AN INTERACTIVE MICROCOMPUTER BASED 3-D ANIMATION SYSTEM

R. Hackathorn, R. Parent, B. Marshall, and M. Howard

Ohio State University

ABSTRACT

A recent effort at the Ohio State University has been to implement a subset of our current animation system, called 'ANTS', on a DEC PDP 11/23 microcomputer. ANTS (Animated Things Through Space) is a high performance, interactive, 3-D color shaded animation system currently running on a DEC VAX 11/780. Its simple but flexible command structure, combined with its ability to procedurally model objects, allows a non-computer oriented user to generate, animate and display very complex animation sequences.

The new microcomputer based animation system using the 11/23 costs considerably less than the cost of the VAX system. Further, the PDP 11/23 could be replaced with a lower performance PDP family CPU that will run slower than the 11/23 but perform the same tasks with little or no code modifications.

A discussion of the design possibilities considered during the planning stage of the 11/23 ANTS system is presented. A comparison between the VAX's and the 11/23's CPU architecture, operating systems, and system peripherals is made. An explanation of what decisions were finally incorporated into the 11/23 based ANTS system is given. A comparison of cost/performance characteristics between the VAX 11/780 ANTS system and the PDP 11/23 ANTS system is included.

RÉSUMÉ

Un des services de recherches de l'Université de l'État d'Ohio a récemment mis au point un sous-ensemble du système d'animation actuel "ANTS" (Animated Things Through Space) sur un micro-ordinateur DEC PDP 11/23. Le système ANTS est un système d'animation interactif à 3-D, à rendement élevé et à couleur ombrée piloté par le DEC VAX 11/780. Ses commandes simples et souples, jointes à sa capacité de former des objets par traitement permet à l'utilisateur qui n'est pas familier avec les ordinateurs de produire, animer et afficher des séquences d'animation très complexes.

Le nouveau système d'animation piloté par le micro-ordinateur 11/23 coûte beaucoup moins cher que le système VAX. De plus, le PDP 11/23 peut être remplacé par un CPU du groupe PDP de rendement inférieur qui fonctionne plus lentement que le 11/23 mais qui peut effectuer les mêmes tâches avec peu ou pas de modification de code.

L'exposé porte sur les capacités de l'appareil qui ont été considérées pendant la planification du système ANTS 11/23. Le document fait également une comparaison entre la structure, les systèmes d'exploitation et les périphériques du VAX et du CPU 11/23. Il donne également des explications sur ce qui a finalement été incorporé dans le système ANTS piloté par le 11/23. Enfin, il comprend une comparaison des caractéristiques coûts/rendement entre le système ANTS VAX 11/780 et le système ANTS PDP 11/23.

INTRODUCTION

Three dimensional computer generated animation might well prove to be the most versatile and creative new medium developed since the motion picture camera was first used to make films over 100 years ago. Current graphics research has turned the computer into a sophisticated and unique tool for producing high quality color animation. With conventional animation, an animator can draw imaginary worlds that can defy or obey the laws of natural physics. Realistic scenes can be animated, but most animators are not prone to draw and redraw the complex images on each separate frame. In 'real world' cinematography, the filmmaker does not have to worry about the complexity of his subject matter, but must obey the laws of physics. Computer animation combines the best of both these mediums by generating highly complex imagery from simple object descriptions, and allows the user to selectively obey or defy whatever physical laws that have been coded into the animation system. Moreover, it has the ability to move the camera's 'eye' throughout a constructed environment with a degree of freedom found in no other art form. Unfortunately, animation of this sort requires enormous computer resources and exotic display devices, all of which are very expensive to purchase and maintain. Because of this artists, filmmakers, and students usually find only limited access at best to facilities with the necessary equipment.

Microcomputer technology has long held hope as a possible cost effective solution to producing computer animation. Microcomputers are small, energy efficient, inexpensive, and require little maintainance outside of good sense. They have also been slow, memory limited, and have provided poor software support when compared to mini or 'maxi' computers. Newer microcomputers, however, have overcome these disadvantages, making a low cost powerful 3-D animation system possible, when combined with the right software and peripherals.

This work was supported in part by the National Science Foundation Grant Number MCS 7923670.

The Computer Graphics Research Group (CGRG) at The Ohio State University has recently implemented such a system using a PDP 11/23 microcomputer running the ANTS animation software designed by Hackathorn and Parent (1). ANTS is the current version of at least six generations of animation systems dating back to 1965. One of ANTS key features is its ability to procedurally generate highly complex data models such as trees and cities, which can contain millions of surface triangles, if necessary. (Marshall, Wilson, Carlson- 2)

THE ANTS ANIMATION SYSTEM

ANTS (Animated Things in Space) is a 3-D computer-based animation system designed for artists and students. It is intended to facilitate the production of a series of computer-generated images which can be stored on disk or magnetic tape, digitally edited and reassembled for recording on 16 mm film or 1" video tape.

