }
    else {
        printf("can't recognize the geometry.\n");
        printf("want to store the geometry into library (y/n)? ");
        while ( (c = getchar()) != 'y' && c != 'Y' && c != 'n' && c != 'N' )
        ,
        if ( c == 'y' || c == 'Y' ) {
            count ++;
            printf("input the given name of the geometry\n");
            scanf("%s",name_arr[count]);
            area_arr[count] = area;
            peri_arr[count] = peri;
            mr_arr[count] = mr;
            savedata();
            }
        }
    return(0);
    }
```

The macro 'XYANGLE.M'

```
win(50,70,350,450)          /* set up the processing window */
dig( )                      /* acquire image */
thr (-20)                   /* convert into binary image */
bin ( )                     /* binary edge */
rng (110,127)               /* specify the intensity range */
v=mxx( )                    /* the rightmost pixel within the intensity range */
ry=v/512
rx=v-(ry*512)
v=mxy( )                    /* the bottommost pixel */
by = v/512
bx=v-(by*512)
rng(110,127)
v=cen( )                    /* locate the gravity center */
cy=v/512
cx=v-(cy*512)
print "the gravity center is located at\n "
print "x = "
print cx
print " \n"
print "y ="
print cy
print " \n"
den=rx-bx                   /* compute the orientation angle */
num=by-ry
print "tan(theta) = "
print num
print "/"
print den
print " \n"
```



(a) Digital picture output from the 3M system (Black hexagox box witn $\theta_z = 60^0$)

(b) Binary image resulted from the thresholding process (thr()) on picture a.

(c) Wireframe resulted from the binary edge detection process (bin()) on picture b.

Figure 1 Intermediate results of the macro 'xyangle.m'

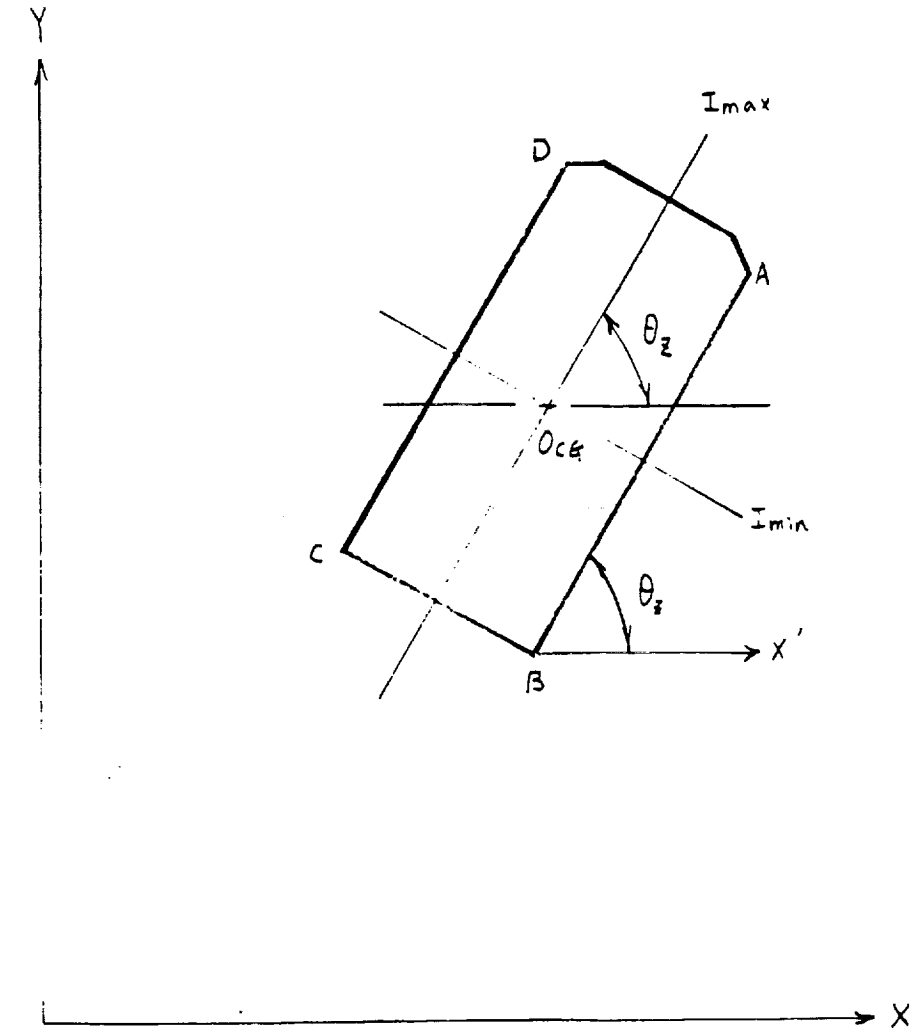


Figure 2 Image space and the principal axes of area moment of inertia about the gravity center

Report

to

RESEARCH INSTITUTE FOR

COMPUTING AND INFORMATION SYSTEMS

(RICIS)

ON

COMPUTER GRAPHICS TESTBED TO SIMULATE AND TEST VISION SYSTEMS FOR SPACE APPLICATIONS

BY

J. B. CHEATHAM

RICIS RESEARCH ACTIVITY AI.2

GRANT NCC9-16

MECHANICAL ENGINEERING AND MATERIALS SCIENCE DEPARTMENT

RICE UNIVERSITY

HOUSTON, TX 77251-1892

JULY 1991

Executive Summary

Objectives:

The objectives of this project have shifted from computer graphics and vision systems to the broader scope of applying concepts of artificial intelligence to robotics. Specifically, the research is directed toward developing artificial neural networks, expert systems and Laser Imaging techniques for autonomous space robots.

The statement of work is

1. Develop a Computer Graphics laser range finder simulator

2. Use laser imaging simulator to study use of artificial neural networks for antonomous robotic navigation.

Overview:

