Graphics Support for Cognitive Research in Human Factors

by

Michael F. Morgan

An essay
presented to the University of Waterloo
in partial fulfillment of the
requirements for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, 1985

Acknowlegements

I would like to thank Phil Bryden for providing me with a basic understanding of the work that has been done in mental rotation (both past and present). I would also like to thank my supervisor, Kelly Booth, for many valuable discussions concerning Watkins's algorithm.

Table of Contents

Graphics Support For Cognitive Research in Human Factors

Michael F. Morgan

Computer Graphics Laboratory
Department of Computer Science
Faculty of Mathematics
University of Waterloo
Waterloo, Ontario.

ABSTRACT

This essay describes collaborative work done with the Department of Psychology, for a series of experiments designed to investigate spatial imagination as it relates to the mental rotation of graphic representations of three-dimensional objects. Previous research into mental rotation, as well as the current work is described. The experiments required the generation of a large number of high quality images to be used as test stimuli. In order to produce these stimuli, new software was developed implementing a version of Watkins's algorithm. A review of Watkins's original algorithm and its complexity is provided. This is followed by a detailed description of the implementation used to generate the experimental stimuli.

1. Introduction

A goal of the Computer Graphics Laboratory has been to find new applications for computer graphics. This essay describes collaborative work done with the Department of Psychology to conduct a number of experiments designed to investigate human cognition.

Many experiments in Psychology require the production of a large number of test stimuli. Using conventional methods this can often be tedious. In some cases it is very difficult or even impossible to produce the desired stimuli using such techniques. The experiment described in this essay required the production of 1,872 photographics slides, an example of the first case. Other work being done in the laboratory involves the production of high frequency spatial masks, an example of the second case.

The work described in this essay is based on research in human cognition done by Shepard at Stanford University, Shepard (1971). Shepard's research was designed to investigate the process of mental rotation. Mental rotation is one type of thought used to determine if two shapes represent the same three-dimensional object. Researchers in the Department of Psychology at the University of Waterloo have been interested in studying this phenomena. Previous work done at Waterloo in this area had been conducted using wooden block models. The current study, however, required a large number of varying stimuli that could not be represented using the wooden blocks. The Psychology department could not easily produce these new stimuli by themselves. Therefore, computer graphics techniques were used to solve their problem. The flexibility of computer graphics allows the production of virtually any type of stimuli that could be desired. Furthermore, the production of large numbers of stimuli can be automated in a fairly simple way.

The specific requirements of the Psychology department were for 35mm slides of shaded and outlined (wire-frame) images. The objects in the images were arm-like structures composed of blocks, similar to the figures used by Shepard in his original work.

The z-buffer device in the GR graphics package was initially considered for rendering these images. However, the images that it produced suffered from aliasing, and it was thought that those artifacts might affect the results of the experiment. At the time that this project was started the only software available in the laboratory that could render high quality anti-aliased images was a ray-tracing package. It was not practical to use this package for such simple images because of the large amounts of computing time that would be required. This led to a decision to implement a version of Watkins's algorithm. The advantages of this were threefold. First, a new output device would be added to the GR graphics package. Second, it would provide a higher quality image than could be obtained using the z-buffer device without incurring the expense of ray-tracing. Finally, Watkins's algorithm could be applied to scenes in order to determine the visible edges that composed them. The edges could then be written to a virtual device, and subsequently displayed on other devices without incurring the penalty for visible surface calculations.

This essay discusses the basic aspects of the Psychology experiment as well as the details involved in the implementation of Watkins's algorithm. Chapter Two discusses previous research into mental rotation and describes the current work being done in the Psychology Department at Waterloo. Chapter Three describes Watkins's algorithm. The first section of that chapter introduces his original scanline algorithm. The second section gives a brief discussion of the complexity of the algorithm and describes some improvements that can be made to its running time. The final section of Chapter Three gives the details of the current implementation. Chapter Four describes how the stimuli were produced. Chapter Five discusses some of the preliminary results of the latest mental rotation experiment and describes future work in this area.