The first complete ANTS system has been implemented on a VAX 11/780 minicomputer with a 32-bit word length. The display hardware currently consists of a 512 x 512 frame buffer with 10-bit pixel pointers into a 24-bit color palette. A new 32-bit color pixel frame buffer is being developed by Crow, Howard (3). The final image is displayed on a RGB color monitor.

In order to give the user a powerful and blexible tool for animation that is still understandable to non-computer types, ANTS, as implemented on the VAX incorporates several significant features into one integrated system.

1) The animation system processes 3-D surface descriptions of objects as well as point and line data descriptions.
2) A very large number of object 'instances' can be created and animated with a unique two level internal control structure.
3) All commands in ANTS are optionally executed in a manner which permits the writing of animation scripts in a structured programming environment.
4) ANTS can be used for many graphical and non-graphical user tasks that would normally be programmed with an advanced high level language such as PASCAL or 'C'.

The system allows the user to concisely control the animation of complex imagery and

specify complex movement.

The system is interactively controlled by commands written in a special high-level language. In this language, the user specifies the viewing parameters, sets up object definitions, and orchestrates the spatial-temporal relationships between objects.

To help provide an interactive environment, the system allows the user to request the calculation and display of an arbitrary frame in an animation sequence or to specify the calculation of an arbitrary sequence of frames. Further, a variety of scanning algorithms exist which range from a quick rough scan to a slower high resolution, anti-aliased image. The ANTS system was designed with several specific capabilities in mind for a 3-D computer animation environment. The desire for these capabilities has evolved from several years experience in computer animation.

We are very concerned about the user interface simplifying motion specification with a need for relatively fast feedback. Fast display algorithms have been emphasized, at the expense of image quality. ANTS represents the latest in our attempt to improve on both the animation environment provided for the user and the quality of the images in the sequence.

In order to minimize the effects of any culture shock encountered by users of our system who come from non-computer science backgrounds, an added general consideration is that the language be easy to learn and the system easy to use by non-programmers. Our group has traditionally been interdisciplinary in nature with a heavy emphasis on users with an artistic background. We have always been interested in developing friendly and 'habitable' animation systems rather than graphical subroutine "add-on's" for extant computer languages.

The main purpose in developing a new animation system was to include in the design of the system a number of advanced techniques for handling complex graphical data. One of the most important of these was a facility for setting up procedure models for building graphical data bases. These procedures can be used to either simplify alterations to a large data base by altering parameters to the procedure, or to efficiently store a compact description of the data base.

Another major improvement over our past animation systems was sought in terms of the complexity of the animation which could be handled practically by this system. One facet of this objective was to include mechanisms to handle a large number of instances of an object concisely. For example, if one thousand identical leaves on a tree were desired in an animation, the requirement that each instance be animated with a separate set of commands would be laborious and time-consuming. By including a suitable mechanism in the language we can allow the user to establish algorithmic control over a large number of instances with relatively few animation commands. (See Plate #1).

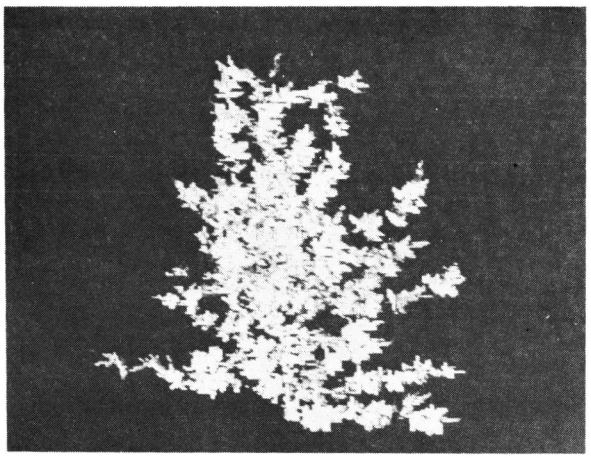


Plate #1
Bush

THE ANIMATION ENVIRONMENT

CGRG is an interdisciplinary university-based facility which currently supports an active graduate research program with the Computer Science Department at OSU, and an equally active animation program with the Art Education Department.

What we desired were relatively inexpensive work stations where students could learn to use the ANTS animation system and prepare animation scripts off-line from the VAX. These scripts could then later be run on the VAX to produce a final high quality animation sequence. The basic work station would consist of a fast 16 bit microcomputer, 128 KB of memory, a video terminal, dual floppies, a data tablet

digitizer, a 256 x 256 bit color frame buffer and a standard color TV monitor (See Figure #1). An improved version would include a multi-megabyte "Winchester" disk and also a 512 x 512 x 8 bit color frame buffer which could double as a 256 x 256 x 32 bit depth buffer for doing hardwired 'Z' compares for the surface calculation.

