1985

NASA/ASEE SUMMER FACULTY RESEARCH FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA AT HUNTSVILLE

MACHINE VISION AND THE OMV

Prepared By:	Michael A. McAnulty
Academic Rank.	Assistant Professor
University and Department:	University of Alabama in Birmingham Department of Computer and Information Science
NASA/MSFC:	
Laboratory:	Information and Electronic Systems
Division:	Software and Data Management
Branch:	Data Management
NASA Counterpart:	Frank Vinz
Date:	30 August 1985
	NASA-NGT-01-008-021 (The University of Alabama in Huntsville)

Machine Vision And The OMV

M. A. McAnulty, NASA/ASEE Fellow
NASA Contact : Frank Vinz
Marshall Space Flight Center

The orbital Maneuvering Vehicle (OMV) is intended to close with orbiting targets for relocation or servicing. It will be controlled via video signals and thruster activation based upon earth or space station directives. A human operator is squarely in the middle of the control loop for close work. Without directly addressing future, more autonomous versions of a remote servicer, several techniques that will doubtless be important in a future increase of autonomy also have some direct application to the current situation, particularly in the area of image enhancement and predictive analysis.

Several techniques are presented, and some few have been implemented, which support a machine vision capability proposed to be adequate for detection, recognition, and tracking. Once feasibly implemented, they must then be further modified to operate together in real time. This may be achieved by two courses, the use of an array processor and some initial steps toward data reduction. The methodology for adapting to a vector architecture is discussed in preliminary form, and a highly tentative rationale for data reduction at the front end is also discussed. As a by-product, a working implementation of the most advanced graphic display technique, ray-casting, is described.

Acknowledgements

Appreciation comes in many flavors. For overall mission orientation and administrative support we owe James Dozier, Leroy Osborn, and Gerry Karr a great deal. For further elucidation of the NASA situation and requirements, as well as technical support, thanks are due to Caroline Wang, Steve Purinton, Tom Bryan, Debbie Graham, and Audie Anderson. My counterparts, Frank Vinz and Ken Fernandez, were liberal with both information and resources, including an excellent technical report which I am still trying to catch up to. [14]

I. Introduction

The orbiting Maneuvering Vehicle (OMV) has ben described as an orbiting tugboat, and is intended to reposition other orbiting vehicles and, eventually, perform simple maintenance procedures. It is expected to be the first in a long line of unmanned remote vehicles, much in the way Mariner was but more than a passive presence. Although the planned version of the OMV is to be entirely ground- or space station-controlled, that is, a telepresence or tele-operated vehicle, the case is made here that some considerable testing and verification of more autonomous versions of the OMV can be performed within the teleoperation framework. The proposed techniques do not in any way propose to modify how the OMV would be operated, but fit well within the scope of enhancements to a ground based operation, primarily in the area of predictive graphics, to be described below. This report describes some preliminary procedures that pertain primarily to on-board vision and scene interpretation, but which would necessarily be well-integrated with other sensory and effector modalities in a working prototype.

The following sections describe, in turn, the planned mode of operation of the OMV, aspects of machine vision that relate to it, a global description of necessary tasks, and a report upon work accomplished so far and planned for the near future.

II. Current Planned OMV Operations

A considerable amount of ground simulation of the teleoperation of a remote vehicle has been performed, utilizing a compressed air controlled vehicle that rides on air bearings on the 'flat floor' at MSFC. Included in the simulations is a very low bandwidth communication channel for video signals from the vehicle, which allows for no more than approximately one 256 by 256 pixel frame per second This mimics the expected performance of the TDRSS satellite channel that would be used for orbital operation. A further feature of this channel is that significant time delays occur, on the order of a second each way in low orbit.

The vehicle operator controls a six-degree-of-freedon joystick to 'fly' the vehicle, and observes the vehicle's surroundings via one or more video cameras mounted, with lights, on the vehicle. Four cameras are currently planned for far and near work, and for different views of a docking operation, although only two are mounted at present. Thus, practically all control of the vehicle is exercised by an attentive human operator.

The communication channel delay means that an operator does not see the result of an action for approximately two seconds This is approximate because there are also abrupt changes in the signal delay, on the order of a half-second or so, as the routing changes due to earth position. Such considerations are critical, since docking and close work cannot be done at great speed. Collisions at too great a speed will either be near-elastic or, if kinetic energy is absorbed, damaging. Even if contact can be achieved and maintained, several internal systems on some targets may be impaired by even moderate shock. In fact, the necessity for slow maneuvering tends to lessen the time delay problem as the operator will necessarily work in finer increments.