Primarily as a result of our contacts with Dr. Timothy Cleghorn through this RICIS project the emphasis of much of our robotics research has changed to research directed toward artificial intelligence applications in robotics. We have become interested in applications of CLIPS, NETS and Fuzzy Control.

A laser range finder simulator has been developed and used to study use of artificial neural nets for robot navigation. CLIPS and NETS have worked their way into our robotics courses and we are currently investigating an application of NETS in aerodynamics. During the past two years there have been 5 MS and 5 PhDs degrees awarded to our students in robotics. Three MS and three PhD theses are directly related to this project and are summarized in this report. Also a number of technical papers and reports that have been written during the duration of this project are listed at the end of this report.

Table of Contents

I. Introduction

This report describes recent robotic activities in Mechanical Engineering at Rice that have been stimulated by RICIS Research Activity AI.2. This work has been directed toward developing artificial neural networks, expert systems and Laser Imaging techniques for autonomous space robotics applications. A computer graphics laser range finder simulator developed by Wu has been used by Weiland and Norwood to study use of artificial neural networks for path planning and obstacle avoidance.

The support provided by NASA/JSC Software Technology Branch through RICIS has been leveraged significantly through support of 5 graduate students by the U. S. Army (Captains Norwood, Weiland, Schuster, Atkinson and Hanusa).

Theses research by Wu, Norwood, Weiland and Schuster is summarized below. An application of NETS in the advanced robotics lab course is described and the potential for NETS in aerodynamics is discussed. Our recent interest in fuzzy logic control for redundant manipulators and mobile robot navigation is mentioned. Finally a listing of recent technical papers, reports and theses is given.

II. Laser Imaging System Simulator

A computer graphics simulation was developed to permit emulation of laser range scanners for use in our robotics research. Details of the simulation are contained in the PhD thesis by Chris Wu (1990). A brief summary of his thesis is given in this section.

The Laser Imaging Simulation Algorithm (LISA) provides simulated laser images using the system parameters entered by the user. All parameters (such as hardware specifications, output data size, and viewing parameters) are changeable to permit emulation of the ERIM and Odetics laser scanners as well as future scanners. LISA computes system characteristics such as signal-noise ratio, ambiguity interval and viewing volume and then generates appropriate images.