The first major decision was the choice of microcomputer for the work station. We first considered a home computer type micro, since many are available and tend to be relatively inexpensive. Though this is a very new field, many impressive products are being offered, such as the APPLE computer or Cromenco system. These systems offer minimal software support with the exception of the Bally home computer which runs a version of the GRASS language developed by Tom Defanti (4) called ZGRASS. Unfortunately all these systems use 8 bit microprocessors with very limited display capabilities. We finally decided that they would not be appropriate because of limited arithmetic ability and slow data throughput.

A custom made 16 bit microprocessor is at the heart of the XEROX 'SMALLTALK' system based on Baecker's GENISIS design (5). This would certainly be an ideal solution in terms of a cost effective system, but we do not have the resources for a major research and development task such as this. Several powerful 16 bit micro's were currently available from the big integrated circuit manufacturers. Unfortunately, microprocessors such as the Zilog Z8000 or the Motorola 68000, two chips which show the most promise in the desired cost/performance tradeoff, have yet to be bundled into a full-blown microcomputer system with adequate support software and peripherals. At the time we began development, the two readily available choices were the Texas Instruments 990 family and the DEC LSI-11 family. We finally chose the 11/23 because of our previous experience with DEC equipment and because of its software compatability with both the VAX and our 11/45. Besides, we had a big investment in assembler level software on both previous machines, and intended to modify existing code as much as possible to get the initial animation system up and running quickly.

Further, the 11/23 has many desired features: fast 16 bit cpu, memory expansion up

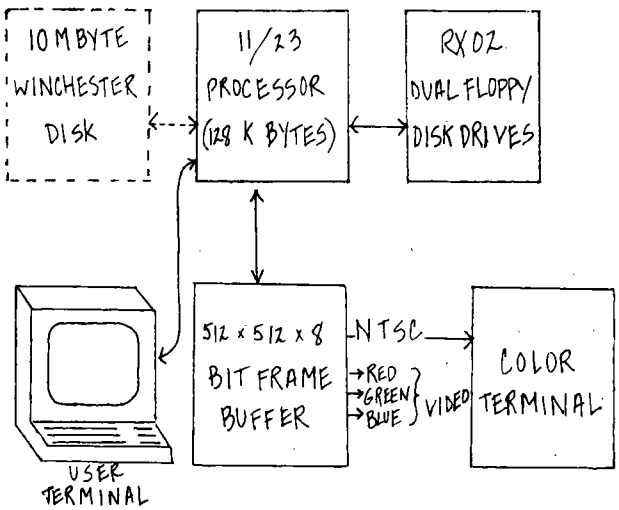


Figure #1
System Configuration

to 256 kilobytes, powerful assembler instruction set, and an excellent range of off-the-shelf peripherals at competitive prices. Also, the 11/23 is part of a family of computers which permit slower, less expensive CPUs to be substituted for possible home use or faster, more expensive versions for a possible stand alone production animation system.

Once we chose the 11/23 a decision had to be made about which operating system, if any, to use. The most direct approach would have been to develop all the necessary software on the VAX 11/780 and then download it to the 11/23. It would have the advantage of being the most memory efficient solution since there would be no operating system cluttering up precious addressing space. The approach would, however, require considerable extra software for file I/O editing and handling. Also, device drivers for all the peripherals would have to be written. Thus downloading without an operating system was decided not to be the best solution.

UNIX was another likely choice since we are currently using it along with the VMS operating system on the VAX 11/780. UNIX has an advantage of being a machine-independent operating system. Theoretically software which we develop for the DEC 11/23 could also be used on other families of micro computers running UNIX with only minor

changes. Unfortunately, a version of UNIX which could run on the 11/23 was not yet available.

Our choices were thus narrowed to a pair of DEC-supplied operating systems: RSX 11-M and RT-11. Both these systems would work well for the jobs required, but RT-11 was chosen, primarily because it is a simpler, more memory efficient system designed primarily to run one task at a time, which is exactly what we wanted. In addition, we were able to get a 'C' compiler to run under RT-11 that gave us compatibility with UNIX, as well as a powerful high level language.

ANIMATION SOFTWARE IMPLEMENTATION CONSIDERATIONS

Our next decisions were about the requirements for new software and the choice of a computer language. We decided that the work station animation system would be at least upward compatible with the VAX-based ANTS animation system. Therefore the first issue of which software to implement was already answered by our demands.