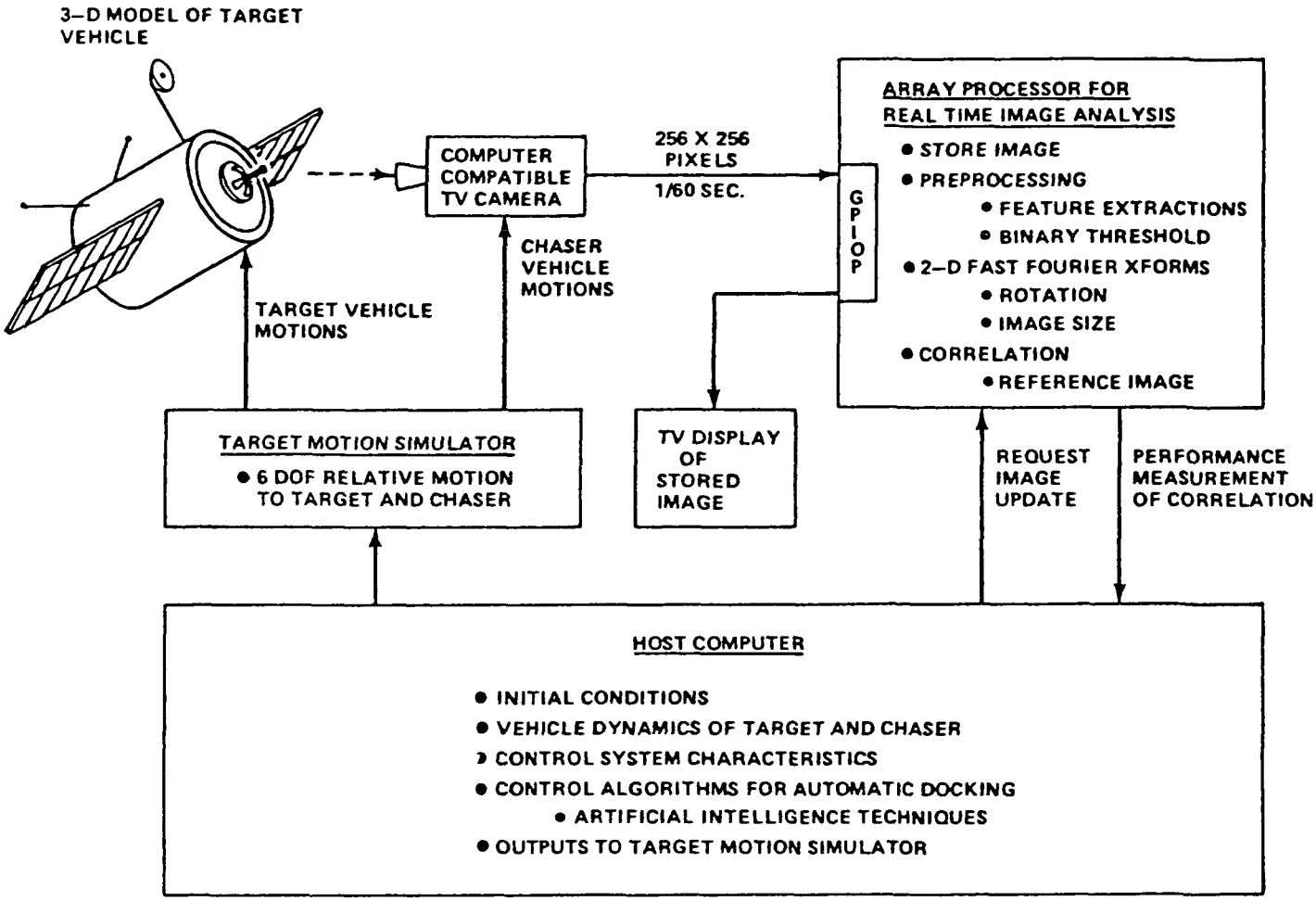


Exhibit 1 - The simulation environment at MSFC. Note that the vector processor, tightly coupled to a host computer, is squarely in the loop as well, and may be used electively to either perform operations on the images, or simply pass them through unaltered.

Although a straight video signal is sufficient for earth simulations, it is anticipated that some cleaning and enhancement of the video picture might be desirable in actual operations. To this end, a signal matching interface between the video signal and the General Purpose Input-Output Processor (GPIOP) of an FPS-5210 floating point array processor is under construction. The FPS can perform simple filtering operations in under half a second (to be described in a later section) upon the entire image, which matches the input rate anticipated via TDRSS, but which also will introduce more delay into the loop.

It has been suggested [1] that the time delay problem might be addressed by a technique termed 'predictive graphics'. Rather than showing the operator the original video signal, a graphic rendering from a formal model, such as an engineering description or a constructive solid geometric model, is shown instead. The model would be so presented as to appear like the object would after a suitable delay. This requires two things which are at present entirely absent from the simulation environment.

- o Matching procedures for fitting the formal model to the video picture, in order to determine the target's current orientation and position, and
- o Means for predicting where the target will be after a suitable delay, which would utilize both measured trajectories of the pieces of the target, including transforming these to geometric operations on the formal model, and knowledge of the vehicle's motion relative to the target, to be derived from joystick or thruster signals.

This topic will be returned to after a generic discussion of what is generally agreed to constitute machine vision

III Aspects of Machine Vision

Two properties of machine vision relevant to this project. The first of these is that machine vision is not an end in itself, rather a means for achieving some end. The second is that current thinking about machine vision separates it into a number of steps that deal with quite different kinds and amounts of data. The flavor of artificial intelligence is, in both aspects, difficult to ignore.

Purely passive vision, with no goal orientation, is rather difficult to contemplate meaningfully. At the very least, a seeing entity is constructing a story. Some situations are rather clear cut. Standard bin-picking problems use a vision capability to aid a robot in the task of picking an object from among other objects, lying at an arbitrary orientation, and move it to another location to be placed at a specified orientation Surveillance systems, in their simplest form, are required to detect unexpected motion, or more generally unexpected phenomena This raises the problem of specifying what expected phenomena look like, and requires a rather more complicated world model Further, human vision works explicitly or implicitly with not only a well-developed world model, but in concert with several other senses as well, of which locomotion and touching are possibly most important.

The basic functions planned for the OMV would appear to lie within a restrictive world model consisting of a specification of the target vehicle and of planned procedures, although even this may become more complicated in the event that damage assessment or contingency actions are required. For the near term, having a person in the control loop bypasses the more sophisticated requirements entirely.

The second aspect of machine vision, its multistep nature, is of some considerable relevance. Consider that a system begins with a very simple data structure, the sampled intensity function in a rectangular array (the raster), and from it should derive a symbolic description of the essential features of the scene, sufficiently informative to enable actions to be taken. A representative, although by no means authoritative, list of steps follows. It should be noted that each of them involves both data reduction and data transformation.

- *Segmentation* is the process of classifying regions of the picture solely in terms of their distinctness from each other. A usual picture of 65,000 or 250,000 pixels will generally produce a number of regions two or more orders of magnitude fewer than the number of pixels. The description of each segment will include a boundary encoding and perhaps a list of central moments, or some other topological description. Naive segmentation is probably the most over-investigated and best understood component of computer vision.
- *Articulation* , or the analysis of the relationships between segments, analyzes adjacencies and connections. Since a segment represents a surface, at this point possible and probable orientations of the surface and its curvature are essayed Relevant data structures include adjacency graphs and preliminary parametric descriptions
- *Naive Construction* attempts to interpret the surfaces as belonging to one or more physically possible objects, and attempts to build a three-dimensional model consistent with the scene At this point components of the scene may be labelled as generic, domain-independent geometric primitives
- *Labelling* ,or attaching domain-dependent names to the objects, relates the world of possible objects to that of known, named objects By this stage a preliminary recognition may be said to have occurred.
- *Updating* the current world model, in the case of a time sequence of scenes, is important. If a single scene were being analyzed, this would not be a critical step, but most useful applications not only involve changes in time, but make considerable use of these changes in interpreting the field of view. In the case of predictive graphics, both quantified trajectories and logical data (visible/occluded) would be used.
- *Domain Consistency* is analogous to naive construction, in that 'the interpretation is tested for possibility as a world situation. The plausibility and consistency of the domain labelling is evaluated in domain terms. It is at this level that symbolic programming is most likely to be employed, as well as sophisticated concept structures. Further, considerable control of all previous steps is likely to be exercised here, perhaps redoing some of them in different fashion to provide a more consistent interpretation. Thus, a system 'thinks' not only about the application domain, but about its own operation as well. [8]

Although these steps are presented as discrete phases, there is likely to be considerable crosstalk between them. It is desirable, however, to construct and maintain a system in separate components, since the objects manipulated differ widely between phases, and in fact different phases may be best implemented by different programming paradigms.