Traditional computer graphics techniques that extract Z-buffer data for distance information would give an orthogonal view of the scene whereas a laser scanner produces a radial measure. A straight line in the real world will appear curved in the spherical world. Thus the simulation program emulates this radial measure by emitting a ray corresponding to each rangel position. Each ray's intersection with the image is calculated and returned as a proportion of the ambiguity interval.

Reflectance values are calculated in the simulation program based on the object's orientation relative to the scanner. LISA generates artificial noise to simulate sources of noise in real laser systems such as transmission noise, photon noise generated by the photodetector, ambient noise, and noise in subsequent amplifiers.

After generating the range and reflectance data of a scene, the ambiguity interval effect is obtained by dividing each range value by the ambiguity interval and storing the remainder in the depth buffer. The depth and reflectance buffers are then combined and saved to a file for further processing. The output data are arranged in standard 16-bit format where the higher 8-bit stores the range data and the lower 8-bit stores its corresponding reflectance value.

LISA has been used in robotics research by Wu, Norwood and Weiland. Their theses

are listed at the end of this report.

III. Application of Laser Range Finder to Robotic Navigation

This section summarizes research conducted by Peter Weiland (1989) for his MS thesis. The laser ranger finder simulator discussed above was used by Weiland to study the application of laser range data to autonomous robotic navigation.

A scanning laser was chosen for the sensory task because of its ability to readily determine the spatial relationship of objects in the scene. It is not hampered by illumination or the correspondence problem of vision systems. It readily displays distance measurements to each point in the scene. In the navigation system proposed by Weiland a preprocessor is used to enhance the laser data. It first screens the laser data for missing points caused by specular reflection. It then scans the image for the existence of ambiguity intervals and noise. Missing points and noise are corrected by using a 'donut filter'. Ambiguity interval problems are corrected by the addition of one cycle to the range value. After preprocessing, the data are sent to the processor for conversion.

Actual conversion to the Cartesian representation is done in the processor stage. Knowing the geometric configuration of the laser scanner, data points are converted to an (x,y) plane. A corresponding elevation, z value, is entered into the (x,y) plane at the appropriate position. The converted image requires additional procesing. There are numerous missing data points caused by the varying data point density across the image. It also has missing point regions caused by object occlusion. These missing points must be filled in by the post processor before the image is usable by the reasoning system.

Conversion enhancement is accomplished by the post processor. It fills in missing points and frames the image. Shadow points are determined by finding the edges of obstacles from the laser image. Edges are determined by range discontinuities. Missing points caused by data density problems are filled in using the locus algorithm.

Weiland's research produced a perception process that effectively represents simulated laser data in a form that permits cognition. This process is the conversion of laser data into a top down, Cartesian elevation map of the sensed environment. System applicability was demonstrated by navigating through the artificial robot world using simulated laser data and the connectionist network navigation system developed by Norwood and described below.

IV. Applications of Connectionist Networks

Some applications of connectionist networks for robotic path planning, autonomous robot navigation, redundant manipulator control, autonomous insertion task and in aerodynamics are described in this section.

IV -1. Path Planning and Obstacle Avoidance

This subsection summarizes the MS Thesis research of Chris Schuster (1990). Automated path planning and obstacle avoidance have been the subject of intensive research in recent times. Many efforts in the field of semiautonomous mobile-robotic navigation involve using Artificial Intelligence search algorithms on a structured

environment to achieve either good or optimal paths. Other approaches, such as incorporating Artificial Neural Networks, have also been explored. By implementing a hybrid system using the parallel processing features of connectionist networks and simple localized search techniques, good paths can be generated using only low-level environmental sensory data. This system can negotiate structured two- and three-dimensional grid environments from a start position to a goal, while avoiding all obstacles. Major advantages of this method are that solution paths are good in a global sense and path planning can be accomplished in real time if the system is implemented in customized parallel-processing hardware.

The electronic hybrid network system developed by Schuster represents an original method for constrained semi-autonomous robotic navigation control. It takes simple binary environmental input, along with a start and goal location, and processes the data through a connectionist network which provides a nodal 'voltage potential' look-up table. The voltage potentials are analyzed by a second network which determines the move direction based on an examination of the neighborhood around a given current node (beginning with the start position). Finally, the system presents a solution path based on a set of locally optimal steps.