ANTS is neatly divided into three sections: control, animation, and display. At the highest level is the ANTS control section. This section needs tasks for script flow control, script parsing, and script execution.

ANTS control has two primary jobs:
1) system services, consisting of invoking a control or animation script, symbolic expression evaluation, listing of files or variables, help messages, status information, printing user messages, and prompting for user input.
2) scene setup, consisting of object identification, palette setup, frame buffer manipulation, observer eye position, center of interest, and light source point positioning. Also, it includes the setting of parameters for light source, color and observer qualities such as 'field of view'.

The animation section requires the same basic script handling routines as the ANTS control section. These include flow of control, script parsing, and animation command execution, as well as many of the same features as the ANTS control section, such as variable manipulation and messages. The animation section is generally concerned with controlling object transformations over

The third ANTS section is the display section, which is invoked after finishing the animation choirs. This section is responsible for turning a triangle into a blob of colored pixels on a TV screen. It has two major parts, a set of object to image space tasks, and also a set of tasks for scanning and displaying colored surfaces.

The object to image space tasks include clipping to the screen area, calculating the color for a face, and performing perspective. The typical Z-buffer scanning routines consist of orienting the triangle, converting it into pixels (tiler), and Z-comparing each pixel with a stored frame image to determine the highest surface pixel.

Naturally, the features from the large VAX ANTS system which control access to the mass memory devices and special display devices could not be transported to the microcomputer. However, a significant subset was still possible which could do all the basic animation tasks.

Which computer language to use on the micro was more a consideration for east of coding than for performance. In our experience we have found that the performance differences between a high level language and assembler are all but equalized when the tasks perform heavy I/O to a secondary storage device. In fact, considering the slow speed of the floppy disks, we could probably have written the 11/23 version of ANTS in COBOL or SNOBOL and hardly have noticed a difference. To allow for easier future expansion, the low overhead tasks such as the language parser were written in the 'C' programming language by Marshall, while the highly repetitive transformation and display routines were written in assembler by Hackathorn and Parent. The decision to use assembler was based largely on the fact that we had much of the code needed for the 11/23 already running on our 11/45 or VAX. We intend, however, to rewrite this section into the 'C' language at some future time.

SCANNING ALGORITHMS

Which scanning algorithm to use was by far the most critical software decision that was made for the microcomputer based ANTS project, because of the extreme slowness of

the floppy disk units. A major consideration was to develop an algorithm that was both computationally efficient and I/O efficient. For our purposes, there were basically two types of scanning algorithms available for us to choose from: list ordering and 'Z' compare. List ordering algorithms, such as those developed by Watkins (6) or Newell (7) fit the requirement of being I/O efficient since neither make use of a disk resident frame buffer in their visible surface calculations.

The Watkins algorithms sorts a list of 'span' segments along both the 'X' and 'Z' axes for each 'Y' scan line with the output going directly to a color frame buffer display. The Newell algorithm sorts a list of faces in 'X' and 'Z', breaking up any faces that overlap into smaller non conflicting faces. The new list is written into a color frame buffer last face first, with faces closer to the observer simply overwriting old faces.

These algorithms would work quickly on a moderate number of faces, but generally they have two drawbacks. First, since they require a sorting step (or several as the case may be) their execution time will grow non linearly as the data count increases. The second drawback is that these algorithms require internal storage of pre-sorted, semi-sorted and fully sorted lists of object data. All these lisss become memory and time limiting factors in data complexity on a small microcomputer system.

The drawbacks of these two algorithms bring up a second and equally important design consideration in the choice of a scanning algorithm. Our VAX ANTS animation system was designed to handle highly complex images. By 'stream processing' a single list of the original object data directly into a stored frame image, using a 'Z' compare algorithm, the VAX system has no theoretical limit to the quantity of data pumped through it.

Crow (8) has demonstrated that list sorting algorithms such as the two described can be carefully designed to minimize non-linear time growth rates and increases complexity handling capabilities to make possible pictures of 100,000 faces and more. The simples solution to higher complexity with these algorithms is to store the required data lists on a disk and swap them into available memory. Unfortunately, this solution tends to nullify much of their advantage of not using the disk for the visible surface calculations in the first place. This left us with the 'Z' compare class of algorithm such as the one we used on our VAX ANTS animation system.

The 'Z' compare or 'Z' depth buffer algorithms are the simplest type of visible surface algorithms available. The closest surface to the observer can be easily found by comparing the depths of all the object image pixels with each other, keeping only the highest ones. As a pixel comes out of the tiler, another pixel corresponding to the same relative 'X-Y' location is retrieved from a stored frame buffer image. Since a depth or 'Z' value is stored with each pixel color in the frame buffer, it becomes an easy matter to compare the two depths and update the stored frame image with the color and 'Z' of the closest pixel to the observer. The combined effect of both high computing costs and huge memory requirements has mostly been relieved by current fast mini computers and ever dropping memory costs. However, on a micro computer system, these issues can still be a big problem, largely because of limited direct addressability of main memory and the use of floppy disks for secondary storage.