Situations which make explicit use of time variance can work in a much more efficient mode, making use of the property that in the real world things should not change radically from one view to the next. In a sequence of views, components may be 'tracked' rather than recognized from scratch every time.

This property is termed time coherence. Its relative, spatial coherence, may be stated that if some pixel is determined to be on a particular surface, then neighboring pixels have a high probability of being on the same surface.

On the proscriptive side, it is useful to discuss briefly what machine vision is not. That is, while the following techniques may play a modified part in certain portions of a system, none is in itself sufficient to achieve the desired objectives on its own.

Image processing projects a picture function onto another functional basis, such as the Fourier, Hadamard, or Walsh functions, and examines the projection for various clusters of coefficients which may relate to image features. The weakness of this approach is that one is confronted with as large a data space after the transform as before, no reduction in the problem size has been accomplished. Further, features in a function space relate only occasionally to useful scene properties. Segments of a scene may be usefully transformed for surface properties such as texture.

Classical *pattern recognition* assigns a data cluster to one of a finite number of classes based upon a training sample. There are both statistical and syntactic forms of pattern recognition, but as traditionally employed they result in extremely difficult program control structures. This is partly the result of trying to push domain reasoning into procedural code, and why the multicomponent model prescribed above has evolved. Another weakness is that part of the recognition process, related to model fitting, involves a continuum of interpretations rather than a finite number of names.

Model fitting appears especially appropriate to the predictive graphics situation, in that certain parameters (the position and velocity coordinates of the target) must be optimally determined There is, however, no global closed form solution that will fit a structural model to the two-dimensional projection in the scene. A trial and error method, using an expensive correlation as criterion function, is necessary if one wishes to take a global approach. It appears that an analysis of the scene components, essentially all the prescriptive steps described above, is much more likely to provide an initial parameter setting that may then be tested at specific points or regions rather than over the entire model. For one thing, a model must necessarily be modified to reflect that certain of its components will not even be visible in the projection, hence cannot be explicitly fit.

IV. Proposed Tasks

There are two broad classes of necessary activity. The first is to develop procedures that will enable recognizing and tracking a typical target. The second is to then modify the procedures so that they will work in near real time, an objective that is initially to be met by using the vector processor, and which can be further approached by reducing data handling requirements.

Recognizing and tracking appear to require at least three different areas of work, segmentation, labelling, and motion interpretation. Before discussing any of these in depth, an overall strategy should be explained, which is an essentially top-down, or global, approach. Under this, no single component of the objective is to receive special attention or optimization in the first pass. Rather, a minimally competent module for each required function should be implemented as quickly as possible, and all the prototypical modules be made to work together. This is primarily because the least well understood aspect of a modular vision system is the communication between modules, the types of data

structures and control messages that inform and direct the performance of each of the modules. To put it another way, it is not possible to evaluate a single component of a system, an edge detector as a trivial example, in isolation. There are at present no criteria for determining that a particular module is optimal for overall performance of the system, we simply do not understand the coupling between modules well enough.

Related to this lack of organizational knowledge, it is expected that a complete effort will, in succeeding passes, experiment with a number of different techniques fit within the overall structure. There exist several documented techniques for segmentation, for example, in both the edge-detection form and its dual, region growing. There are a great number of region description techniques suitable for fitting to a two-space model, although fitting to three-dimensional models is less well understood. Specific algorithms are proposed in the following for two reasons, they represent the state of the art at the component level, and they appear suitable for adaptation to parallel processing. As a result, the proposed inventory is weighted heavily toward the class of relaxation algorithms, those which appear to rely upon local, semi-cooperative processes. That these happen to coincide with our concept of biological vision is merely fortuitous, their main property is that they may be adapted, in some fashion, to near simultaneous processors. Although each will be referenced individually, they are all drawn from the same source [2] and will be given page numbers in the references.

Segmentation Using Edges

The Prager edge relaxation algorithm [10] essentially amplifies consistent edge phenomena, initially detected as a nontrivial intensity gradient across any pixel. Even weak gradients will be amplified providing that a spatial series of them across many pixels exists. Thus, pure gradient magnitude is a secondary determinant, rather the repetition of a gradient on neighboring pixels determines which edges are amplified and which suppressed.

Although the detection and amplification of edges by this process is indeed highly parallel, the result of it still requires considerable work in order to define regions Each edge must be traversed, an essentially serial process, to determine closed curves (which thus define a segment), and it will often happen that curves do not close, so that some higher order processing is eventually necessary.

Segmentation By Growing Regions

The line-handling problems of the edge detector may be quite elegantly side-stepped by the device of merging picture regions, which must always maintain well defined boundaries, though this is expensive in terms of storage. An older method [4], suitable for monochrome images, initally treats each pixel as a separate region (segment), and then dissolves boundaries between segments if there is sufficient similarity between regions at the boundary. Regions are maintained in tabular form, recording various statistics such as central moments and intensity distribution statistics, as well as suitable boundary encodings This is a formidably large data-base, and it tends to remain somewhat constant in size, since as the number of regions declines the length of the boundaries increases.

The adaptation of this procedure to vectorized form has not yet been attempted, even on paper, since adjusting to the unsystematically changing granularity of the dynamic pictorial data base appears to be unsuited to the systematic gridding of a vector processor. However, as described in Appendix B, sufficient other techniques have proven feasible that this may eventually yield as

well, and a proof that it will not, if demonstrated, is in itself a highly informative and useful result.

Segmentation By Thresholding

This is perhaps the most ancient of research techniques, highly useful because the resulting objects, binary images, are by far the most tractable. Further, the camera controller currently being used will threshold automatically, "on the fly", at any arbitrary intensity value, which eliminates a processing step. This has not been considered for two reasons. First, although initial approaches to an orbiting target in fact present an object-versus-background situation, the background in question may be light (earth), dark(space), or composite (horizon in view). This complicates the background definition. Further, it has been argued previously that simple interpretation of an entire target is not likely to be as useful as an analytic, piecewise interpretation of its components. Second, thresholding does not deal at all well with curved surfaces which will, under anticipated lighting conditions, exhibit a range of intensities. Both edge detection and region growing handle such surfaces quite gracefully.

It should be noted that the result of a segmentation, however achieved, is a binary image, so that all the algorithmic advantages remain, and the various regions are more descriptive than would be the case with pure thresholding.

Optical Flow

We anticipate that the process of connecting image pieces (segments) and analysis of scene changes, often termed optical flow, will be intimately connected, in that segments which move together are connected Two algorithms for optical flow are proposed, and one is the object of careful scrutiny with respect to vectorization Each method constrains neighboring flow (motion) vectors to be similar in orientation and magnitude, thus enforcing a motion coherence heuristic.