This system exhibits some distinct advantages over the traditional approaches. Due to the parallel architecture of the connectionist network, the system can be expected to be much faster (and possibly more damage resistant) than Artificial Intelligence search algorithms. This navigation system also has the flexibility to account for both moving obstacles and a moving goal. This is accomplished by simply applying new inputs to the connectionist network and reevaluating the path problem. Also, assuming the hybrid network system is implemented in hardware, it does not require a dedicated CPU/microcomputer to make it work and could conceivably be built right into the sensory system and servomotors of a robot.

An example of a near term use for an expanded 'Maze Machine'-type navigation system is the control of a robotic delivery vehicle in a large factory. The environment/floorplan would be fairly stable, however local sensors in the factory could easily update the 'maze' database on board the robot through radio communications. A human or central computer would assign the robot a task, probably also by radio link. An example task could be to "take pallet #196 from point A to point B". The navigation system on board the vehicle would then plan the path from the current location to point A (the pick up point for the pallet) and then plan a second path from A to B (the pallet drop off point). Other routines on board would handle the pallet upload/download procedure, emergency stop procedures, etc.

There are disadvantages as well. First, the traditional AI procedure, regardless of whether they find an optimal path or just feasible paths, are proven methods that can be implemented relatively cheaply on microcomputers. The network system presented here can not be implemented on microcomputers while keeping its parallel architecture, however the sequential simulation AMAZ3D can be a valuable alternative method, especially in feasibility tests to determine if a custom VLSI network setup is warranted for a particular application. Since the AI approaches are software based, they can be modified much more easily to represent different type/size systems and environments. Also, because they are tried and proven procedures, software is readily available for their implementation.

In summary, this electronic hybrid connectionist network system solves path planning/obstacle avoidance problems for a grid-structured, two- or three-

dimensional environment. This navigation system provides good (often optimal) path solutions based on a collection of locally optimal steps/moves found using the output of the sensory analysis connectionist network. It operates using only low-level (binary) descriptions of the environment which can be provided by a variety of current and experimental sensory systems. The navigation machine would best be used in combination with a computer and global sensory input systems. Possible applications of this hybrid network system span from the home to industry to outer space.

IV -2. Autonomous Robot Navigation

This subsection is from the MS thesis of John Norwood (1989) and the Ph.D thesis of Peter Weiland (1991). This research relates to that of Weiland that was discussed earlier.

Robotic path planning and obstacle avoidance have been the subjects of intensive research in recent years. Most solutions to these problems have been reached through the use of traditional Artificial Intelligence search techniques. However, these methods have proven inadequate when applied to highly unstructured or unknown environments. By using an Artificial Neural Network, one can get near optimal paths using only low level information about the scene. In this way, it is possible to navigate from a start position to a goal position while avoiding all obstacles. Major advantages of the method developed by Norwood are that the solution is very fast and does not rely on any a priori knowledge of the robot's environment. The system was shown to be very effective for path generation when used in conjunction with the simulated Laser Imaging System.

Norwood's connectionist network model consists of a collection of individual nodes each having N-1 inputs and two outputs. The first of these outputs is a transient output and connects only to the inputs of the other nodes in the network. The second output is the one of primary interests in the navigation problem. This output is an artificial potential that is induced on a given node by both the goal node and the obstacle nodes in the network, the former having a high positive potential and the latter having a smaller negative potential. Once inputs, representing the robot's environment, are applied to the network, the output potentials emerge as the transient outputs are propagated through the network. Navigation is then a matter of local processing in the neighborhood of the robot until such time as the current node is equivalent to the goal node.

This navigation scheme has been shown to be well suited to simulated unstructured environments where precise information about the terrain and obstacles therein is not generally available. This characteristic of the network is developed from the low-level representation of the environment on which the network operates. The network model can be applied wherever a top-down view can be easily generated from the data provided by the Laser Imaging System Algorithm (LISA).

The navigation network is a general method. It has the flexibility to account for both a moving goal and moving obstacles. This is done simply in hardware by applying new inputs to the network. In software, one need only set a variable which is checked at certain intervals to allow the looping structure to change. In this way, the new environment can be evaluated, and the potentials changed to generate an appropriate new path. Additionally, the network should generalize rather easily to a higher dimension to allow one to generate obstacle free paths in a robot workspace.