One 'Z' compare algorithm, first implemented by Myers (9) attempted to solve the problem of a disk resident frame image by processing all the object faces one scan line at a time into a single, memory resident scan line 'Z' buffer. But because all the faces must be in memory at once, there becomes a very definite upper limit to the data complexity of an object, determined by the size of available memory. Of course, this limit can be extended by reloading data to and from the disks, but again, this defeats the algorithm's principle advantage to a microcomputer system, which was to use the floppy disks as little as possible for the visible surface calculations.

The VAX ANTS system used a direct brute force 'Z' compare approach that we felt would be inappropriate to our micro environment. On the VAX, for each tiled pixel, the display algorithm would directly access a disk resident 'Z' buffered image. This was possible becuase of the VAX's very efficient virtual memory mapping capability that allowed us to treat the disk frame buffer file as though it were part of our internal program address

space. Since it is unlikely that we will see a microcomputer system in the near future with the virtual memory efficiency of the VAX, this approach could not be considered practical because of the algorithm's excessively heavy disk usage.

One possible approach to using a full disk based frame buffer was to keep a kind of 'expandable' combination pixel/runlength frame buffer in main memory In this scheme, the memory resident frame image would begin as a single very long runlength of background color. This runlength would represent all the pixels contained from pixel #1 of the first scan line to the last pixel of the last scan line. As pixels come from the tiler, new pixel locations would be created in the frame buffer image to describe only those pixels which indicate the highest surface element at that pixel location. All other pixels would remain compressed in the background colored runlength format. When the frame buffer image expands to the limits of internal memory, the disk based frame image would be updated with new information from the memory based frame buffer. Then the memory buffer would begin being filled up once again. While this approach does minimize the access to the secondary storage devices, it also is a much more complicated algorithm than we desired on the micro. One big problem to overcome would be the ability to randomly access the memory buffer. Because of its expandable nature, a particular location within the buffer would probably require a different offset address each time after the buffer expanded.

Perhaps a more preferable approach was to explore efficient buffering methods for the object data instead of trying to buffer only the frame image. One algorithm considered used a runlength buffer scheme developed by Hackathorn (10). In this algorithm, runlengths are created in the tiler as usual, but instead of trying to compare the run's pixel 'Z's immediately, they are stored in a runlength buffer first. The runlengths are stored ordered in 'Y' using a simple radix sort, but are not ordered in 'X'. The runlength buffer was a list of fixed memory arrays, one array for each scan line of the display. As a runlength was created, the corresponding memory array was indexed within the runlength buffer and the runlength itself stored as the last element of that array. A list of "runs per array" was kept with the buffer to indicate where a run should be thrown and also when a buffer was full. When a buffer was filled, a frame buffer section was brought in from the disk resident frame image and as many scan line arrays as possible were mptied into the buffer section by comparing the 'Z's of each pixel in each runlength. Once the scan line arrays are empty, the memory buffer section was written back to disk and the runlength buffers were ready to be refilled.

This algorithm was our first attempt at developing the 'stream processing' capability that ANTS makes use of to create high complexity images and generate procedural models. The algorithm was quite efficient and displayed linear growth rates as the data complexity increased even up to several million trinagles. However, memory had to be equally shared between all the scan lines which meant that each scan line could store only a small number of unordered runs before filling up and having to be emptied onto the disk.

A better way we felt was to buffer the runs unordered in both a scan line (or several as memory permits) from the disk resident frame buffer and search for any runlengths that fall within the memory resident 'Z' depth scan line buffer, comparing 'Z's and updating as we go along. Once the entire disk frame buffer has been cycled through, the unordered runlength would be empty and could be restarted. This method would be straight forward to implement and fairly quick provided that faces being scanned were relatively small. Large faces would, however, cause the run buffer to fill up quickly, because of the increased number of scan lines a larger face would cover. The way to get around this problem would be to store the faces in their compact 'ready to tile' form in the buffer instead of the faces' individual runs. This way, no matter how big or little a face is, it will always take up the same amount of buffer memory. Once the 'tile block' buffer is full, then it must be emptied in a manner similar to the run buffer just mentioned. To prevent access to a frame buffer scan line on the floppy which is not needed in the visible surface calculation, the actual read from the disk should not be performed until a bonified face is found which covers the scan line in question.