The first, due to Horn and Schunk [6], relates object motion to changes in intensity, using the deceptively simple relationship between time, intensity and position

$$f(x + dx, y + dy, t + dt) = f(x, y, t).$$

A system of constraints results in a problem soluble by Gauss-Seidel relaxation, a well-understood iterative technique quite suited to vector processors. This is one instance of the application of regularization, esssentially the introduction of a stabilizing functional (the coherence assumption) into a minimization calculation, and is widely applied in computer models of early vision

The second method, due to Barnard and Thompson [3], is more constructive and involves a large series of local cross-correlations at 'interesting' image points. It attempts to relate, subject to a smooth motion constraint, interesting points in two images An 'interesting' region may simply be a large intensity gradient (an edge or corner, by assumption) or may instead correspond to specified phenomena in a segmentation description. This method is considerably more challenging than the first as a candidate for vectorization but has the advantage that processing is not global over the picture, but rather only in selected portions of the images

A velocity map is useful in two ways. First, it provides a lot of information prior to recognition and model mapping, independent of the problem domain, concerning range and timing. Avoidanc maneuvers, if necessary, can work from restricted 'objects world' consideration of velocities Thus, the rate and direction of 'looming' provides quite accurate information about the relative motion of chaser and target. Second, since non-trivial velocities generally correspond

to edges and other detectable features, rather a lot of segmentation work can be achieved in the process of deriving velocities It appears that relatively competent velocity (flow) derivations can be achieved at considerably reduced resolution, which may in fortunate cases speed up this process by a significant amount.

Data Reduction

The two alternatives sketched in optical flow illustrate a powerful choice, that between global techniques, analogous to image processing transforms, and local, semantically informed techniques. The major challenge of the second sort is to efficiently represent and describe interesting regions, so that the overhead required in restricting computation to those areas is not unduly great. As an initial guess, this is expected to involve heavy use of list structures quite unsuited to the systematic coordinate scheme within vector processors.

Another, highly systematic, form of data reduction involves a variable resolution scheme. This term is generally applied to pyramid schemes, which involve calculating lower resolution pseudo-pixels at a level of the pyramid as an average of several (usually four) pixels at the next lower level. The overhead involved in calculating a full resolution pyramid is substantial and unacceptable for current purposes, but the advantage of it is that considerable work can be done at lower-resolution levels, which cuts processing time considerably. We suggest that, rather than calculate a full pyramid, some small percentage of the image points that samples the scene may perform as well as a true averaged pyramid level. This is the object of an independent study.

Model Matching

The process of matching image segments to model components has not been directly addressed yet, although a candidate model has been developed somewhat fortuitously. The constructive solid geometry (CSG) used to generate test objects is quite adequate for man-made objects, constructed of relatively simple primitive shapes. Further, the absence of streamlining, necessary for aeronautical objects but not astronautical ones, means that very few complex surfaces are likely to be present. In addition to this correspondence, a CSG description is more concise and complete than most surface-oriented models, and it is well structured (a binary tree) for algorithmic manipulation. The model is discussed briefly in Appendix A, and the reader is referred to Roth [12] for a highly readable and competent discussion

V. Summer 1985 Achievements

Activity during this period concentrated, not by design, upon building an infrastructure This consisted of the generation of test images, the establishment of data paths between the three involved processors, and acclimatization to the computing environment One segmentation procedure was successfully implemented in serial form, and analyzed on paper in vector form. These topics are discussed in turn.

Test Images

It is desirable to begin with images that incorporate properties that will exist in the real environment, such as perspective distortion, shadowing, shading, hidden surfaces/objects, and noise. Of all the graphical rendering

techniques, only ray-casting can elegantly incorporate all these phenomena, and transparency, mass properties, and translucency as well. (Noise generation is a different, though routine, matter, and hasn't been necessary yet) The geometric model that underlies this implementation is constructive solid geometry, and is described, along with ray-casting proper, in Roth [12]. A brief orientation to the implemented software is presented in Appendix A. Ray-casting has been adapted to a vector processor [9], but to do so in this project was felt to be of minor significance, since implementing the serial version (10-30 minutes per image) used up an unexpected amount of calendar time.

Data Paths, Environment

The vector processor, an FPS-5210, is hosted by a PDP-11/570 running RSX-11/M. This is connected via Decnet to a VAX-11/780 running VMS. The connection turns out to be fortunate. Evaluation of performance, particularly of the segmenting procedures, can only be done visually, using pictures, so that some graphic capability is essential. A small half-toning routine enables shaded raster images to be sufficiently well-displayed on the available vector displays, which are various Tektronix terminals, to enable interpretation. This is described in Exhibit 4. Raw pixel intensities are logarithmically adjusted by the half-toning routine so that a satisfying spread of shading levels is achieved. Because of a noisy connection to the 11/570 terminal, all graphics work must be done on the VAX. Raster images generated in the VAX cannot be directly transmitted and used by the 570, so must be put into character form and sent to the 570, where they can then be directly loaded into the FPS. Returning images to the VAX for display requires the inverse process.

Although anticipated image sizes will be 256 x 256, many procedures take long enough during testing that smaller images are desirable. To this end, all raster files (*.ras) are preceded with two numbers describing the dimensionality of the image, and most programs adjust themselves to this information.

While this seems to be an inordinate amount of procedural overhead, it does not directly relate to the target situation at all, where images will go directly from the video controller into FPS memory, and it is certainly acceptable for development and testing purposes.

A further accomodation is necessary in that many algorithms are expressed most concisely in a language that supports dynamic allocation (for lists, queues, etc), data structuring (so that different types may be included in a single object), and recursion (for concise management of list and tree structures). These capabilities exist in Pascal, Ada, PL-1, and C, but of these only Pascal is available, and its treatment of libraries and modularity is extremely awkward. As a result, all implementations are in Fortran.

The Prager Edge Relaxor

Of all the proposed procedures, only this was fully implemented in serial form Although the serial implementation was quite straightforward, the time performance is disappointing, requiring about five minutes for each cycle, or thirty or so minutes for reasonable convergence on a 256 x 256 image An illustration of its operation is included as Exhibit 6.

As suggested in a previous section, an edge-based segmentor is inconvenient, in that the results do not directly define segments. This is because a segment must include a closed and unambiguous boundary, and the edges that are produced by this algorithm need further attention before closed boundaries can be assured. The choice of this procedure was due to its always local nature, so that it looked like a good practice candidate compared with region growing.