Robotic navigation has been an area of intense research since the onset of mobile robot development. The usefulness of mobile robots ultimately reside in their ability to move and interact with the environment. Current approaches to robotic navigation are primarily based on simulating intelligent, human-like behavior through the intelligent system model processing cycle; sense, perceive, reason, act. Unlike these methods, Weiland's PhD thesis (1991) presents a navigation system based on biological and behavioral principles which function in a stimulus-response manner. Using connectionist architectures, a relationship between stimulus and response is acquired through the learning of conceptual information pertaining to navigation. In this research, the mammalian visual system provides a guide for the processing of environmental stimulus. Simulated laser range data are processed in retinal patch size elements by a cellular neural network. This network is designed to detect obstacle existence for each path segment based on an invariant feature of range discontinuity. Obstacle information is then mapped in binary format, indicating the traversable state of the patch, to the system's visual cortex. Response to this mapping is derived from a hierarchical structure of back error propagation neural networks in which each network has learned a particular navigational behavior; obstacle avoidance, wander, and goal seeking. Outputs from these networks indicate appropriate motor response for the environmental stimulus.

A simulation was develolped to evaluate the performance of this system by having a simulated robot traverse an environment. The connectionist approach was verified through system display of human-like navigational behavior for the simulation's environment. Advantages of the neural network approach were also demonstrated by its processing speed and adaptability. Procedures are discussed for actual system implementation in which cycle times of under one second are completely feasible. Proposals for unsupervised learning of navigational responses for environmental stimulus are also made. Weiland's research provides a foundation for our continuing study of the connectionist approach to the problem of autonomous robot navigation.

IV -3. Redundant Manipulator Control

Norwood's PhD thesis (1990) is concerned with the application of Artificial Neural Networks to the problem of controlling a redundant manipulator.

Redudancy in robots is very much an open research area in the field of robotics. As the tasks required of robots become more and more complex, the ability of robots to perform satisfactorily in these applications must increase accordingly. Redundant manipulators have a greater ability to perform difficult tasks, such as obstacle avoidance, than non-redundant ones. In order to make use of this extra ability of redundant robots, more effective control schemes must continue to be developed and to this end, more and more researchers are looking to expand the body of knowledge in this area. Norwood's thesis addresses the problem of moving a redundant robot within a defined workspace in the presence of obstacles. Additionally, criteria are developed that may be applied to the robot to constrain the redundant equations. Finally, a neural network solution to the redundant inverse kinematic problem is presented. It is shown that the inverse kinematics can be developed through a network architecture which provides accurate and fast solutions to a problem that is computationally and structurally complex.

IV -4. Insertion Task

The insertion task, described here, was performed by students in our Advanced

Robotics Laboratory course last semester. It illustrates how results of our research efforts are being transferred to the class room. Two-dimensional geometrical objects consisting of a triangle, a rectangle and a circle are positioned at random locations in the workspace of our RTX robot. The objective is to have the robotic system locate the object specified by a voice command, retrieve the object and insert it into the appropriate receptacle.

This exercise involves use of the TI Speech System, the 3M VDL Vision System, NETS and a JR3 force/torque sensor. The perimeter and area data from the digitized images is sent to the artificial neural network that classifies the objects. A small circular rod located at the center of the area of the object to be retrieved, is grasped by the two parallel jaws of the gripper and force control is used during actual insertion.

Currently this task is being extended to the use of our three fingered hand for grasping the objects by their edges rather than using the rod at the C. G. Then insertion will be attempted using finger control instead of arm motion only. This work is part of the current thesis research by Paul Hanusa and Joe Grisham.

IV -5. Application of NETS in Aerodynamics

Presently wind tunnel experiments and computational fluid dynamics (CFD) numerical modeling are the tools used by researchers and designers for understanding fluid flow phenomena and exploring various aerodynamic designs. For example, in order to predict the pressure distribution across an airfoil, one can use experimental data or obtain numerical solutions of the Navier-Stokes equations that depend on Mach number, Reynolds number, and angle of attack.