We chose a buffered 'tile block' scheme as the method to be implemented. This algorithm has many advantages to a small microcomputer environment. First, the buffering arrangement offers a great deal of flexibility in allocating memory within the precious micro addressing space. At the slowest level, the algorithm can run if there is memory for only 1 'tile block' (about 14 words each) and only 1 scan line buffer (about 256 words at 1 word per pixel) for the disk frame image to swap in and out of. Of course, this configuration will run very slow but it does make possible a simple implementation of a visible surface algorithm using practically no primary memory at all. As more memory becomes available it is a trivial matter to increase the size of these two buffers and dramatically increase the performance of the micro.

ANTS PERFORMANCE RESULTS

ANTS has been running on the PDP 11/23 for several months. Figure #2 shows examples of some of our preliminary results using test data. The objects were a small ball, the head from the man's body and 120 procedurally generated squares evenly spaced in 'X' and 'Y'. The information given with the timings shows the number of faces (given as a triangle count) and the number of scan lines the object(s) cover from top of the screen to the bottom.

There were five basic tests conducted on this set of data. First, ANTS was run with its internal memory buffers set to their absolute minimum. In our system this meant that 1 tile block and 1 scan line could be buffered at a time from the stored disk image. This test was similar to a direct 'brute-force' 'Z' compare in which the algorithm must access the floppy disk for each scan line of every face. This test was naturally the slowest of all five. The second test increases the number of scan lines from the disk image that could be buffered internally. This means even though each face is processed immediately as it comes from the tiler, the floppy access is slightly more efficient.

The third test does just the opposite of test #2. The tile buffer was increased to hold up to 200 faces before scanning was performed, but the internal scan line buffer could only hold one scan line at a time from the disk. Test #3 shows the importance of buffering the ready-to-be-scanned faces as opposed to only buffering the scan lines from the floppy. The main reason for this difference is that whereas the floppy must be read sequentially regardless of how many scan lines are kept internally, the tiled face buffer determines how many faces can be scanned at once into each accessed disk section. The time savings, therefore, come from making better use of the scan lines already read in, rather than trying to read more scan lines in with each access.

The fourth test shows the normal ANTS environment as it currently exists. Trial and error was used to determine the combination of 200 buffered tile blocks and 8 buffered scan lines as the most efficient time/memory tradeoff given our present hardware/software configuration. The last test used the memory buffer limits from Test #4, but shows how ANTS would do if it never had to use the floppy for storage of its depth frame buffer. Test #5 performs all the calculations of test #4, including transformations, lightsource and all scanning tasks including the 'Z' compares. Test #5 does not, however, perform any accesses to the floppy disk because the ANTS program was temporarily modified to think it's scan line buffer was always current with the stored disk image. This test dramatically underlines the point made several times in this paper that the major bottleneck in producing efficient computer animation in a microcomputer environment is not the speed of the micro, but rather the lack of speed in the floppy disks.

Since 1978 we have developed four versions of the ANTS animation system at CGRG. A prototype was first implemented on a PDP 11/45 using the RSX 11-D operating system. This used the 'buffered runlength' 'Z' compare algorithm and a fast 88 megabyte disk for its stored frame image. The most recent effort has, of course, been on the 11/23 microcomputer system running RT-11. It uses the 'tile buffered' 'Z' compare approach and a floppy disk for its stored depth buffer. Currently the most complete version has been on our VAX 11/780 running the VMS operating system, but all future work is going into a version on the VAX using the UNIX operating system. Both VAX ANTS versions directly access a 300 megabyte disk for each pixel using a 'Z' buffer algorithm.

	Little Ball	120 Squares	Little Head
Face count:	60	240	220
Scan lines covered:	150	200	170
Elapsed time (seconds)			
1 tile block + 1 scan line	165	605	231
1 tile block + 8 scan lines	74	452	137
200 tile blocks + 1 scan line	22	164	21
200 tile blocks + 8 scan lines	11	120	12
same but without floppy access	2	10	5

Figure 2
ANTS 11/23 Timings

It is difficult to make an exact comparison of the ANTS animation system as it is currently implemented on our various computers. First the actual implementation of ANTS differs from computer to computer. Because of our commitment to research, we have tried to modify and improve the system with each implementation. Each version of ANTS has a different set of parser and transformation tasks and more importantly each has different display algorithms.