As an exercise for learning the existing capabilities of the FPS it worked well.

Using The Vector Processor

There are two considerations, how to vectorize algorithms, and how to use the specific processor. The first issue is addressed in Appendix B, the second is treated here.

A first rule, which will not vary, is that once one is into the vector processor one must stay there until finished. A single vector operation on a full picture (65,536 pixels) takes about a tenth of a second. Loading or unloading a picture to host disk (the 570 can't hold an entire picture) requires thirty seconds. This is a general rule for any peripheral accelerator.

A second principle, for development purposes only, is to use only the FPS Math Library routines, and forgo various optimizing possibilities such as the vector function chainer, the arithmetic coprocessor, and direct assembly coding until the need for these becomes apparent. The repertoire of routines is quite rich, if one stays within a single page of the system memory. Routines which work across page boundaries include only a few arithmetic operations. There is the option of coding some of the logical vector routines to work across page boundaries as well, thus avoiding the necessity of working with images in smaller pieces (the 'mosaic' problem), but it appears that in many cases temporary data requirements will be so extensive that subdivision cannot be avoided in any case, so that a direct confrontation with the subdivision-reconstruction problem must be made.

The FPS will perform a simple vector operation such as addition or comparison in about .1 seconds for a 65,536 element vector, which is the size of a picture (256 x 256). A very simple four-point averaging filter can thus execute in under a second (see Appendix B), although more complex operations will, if the full resolution of the picture is used, take on the order of several seconds or more. The speed of the FPS is nearly the same as that of the VAX for floating point operations, so that if the co-processor is used along with the main processor a significant amount of routine work can be offloaded to it.

Further Proposed Work

The summer's effort has barely scratched the surface of the tasks necessary to enable, even in non-real-time, a primitive matching and prediction facility. This was occasioned partly by the amount of effort to generate raster test images (what a video camera 'sees'), and partly by extensive upgrading of equipment, rendering it unavailable.

The tasks necessary to establish a working, though creaky, prototype are essentially of two kinds. The first is to implement a better segmentation algorithm using region growing, and to implement a motion analysis algorithm, essentially a tabulation of differences between two pictures. This set of tasks, while nontrivial, can be expected to require two months of effort. The second kind is considerably more challenging, and deals with matching picture segments to a geometric model. This is considerably eased if we assume that the object in view is already known, so that we know which model specification to work from, but it is still a very challenging opportunity.

Tasks of the first kind appear to adapt well to a parallel processor, those of the second appear not to These latter will probably, in a first cut, consist of some variation upon (partial) graph matching. Rather little work of any kind has been reported concerning graph matching on a parallel processor, it is more intuitively suited to a symbolic or list-processing environment and perhaps best

implemented there for development purposes. This is not to say that graph matching can't be done 'in parallel', merely that to do so in a general way requires breaking new ground.

Finally, the integration of the pieces discussed is in itself not very well defined, and a control structure that applies the pieces correctly is also likely to be better specified in a linear, symbolic environment. Further, we suggest that analysis would proceed in two rather different modes. One of these would bypass labelling entirely, analyzing a motion sequence entirely in terms of object continuity and apparent motion, a task that can be done at considerably reduced resolution and can cut the cost of a vector operation by a factor of 100 or more, and provide sufficient information for obstacle avoidance and tracking.

Motion analysis itself can provide considerable segmentation information, thus aiding recognition of target components and their labeling. Matching and tracking also can make do with considerably less data than an entire picture, but the data is localized in 'interesting' areas, generally object boundaries the specification of which is a primary objective of segmentation. While motion analysis should provide rapid and continuing results for avoidance and tracking, recognition and labeling may proceed in a piecemeal fashion. A graphical rendition might thus be a simple and coarse outline, tracked by local trajectories Assuming there is leftover bandwidth, it may be used to identify and correlate scene pieces with the known model, which eventually will generate the displayed scene entirely, and tracking of the target would switch over into a model-oriented scheme. Stability of the tracking can be maintained using only a few regions of later pictures.

REFERENCES

[1] Akin,D.L., M. L. Minsky, E.D Thiel, and C. R. Kurtzmann, NASA Contractor Report 3734, *Space Applications of Automation,Robotics and Machine Intelligence* , Contract NAS8-34381, October 1983

[2] Ballard, Dana H. and Christopher M. Brown, *Computer Vision* , Prentice-Hall, Englewood Cliffs NJ, 1982

[3] Barnard. S. T. and W. B. Thompson, "Disparity Analysis Of Images", Technical Report 79-1, Computer Science Department, Univ. Minnesota, January 1979 ([2],208-210)

[4] Brice, C. and C. Fennema, "Scene Analysis Using Regions", *Artificial Intelligence 1* (3).205-226, Fall 1970 (described in [2], pp. 158-159)

[5] Freitas, R.A. and W. P Gilbreath (eds), *Advanced Automation for Space Missions* , NASA Conference Publication 2255, 1982

[6] Horn, B. K P., and B. G. Schunk, "Determining Optical Flow", AI Memo 572, AI Lab, MIT, April 1980 (described in [2] pp. 102-105)

[7] Iverson, Kenneth E., *A Programming Language* , John Wiley 1962

[8] Nazif, A. M and M. D. Levine, "Low Level Image Segmentation: An Expert System", *IEEE Transaction on Pattern Analysis and Machine Intelligence 6* (5):555-574, September 1984

[9] Plunkett, David J. and Michael J. Bailey, "The Vectorization Of Ray-Tracing", IEEE Computer Graphics and Applications *5* (8)·52-60, August 1985

[10] Prager, J. M.,"Extracting and Labeling Boundary Segments In Natural Scenes", IEEE Transactions on Pattern Analysis and Machine Intelligence *2* (1) 16-27, January 1980 ([2], 85-88)

[11] Requicha, A. A G.,"Representations For Rigid Solids. Theory, Methods, and Systems", *Computing Surveys 12* (4), December 1980

[12] Roth, Sott·D.,"Ray Casting For Modelling Solids", *Computer Graphics and Image Processing 18* ,109-144 1982

[13] Tanimoto, S. and T. Pavlidis, "A Hierarchical Data Structure For Picture Processing", *Computer Graphics And Image Processing 4* (2):104-119, June 1975

[14] Vinz, Frank L., Linda L. Brewster, and L Dale Thomas, *Computer Vision for Real-Time Orbital Operations* , NASA Technical Memorandum TM-86457, August 1984

APPENDIX A - Raycasting And Constructive Solid Geometry

Raycasting is a deceptively simple computer graphic rendering technique, which consists of mathematically constructing a line from the viewer through each of the viewscreen coordinate positions, and determining what part of a scene or model each hits. If the object so hit is solid, then if its surface normal can be determined at that point its shading (knowing the direction from which light is coming) may be determined, thus determining the desired intensity for that pixel on the rendering. This may be considerably expanded Another ray may be cast from the point of intersection toward the light source, and if it should hit some object in the scene then the pixel through which the original ray was cast should be darkened to represent shadowing at that point. Further, if the object should be transparent, and possibly refractive, further rays, appropriately bent, may be cast from the point of intersection to determine just what light the viewer should see at that point.