We believe that Artificial Neural Networks (ANN) have potential applications in aerodynamics to complement and extend the traditional tools. As an example of such an application, four sets of experimental data for chordwise coefficient of pressure (Cp) distribution on the NACA 0012 airfoil were used to train a backpropagation network using NETS with Mach number as input and Cp at 38 points on the airfoil as output. The network was then tested at 5 other Mach numbers for which data were available with excellent agreement between the network output and the experiments. This work is being done in collaboration with Dr. Andrew Meade, a member of our faculty who works with CFD research at Rice and had a NASA Faculty Fellowship at NASA Langley during the summer of 1991.

V. Preliminary Studies of Fuzzy Logic Control Applications in Robotics

After discussions with Mr. Bob Lea and Dr. Jani of the Software Technology Branch as NASA/JSC we have been convinced that there are opportunities for applying fuzzy logic control in robotics. William Atkinson and Kevin Magee are two graduate students in our program who are currently investigating ways to apply both artificial neural nets and fuzzy control to redundant manipulators. Sarmad Adnan, a PhD candidate, with the help of Alex Stewart, an undergraduate, is working toward developing a camera tracking system that can have applications for use with fuzzy control of a mobile robot.

Technical Papers, Reports, and Theses

Cheatham, J. B., and Norwood, J. D., "Robotic Path Planning and Obstacle Avoidance: A Neural Network Approach", 5th IASTED Int. Conf. Expert Systems and Neural Networks, Honolulu, Aug. 16-18, 1989.

Cheatham, J. B., Weiland, P. L. and Norwood, J. D., "Framework for a Laser Navigation System", RICIS '88 Symposium, Nov. 7-9, 1989.

Regalbuto, M. A., Cheatham, J. B. and Krouskop, T. A., "A Model-Based Graphics Interface for Controlling a Semi-Autonomous Mobile Robot", 11th Annual Conference IEEE-EMBS '89, Seattle, Nov. 9-12, 1989.

Cheatham, J. B. and Adnan, S., "Kinematic Analysis and Trajectory Control of a Mobile Omni-Directional Robot", First National Conference on Applied Mechanisms and Robotics, Cincinnati, Nov. 5-8, 1989.

Wu, C. K., Weiland, P. L., and Cheatham, J. B., "A Computer Graphics Testbed for Developing and Testing Laser Imaging Algorithms," SPIE/SPSE Symposium on Electronic Imaging, Santa Clara, CA., Feb. 11-16, 1990.

Krouskop, T. A., Cheatham, J. B., Kachroo, P., and Barry, P. A., "Non-Invasive Measurement of the Stiffness of Soft Tissue Using an Ultrasonic Perturbator," 8th Annual Conference on Biomedical Engineering Research in Houston, Feb. 15-16, 1990.

Norwood, J., Weiland, P. L., and Cheatham, J. B., "Robot Navigation From Local Sensory Data Using Neural Networks," 5th Int. Service Robot Congress, Detroit, June 6, 1990.

Adnan, S., and Cheatham, J. B., "An Omnidirectional Platform to Simulate a Free-Flying Robot," ISMCR '90 - 1st Int. Sym. Measurement and Control in Robotics, NASA-JSC, June 21, 1990.

Chen, Y. C., Walker, I.D., and Cheatham, J. B., "Grasping Analysis for a Multifingered Hand," *Proceedings International Symposium on Measurement and Control in Robotics*, NASA/JSC, June 1990, Vol 3, pp J2-3-1-6.

Kachroo, P., Krouskop, T. A., Kachroo, A., Cheatham, J. B., and Barry, P., "Ultrasonic Technique and Artificial Intelligence: Differentiation of Tissue Types," Annual Fall Meeting of the Biomedical Engineering Society, Blacksburg, VA, Oct. 23, 1990.

Regalbuto, M. A., Cheatham, J. B., and Krouskop, T. A., "A Framework for a Practical Mobile Robotic Aid for the Severely Physically Disabled," RESNA 13th Annual Conference, Washington, D.C., 1990.

Walker, I. D., Cheatham, J. B. and Chen, Y. C., "An Efficient Method for Computing the Force Distribution of a Three-Fingered Grasp," *Proceedings SPIE Conference on Cooperative Intelligent Robotics in Space*, Boston, MA, November 1990.