Another major disimilarity between the different versions concerns the actual hardware involved in each system. The VAX is a high speed 32-bit minicomputer with hardware features such as a CPU employing ECL integrated circuits, a 200 ns. internal SIB bus, and fast cache memory. The 300 megabyte disk used by the VAX has an access time of around 30 milliseconds and a transfer rate of about one 16-bit word every 2 or 3 microseconds. The hard disk used by the 11/45, the RP04, has similar access and transfer timings. These disk times are in sharp contrast to our poor little 11/23 microcomputer system which uses a floppy disk drive. The floppy disk has an access time of 165 milliseconds and a transfer rate of about 82 microseconds per word if the disk is formatted single density or 51 microseconds if it is formatted as double density. In the future, the addition of a 'Winchester' type disk drive which has an access time of around 70 milliseconds and transfer rate of 4 microseconds per word, will of course help enourmously.

The tests on all the three computers were performed with the same scripts and the same data. The VAX ANTS versions use 32 bit floating point numbers throughout their calculations. Both the 11/45 and the 11/23 use 16 bit integer calculations. Further, the VAX uses a stored 32 bit floating 'Z' for each pixel while the 11/45 uses a 16 bit integer 'Z' and the 11/23 ANTS system uses an 8 bit 'Z'. To help keep the tests somewhat comparable, all tests were run at 256 x 256 resolution. Because UNIX and VMS share the VAX and its peripherals, the ANTS timings for each system are roughly the same. Therefore, we have included only one set of timings for the VAX.

name	faces	11/23	11/45	VAX 11/780
			seconds	
Little Ball	60	11	6	3
Big Ball	168	12	7	4
Little Head	220	12	8	4
120 Squares	240	120	-	65
Big Head	384	28	14	7
Goblet	420	24	-	13
Man Body	1644	100	-	36
6 Big Heads	2304	134	45	22
Random Balls	3600	210	-	-
Tree 1	10098	642	-	124
Tree 2	2189	393	-	73

Figure 3
Comparative CPU Timings

CONCLUSION

A microcomputer-based ANTS animation system has recently been implemented at The Ohio State University. It uses a PDP 11/23 microcomputer with dual floppy disk drives for secondary storage. The system serves as a stand along workstation for the development of animation scripts which can be run either on the 11/23 or on the VAX for higher quality. Several

art students including two new students with no previous computer experience have been working on the micro system. Plans for 8 to 10 students on two stand alone 11/23 systems are scheduled to begin Summer of 1981. Two additional 11/23 computers with frame buffers will also function as workstations for the VAX. The intended cost of the microcomputer system will be between $15 to $20 thousand per system with a full set of peripherals.

A major addition to the microcomputer ANTS system will be the completion of our new 512 x 512 x 8 bit color frame buffer. The display device, developed by Howard, will be used to make the 11/23 a complete stand-alone animation system. Currently we are using the 512 x 512 x 10 bit color frame buffer on the VAX by transporting the stored binary frame image from the 11/23 to the VAX via floppy disks. The new frame buffer will function as a normal 512 x 512 color display, but it will also have the option to double as a 256 x 256 x 32 bit color and 'Z' buffer. In this mode we will be able to send pixels directly to the frame buffer and have the 'Z' compares and conditional color updates performed in hardware rather than in software.

Another upcoming addition will be a 10 megabyte Winchester disk drive. Currently we are using the floppy disks and consequently we are very limited in the amount of disk storage space that we can utilize. Because of the memory restrictions, we are presently using only 8 bits for our 'Z' depth value. This allows a full featured visible surface algorithm with virtual intersection to be implemented, but it does limit the amount of spacial movement that we can accomplish. When the Winchester disk arrives, it will be a simple matter to increase the 'Z' to 16 bit space.

The PDP 11/23 has 256 kilobytes of addressable memory space. We currently have 128 KB of memory on the micro, but are using only the first 64 KB for this ANTS version. This has been primarily done out of concern to get a version up and running as quickly as possible, but also has the benefit of being a version of ANTS that could run directly on the smaller LSI-11/03 microcomputer system, which can only address 64 KB. As soon as possible we will be directly controlling the 11/23 memory management registers to give us access to the other 64 KB that we are not using. This extra memory will be used to increase the size of the 'tile block' and 'scan line' buffers. We expect this increase in buffer size to give us better overall performance calculating animation, since it can be clearly seen by the timing tests that an increase in buffer size translates into a decrease in calculation cost.

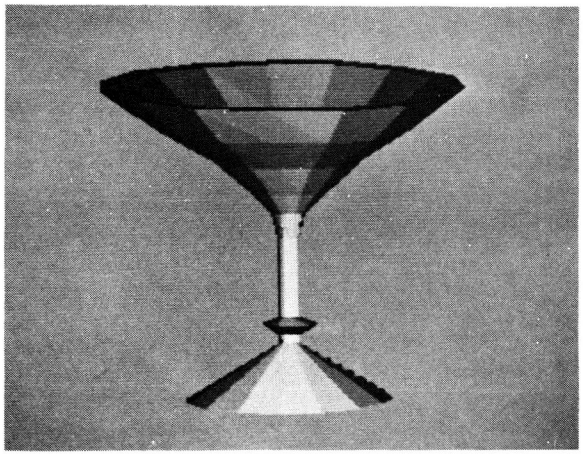
The major advantage of a language parser written in 'C' was that it could be debugged on the VAX under our UNIX operating system. However, it also meant that when we had a running version for the 11/23, we also had a running version on the VAX. By rewriting the display routines in 'C' we now have a version of ANTS working under UNIX. We intend to rewrite the assembler display routines on the new microcomputer system at some future date.