Raycasting in and of itself is rather straightforward and lends itself to practically all forms of light modelling Of more challenging interest is the determination of intersection points, or how we know where some ray actually hits. For arbitrary objects represented by some volumetric technique such as oct-trees this can be quite exhaustive. We have chosen a more tractable means of constructing objects from simple geometric solid primitives which is known as constructive solid geometry (CSG) This and alternative models are discussed in an extensive review by Requicha. [11] Transformed primitives may be combined using the set operations of intersection, union, and differencing, and quite complex objects may be so constructed This appears to be a useful model for orbiting manmade objects, since they tend to comprise relatively simple forms in combination. Some moderately simple examples are shown below. They are copied from Roth [12], to which the reader is referred for a highly competent treatment of raycasting and CSG.

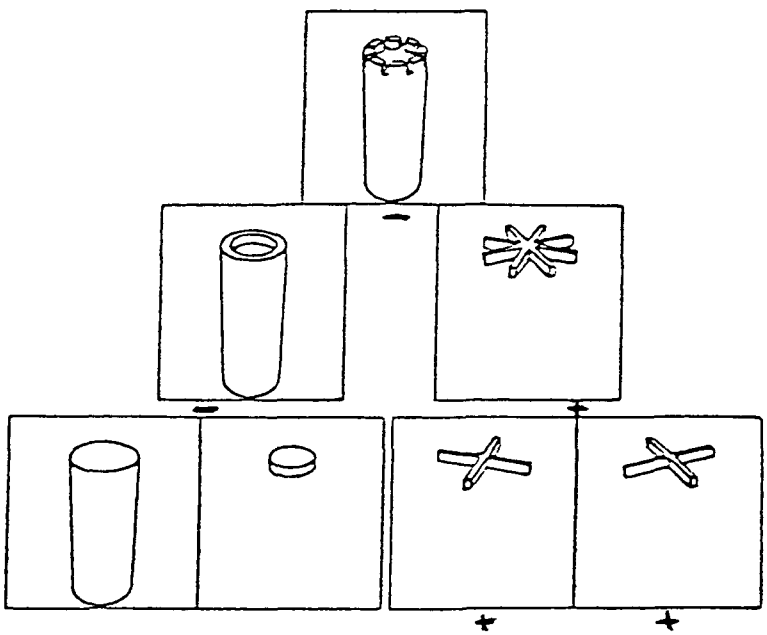


FIG 2 Example of composition tree.

Current Implementation

A reasonably competent CSG model works with five primitives, the cube, the sphere, the cylinder, the cone, and the torus. This effort has implemented only the first three, although addition of the other two (and, indeed, any other formally tractable object) is reasonably straightforward and truly additive to the existing software. Only matte lighting (single light source) and shadowing are currently implemented. As well, monochrome (colorless) objects alone are treated. These properties are sufficient to represent a useful range of targets.

An object is constructed as a binary tree, specified in pre-order (root, left child, right child), where every node is of degree two or zero. All interior nodes are combinations (intersect, union, difference), and are of degree two. All leaf nodes are primitives and are of degree zero. At any node one or more geometric transformations may be applied which will affect that node and all its children, if any These include translation, scaling (stretching), and rotation.

The current program reads a file of extension .csg, which consists of comment lines (begin with semicolon), node names (in quotes), or transformations (again, in quotes) to be applied to the most recently encountered node (and its children). Each transformation specifies three parameters. A primitive telescope file specification is shown in Exhibit 3. The object is shown in three different views in Exhibit 5, in which object outlining and shadowing are selectively illustrated. The output of the raycaster is a raster file which is fully competent for display on a raster graphic terminal. Since none is available locally, the single intensity dot-drawing capability of the vector terminal is utilized to make a primitive half-tone rendering, described in Exhibit 4.

The raycasting technique competently renders smoothly varying shading, shadowing, and horizon effects, all of which will present challenges to any scene interpretation procedure, and it does so in a concise and unified manner. Specification of an object is considerably more concise and intuitive in CSG than in a surface patch model, the other widely-used alternative. Further, it is a truly three-dimensional model, entirely formal, and thus a highly suitable candidate for matching to two-dimensional scenes.

The current program reads an object specification from FOR010.DAT, into which one has previously copied a *.csg file, accepts specification for boundary lines, shadowing, and display size (default of 256 x 256), and in 5 to 20 minutes creates a file in FOR008 DAT. If the file is acceptable, it should be copied to a *.ras file. The program rplay will render the contents of FOR008.DAT to a Tektronix using the ddot routine.

At the outset specific recognition should be given to existing vector languages, of which only APL is truly deserving of the name. The techniques and operations described in its original presentation (Iverson [7]) have been embodied substantially in the design of certain vector architectures such as the Cyber 205, and to an unfortunately lesser extent in the FPS Math Library.

All vector processors, and parallel Pascal and PL-1 as well, concisely express element-by-element operations. Thus, the statement

C = A + B (in PL-1) or c <- a + b (in APL)

will assign the sum of the i'th elements of A and B to the i'th element of C. It is assumed, implicitly in APL and by specification in PL-1, that A,B and C are all of the same magnitude, otherwise the operation is undefined.

Aggregate operations, such as sum, maximum, or mean of vector elements are treated in PL-1 with library functions. In APL basic reduction operators are used Thus,

X = SUM(A) (in PL-1, X a scalar) is x <- +/a (in APL)

where / is the reduction operator.

A capability of APL (and the FPS as well) is the logical vector While IF A > B THEN... is undefined in PL-1 for vector operands, the operation z <- a > b in APL creates a vector z of elements zero and one, of same magnitude as a and b, where z(i) is one if a(i) > b(i), zero otherwise.

To this point the FPS library embodies all these functions. The departure is selection/compression. In APL, a logical vector may be used to select elements from another arbitrary vector (of same length), producing a new and generally shorter vector consisting only of those elements of the operand vector that corresponded to 1's in the selection vector. This sort of arbitrary selection is endemic to a lot of vision routines, and is so far the only routine that we would implement outside the math library.