Wu, C. K., Lin, Y. H., and Cheatham, J. B., "Computer Graphics Modelling for Simulating and Testing Robot Vision Systems," *Journal of Modelling and Simulation*, **10** (2), 67-70, 1990.

Chen, Y. C., Walker, I. D., and Cheatham, J. B., "A New Approach to Force Distribution and Planning for Multifingered Grasp of Planar Objects," submitted to *Journal of Robotic Systems*, 1990

Walker, I. D., Cheatham, J. B., and Chen, Y. C., "An Intelligent Grasp Planning Strategy for Robotic Hands," *Proceedings SPIE International Symposium on Optical Engineering and Photonics in Aerospace Engineering*, Orlando, FL, April 1991.

Chen, Y. C., Walker, I. D., and Cheatham, J. B., "A New Approach to Force Distribution and Planning for Multifingered Grasps of Solid Objects," *Proceedings of the IEEE International Conference on Robotics and Automation,* Sacramento, CA, April 1991.

Chen, Y. C., Walker, I. D., and Cheatham, J. B., "A New Vectorized Approach to Visualization and Full Use of Kinematic Redundancy," submitted to *1991 IEEE International Conference on Robotics and Automation.*

Chen, Y. C., Walker, I. D., and Cheatham, J. B., "Grasping Technology for Dextrous Remote Manipulation," submitted to *IDEEA One - The First International Design for Extreme Environments Assembly,* University of Houston, November 12-15, 1991.

Adnan, S., Cheatham, J., Galicki, P., and Brock, J. "Telepresence for Remote Robotics Operations," submitted to *IDEEA One - The First International Design for Extreme Environments Assembly,* University of Houston, November 12-15, 1991.

Oldham, T., Stokes, S., Venverlok, D., and Zearfoss, J. "Wheel for Omnidirectional Platform," Final Report, Senior Design Project, Mech 408, Rice University, April 26, 1990

Fessler, J., Jenkins, E., Miller, R., Peiffer, T., and Ragan, E., "Seven Degree of Freedom Robotic Arm," Final Report, Senior Design Project, Mech 408, Rice University, April 20, 1990.

Ducceschi, L., Matthews, M., O'Connell, S., and Yuan, L. "The Rice Four-Fingered Dextrous Robotic Hand," Final Report, Senior Design Project, Mech 408, Rice University, April 19, 1991.

Bartosh, B., Boncimino, G., Caldinell, T., Kirkpatrick, D., and Shampine, R., "Four Degree of Freedom Robotic Wrist," Final Report, Senior Design Project, Mech 408, Rice University, April 19, 1991.

Recent Theses Supervised

Paul B. Fisher. "Development of Sensors and Algorithms for Automating Robotic Grasping," M.S. Thesis in Mechanical Engineering, June 1988.

Sarmad Adnan. "Kinematic Analysis and Trajectory Control of the Rice Omni-directional Mobile Robot," M.S. Thesis in Mechanical Engineering, April 1989.

Peter Weiland. "Use of Laser Scanning Rangefinders for Autonomous Robotic Navigation," M.S. Thesis in Mechanical Engineering, May 1989.

John Norwood. "Robotic Path Planning and Obstacle Avoidance: A Neural Network Approach," M.S. Thesis in Mechanical Engineering, May 1989.

Chris K. Wu, "The Use of Laser Imaging System for Automated Vehicle Guidance and Space Servicing Tasks," PhD Thesis in Mechanical Engineering, April 1990.

Michael A. Regalbuto, "A Semi-Autonomous Mobile Robot/Teleoperator with Applications as an Aid for Severely Handicapped People," PhD Thesis in Mechanical Engineering, January 1990.

Pushkin Kachroo, "Ultrasonic Technique and Artificial Intelligence: Differentiation of Tissue Types," M.S. Thesis in Mechanical Engineering, May 1990.

Christopher E. Schuster, "A Path Planning and Obstacle Avoidance Hybrid System Using a Connectionist Network," M.S. Thesis in Mechanical Engineering, June 1990.