ACKNOWLEDGEMENTS

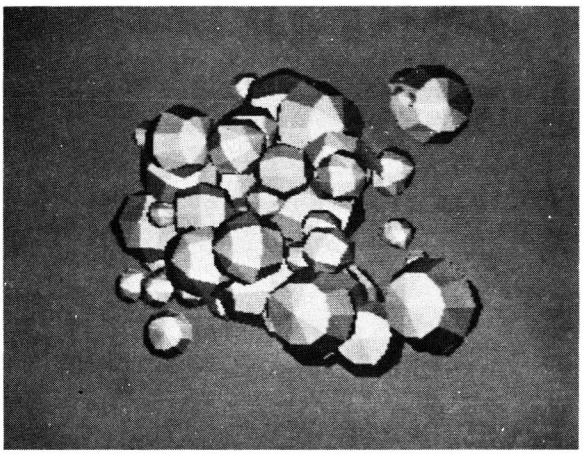
The authors would like to extend our appreciation to the members of CGRG for their helpful suggestions.

BIBLIOGRAPHY

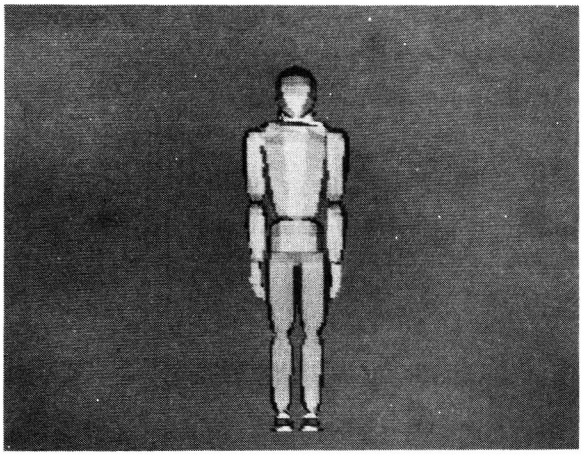
1. Parent, R., Hackathorn, R., "ANTS: A High Performance 3-D Color Animation System."
2. Crow, F., Howard, M., "A Frame Buffer System With Enhanced Functionality", SIGGRAPH 81.
3. Marshall, B., Wilson, R., Carlson, W., "Procedural Models for Generating 3-D Terrain", SIGGRAPH 80.
4. Defanti, T., ZGRASS Users Manual.
5. Baecker, R., "Interactive Computer Mediated Automation", Ph.D., 1969.
6. Watkins, G., "A Real Time Visible Surface Algorithm", Univ. of Utah/Ph.D., 1972.
7. Newell, M., "The Utilization of Procedural Models in Digital Image Synthesis", Ph.D. Dissertation, 1975.
8. Crow, F., "Shaded Computer Graphics in the Entertainment World", IEEE Computer Magazine, March 1978.
9. Myers, A., "An Efficient Visible Surface Algorithm", Tech Report OSU, 1974.
10. Csuri, C., Hackathorn, R., Parent, R., Carlson, W., Howard, M., "Towards An Interactive High Visual Complexity Animation System", SIGGRAPH 1979.



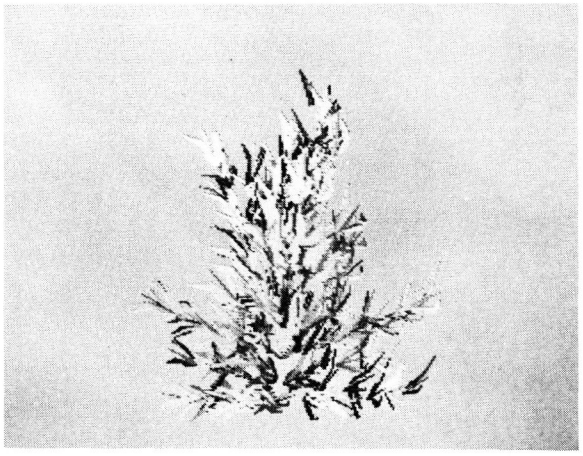
Sacrificial Chalice



Gamma Globules



Grad Student



Fuller Bush