The image must in all cases be processed in pieces, because most procedures create too much temporary data to be held by the four pages of the FPS. A 256 by 256 image occupies exactly one page The Prager edge amplifier is a global/local technique, in that operations on a pixel depend only upon neighboring pixels (local), and all pixels may process independently and in parallel. This routine works, in fact, upon the edges between pixels, of which there are twice as many, so that two pages are required to hold edges. One step in the Prager amplifier is, for each edge, to determine at each end whether 0, 1, 2, or 3 other edges abut the edge (recall that each edge has three neighbors at each end), based on their strengths From this information the strength of the edge will be raised, lowered, or left alone A part of this operation requires ordering the strengths of the three neighbors, a small sort, and simply to do the sort requires temporarily maintaining over six numbers for each edge, which clearly outstrips memory capacity.

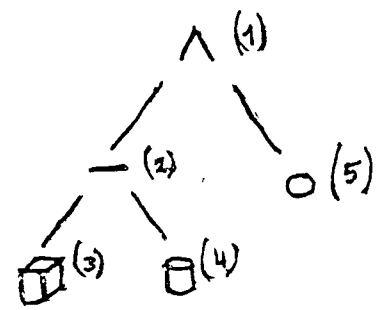
Thus, operations must necessarily be performed on subdivided images (the mosaic problem) if one wishes to retain the entire picture This applies as well to the Horn and Schunk optical flow routine, which requires a permanent data structure of six to eight pages exclusive of temporary memory. While the mosaic problem is a major irritant, the more fundamental objection to global processing is its speed. A single step of an iterative process on the whole image will take many seconds, which is clearly too slow.

The most satisfying route through the timing impasse is some form of data reduction, already discussed in the body of this report. Two forms are considered. The first is to work at reduced resolution, either by aggregating (averaging) into a resolution pyramid (Tanimoto [13]) or simply selecting systematically spaced pixels into a smaller picture. Working at reduced resolution serves to identify areas of the picture which require further, high-resolution analysis.

The second is similar, in that it begins by identifying interesting areas (cf. the Barnard-Thompson optical flow procedure), and then works only within them. The major challenge in either case is that procedures at this level work by updating tables and lists, a form of computing that departs substantially from traditional vector problems. This is currently under intense scrutiny.

```
ty simp.csg
; A simple object-- that part of a thick board that lies
; within a sphere. The board has a hole in it. The calculus
; is intersect big sphere with (board minus cylinder).
1 { 'int'
    'sca' 100 100 100     blow up to "world" size
    'rot' 20 20 0          and tilt some so we can see hole
  ;
2 [ 'dif'  left subtree, board(flat cube) minus thin cylinder
3 { 'cube'
    'scale' 1 .2 1      flatten it in y axis
4 { 'cylinder'         this makes hole in board
    'scale' .3 1 .3
    'translate' .25 0 -.2  not in center of board
  ;
5 { 'sphere'    right subtree, big sphere
    'sca' 1.1 1.1 1.1      leave some straight edge on board
    'end'
```

(a) A file (simp.csg) that describes a simple, three-primitive object, with two connectives.
The tree at the right is numbered to show correspondence.



(b) Running the ray-caster. Boundary lines are only for easier visualization, not test images.
Shadowing should be specified for test images.

```
cop simp.csg for010.dat
$ run rayc
enter non-zero if display run
0
type nonzero if boundary lines explicit
1
 type non-zero for shadowing
0
  enter size (square) if not 256
130
```

(c) The file (simp.ras) may be fed to program rplay, which makes this picture.

What simp.csg makes

Exhibit 2 - A Simple CSG Specification, and The Result Of 'Playing' It On A Vector Graphic Terminal.

```
;       a rudimentary space telescope, based on a picture
;       in the hall.
;       Three parts to this, main barrel, funny little flap
;       at end, and the solar panels.
;
;       mcan 26 Aug 85
;
'uni'
'sca' 50 50 50
'rot' 0 -50 0
'rot' 40 0 0
'rot' 0 -10 0
'tra' 0 0 30
;********** you fly this thing using previous transform
;
;       that was the main global transform (whole object)
;       all the following are relative to origin
;
;       main barrel
;
'uni'
'cyl'
'sca' 1.5 3 1.5
'uni'
;
;       these are two end collars
;
'cyl'
'sca' 2 1.5 2
'tra' 0 -1.5 0
;       second collar is hollow
'union'
'tra' 0 1.5 0
'cyl'
'sca' 2 1.5 2
'cyl'
'sca' 1.7 2 1.7
;
;       funny little flap next
;
'union'
;       well actually union of flap and panels
'union'
'rot' 90 0 0
'tra' 0 3 .5
'cyl'
'sca' 2 .1 2
'cub'
'sca' 2,.1,1
'tra' 0 0 -.5
;
;       ok, here's the panels
;
'union'   of two slabs and a stick
'rot' 45 0 0
'cyl'
'sca' .05 4 .05
'rot' 0 0 90
;       now two slabs
'uni'
'cub'
'sca' 2 5 .05
'tra' 2 0 0
'cub'
'sca' 2 5 .05
'tra' -2 0 0
'end'
```

Exhibit 3 - A Constructive Solid Geometry Specification of Primitive Telescope

The tree is specified in pre-order form. Lines beginning with ; are comments. Leaf nodes are 'cube', 'sphere', and 'cylinder' ('cub','sph','cyl'). Internal nodes are 'union','intersection', and 'difference'.

Any node may be followed with unlimited number of transformations, which are applied in order. The line 'end' is necessary at present to terminate tree.

```
c
c       ddot(ix,iy,den) - mean 3 jul 35
c
c
c       fix 19 jul - use log tab
c
c       half-tone dot drawer- draw only if supplied
c       density is below next random no.
c
c       the sense would be reversed if we were
c       drawing white on black
c
        subroutine ddot(ix,iy,den)
        integer*4 seed
        real e
        data seed,e/33517243,2.71828/
        d = log10 ( (den*(49) + 1) )/ log10(50.)
        if ( d .lt. ran(seed)) call pntabs(ix,iy)
        return
        end
```