John David Norwood, "A Neural Network Approach to the Robot Inverse Kinematic Problem in the Presence of Obstacles," Ph.D Thesis in Mechanical Engineering, December 1990.

Peter L. Weiland, "A Connectionist Approach to Autonomous Robotic Navigation," Ph.D Thesis in Mechanical Engineering, January 1991.

William Thomas Atkinson, "Remote Control of a Robotic Arm Through Speaker-Dependent Isolated-Word Speech Recognition," MS Thesis in Mechanical Engineering, February 1991.

Yu-Che Chen, "A New Method for Solving the Kinematics of Multifingered Grasping and General Redundant Manipulators -- A Task Oriented Approach," Ph.D. Thesis in Mechanical Engineering, April 1991.

REPORT

TO

RESEARCH INSTITUTE FOR
COMPUTING AND INFORMATION SYSTMES

(RICIS)

ON

RESEARCH ACTIVITY AI.2

COMPUTER GRAPHICS TESTBED TO SIMULATE AND TEST VISION SYSTEMS FOR SPACE APPLICATIONS

By

J. B. CHEATHAM

CONTRACT NCC9-16

MECHANICAL ENGINEERING AND MATERIALS SCIENCE DEPARTMENT

RICE UNIVERSITY

HOUSTON, TX 77251-1892

MARCH 1990

RICIS RESEARCH ACTIVITY AI.2

Computer Graphics Testbed to Simulate and Test Vision Systems for Space Applications

by

J. B. Cheatham

The major objective of this research activity has shifted from computer graphics and vision systems to the broader scope of applying concepts of artificial intelligence to robotics. Specificially, the research is directed toward developing Artificial Neural Networks, Expert Systems and Laser Imaging Techniques for Autonomous Space Robots.

This activity is being conducted by Wu, Chen, Norwood, Weiland, Schuster and Atkinson, who are Mechanical Engineering graduate students at Rice, and it is directed by Professor John Cheatham. Major accomplishments have been reported in technical papers and two M.S. and one Ph.D. thesis. A computer graphics simulator for laser imaging systems has been developed by Chris Wu as part of his Ph.D. research. This simulator has been utilized in research by Peter Weiland to provide a top-down map of the environment from laser range data. John Norwood has applied potential theory for obstacle avoidance and developed a network for path planning using the representation of the environment provided by Weiland's research.

Weiland and Norwood are extending their research while working toward Ph.D. degrees. This effort is now being directed toward direct interpretation of laser imaging data using an Artificial Neural Network and extension of the robot navigation to three dimensions. Work is also aimed at using an Artificial Neural Network for control of a 7dof redundant manipulator with three 3dof fingers. It is hoped that this research will be of some assistance to the NASA robotic activities related to the Space Station and to exploration of the moon and Mars.

Thesis Resulting from Research Activity AI.2

Peter Weiland. "Use of Laser Scanning Rangefinders for Autonomous Robotic Navigation", M.S. Thesis in Mechanical Engineering, Rice University, May 1989.

John Norwood. "Robotic Path Planning and Obstacle Avoidance: A Neural Network Approach", M.S. Thesis in Mechanical Engineering, Rice University, May 1989.

Chris K. Wu. "The Use of Laser Imaging System for Automated Vehicle Guidance and Space Servicing Tasks", Ph.D. Thesis in Mechanical Engineering, Rice University, January 1990.

Recent Technical Papers

Chris K. Wu, Peter L. Weiland and John B. Cheatham. "A Computer Graphics Testbed for Developing and Testing Laser Imaging Algorithms", 1990 SPIE/SPSE Symposium on Electronic Imaging Science & Technology, Santa Clara, California, February 1990.

Peter L. Weiland, John D. Norwood and J.B. Cheatham, Jr. "Robotic Navigation from Local Sensory Data Using Neural Networks", 1990 International Automation Conference, Detroit, Michigan, June 1990.

C. K. Wu, J. B. Cheatham, Y. H. Lin and T. F. Cleghorn. "Computer Graphics Modelling for Simulating and Testing Robot Vision Systems", to be published in Journal of Simulation and Modelling.