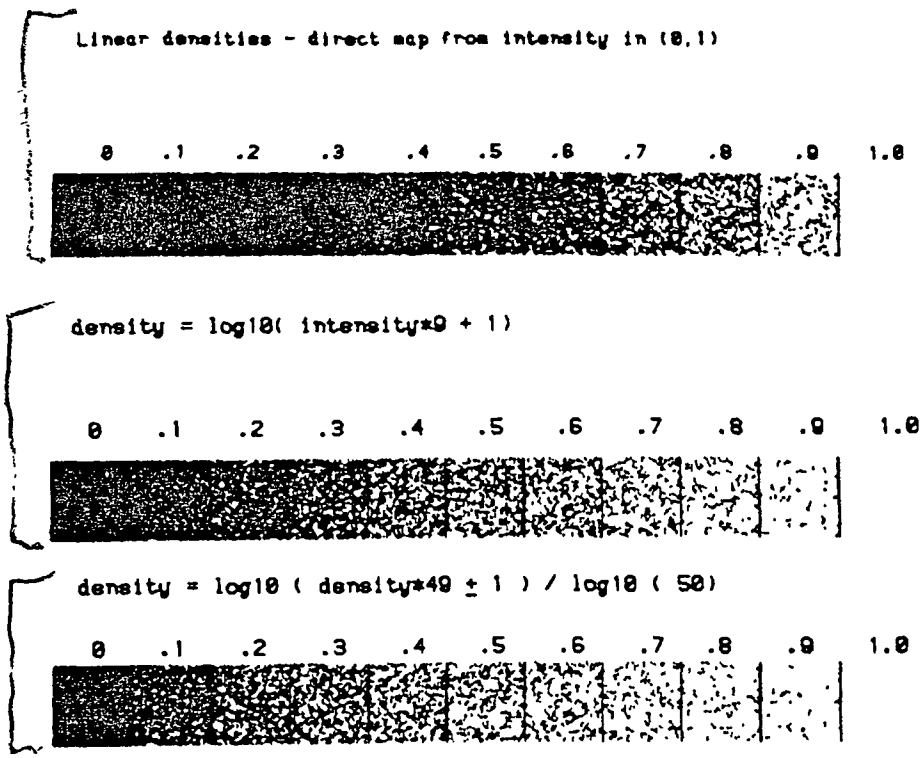
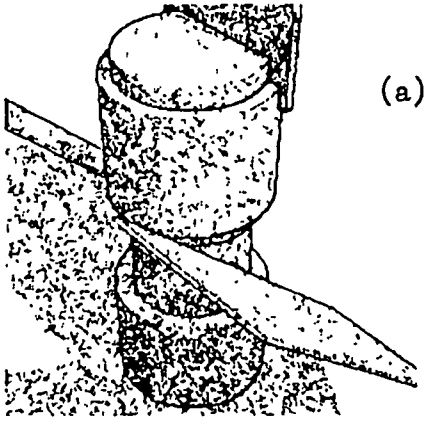


Exhibit 4 - Raster Graphics On A Vector Terminal

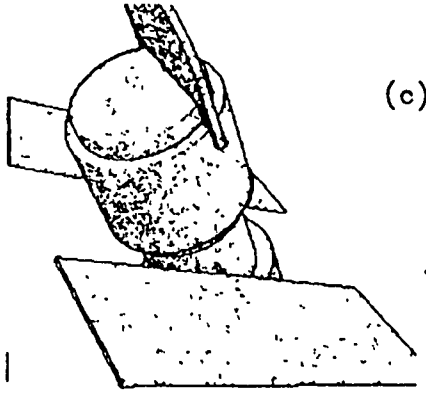
It is much easier to check a picture program by eye than by data listing. The routine ddot() listed at the top illustrates how pixel-independent half-toning works. pntabs() will draw a single dot, based on the supplied intensity and the next random number.
Several logarithmic transformations were tried, the last judged by eye to provide best separation on plotter.



(a)

View of Primitive Telescope, Lines Added

Same telescope, twisted to its right and omoving away from you. Also, boundary lines not added.



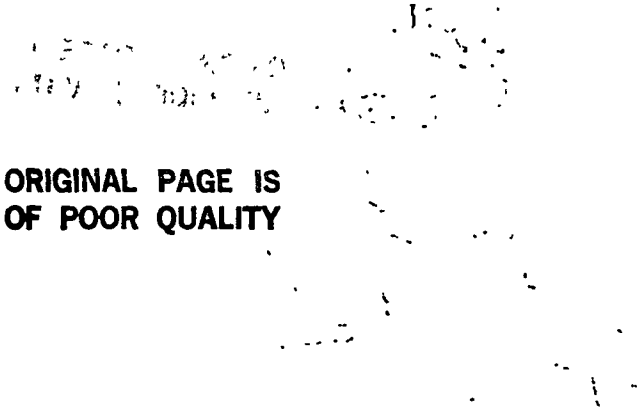
(c)

A third view, tilted toward and twisted a bit more, and father away. All this flying is done with the 'rot' and 'tra' lines at top node, nothing else. Just six numbers.

Exhibit 5 - Three Views Of A First-Draft Impressionistic Telescope

The raycasting program offers the options of boundary lines between different primitives, and the casting of a feeler for shadows. The three views here have the target moving away from you, turning to its right, and dipping toward you. The first view uses shadows and bound lines, the second shadows only, and the third bound lines without shadows. On a raster display the boundary lines would not be necessary.

Same telescope, twisted to its right and omoving away from you. Al;so, boundary lines not added.



Zeroith Pass - Raw Differences

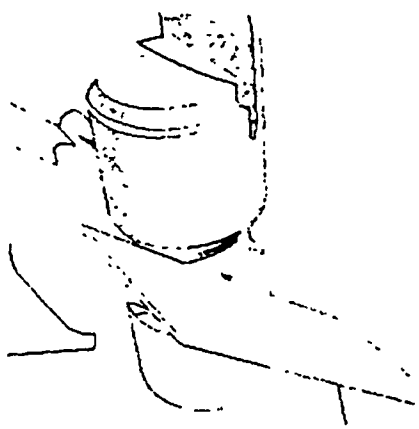
2828 148 3768

First Adjustment Pass

577 270 3699

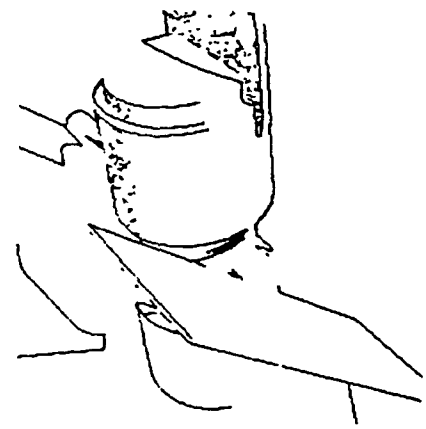
Second Adjuistment Pass. Shadow lines show best, since mostly on light background, hence began with strongest gradient

771 266 3312



Third Adjustment Pass

201 306 2280



Fourth Adjustment. Note right hand corner of distal panel visible for first time.

Exhibit 6 - The Prager Edge Amplifier At Work

This begins with the same view of the previous Exhibit, (b), with shadows but no boundary lines. The three numbers are single edges that have been downgraded, left alone, and upgraded during the pass.