2. Mental Rotation

An area of current interest in psychology is the investigation of the mechanisms involved in human thought processes. Of particular interest is one mode of thought known as mental rotation. The first section of this chapter describes the pioneering research in this area done by Shepard at Stanford University. The second section describes current research into mental rotation that is being done by the Psychology Department at the University of Waterloo. This work has served as the motivation for this essay by providing an interesting application for computer graphics.

2.1. The Original Experiment

Shepard's experiments were designed to investigate the process of *spatial imagination*. Specifically, the goal of his work was to establish how human beings determine whether two representations of three-dimensional objects are the same or different. The original study done by Shepard and a graduate student, Metzler, found that the time required to recognize that two perspective drawings represented the same three-dimensional object was a linearly increasing function of the angular displacement between the orientations of the two objects. These results, along with information from post experiment interviews, led Shepard to hypothesize that the decision about "sameness" was made by *mentally rotating* one of the objects into congruence with the other, and then checking to see if the two objects were identical.

The mental rotation experiment is significant in the study of human cognition because it was one of the first attempts to investigate human thought processes in an empirical manner. Much of early psychology was based on introspection, and thus was subjective and qualitative. The behaviorist movement insisted that psychology must become empirical in nature, based on specific stimuli and objective responses. However, the problem of how to investigate mental processes in a quantitative manner remained unsolved. Shepard's experiment was designed to address, at least in part, this problem. He studied the human thought processes involved in spatial manipulation, an area previously unexplored. Shepard's work met the criterion set by the behaviorists in that it was objective and quantifiable; answers were either correct or incorrect, and response time was an easily measurable quantity.

In the actual experiment conducted by Shepard and Metzler, each of eight adult subjects was shown 1,600 pairs of perspective outline drawings that represented three-dimensional objects. The subjects were asked to decide as quickly as possible whether the two objects depicted in a drawing were the same. Two objects were deemed to be the same if one of them could be rotated into congruence with the other. Once this determination was made the subject pulled a right hand lever to indicate that the two objects were the same, or a left hand lever to indicate that the two objects were different. A warning tone was used to signal that

a pair of objects was about to be displayed. Presentation of the stimulus corresponded with the starting of a timer. Depression of one of the levers stopped the timer and caused the recording of the response time and the subject's answer. Half of the pairs depicted similar objects, which could be rotated into congruence, the other half depicted isomorphic pairs which could not be rotated into congruence. The reason for making half of the pairs isomorphic was to prevent subjects from making the decision of non-congruence based on some feature possessed by only one of the objects. Slide 1 shows two similar objects, these can be rotated into congruence. Slide 2 shows two isomorphic objects, these cannot be rotated into congruence.

Each figure used as a stimuli was composed of ten blocks that formed an arm-like structure with three right angle twists. This particular type of object was chosen because it is relatively unfamiliar and meaningless to most subjects. The twists were placed at strategic points along the structure to ensure that the object could not be rotated into congruence with itself. This condition is necessary so that there will only be one axis about which an object can be rotated into congruence with another similar object that is in a different spatial orientation.

In all, ten such objects were used. The ten objects were further subdivided into two distinct subsets of five objects each. These objects are distributed between the two subsets such that, each object's mirror object was contained in the other subset. These figures were then previewed in eighteen orientations that corresponded to 20° steps about a specified axis. Seven of these eighteen orientations were selected in order to avoid views in which some part of an object was occluded by another part. Furthermore, the selected orientations had to allow the construction of pairs of objects that differed by every increment of 20° between 0° and 180°.

The resulting seventy views were photocopied and placed on cards in pairs to be used as stimulus for the experiment. Half of the pairs were rotated in depth (about the x-axis), the other half were rotated in the picture plane (about the z-axis). Of the 1600 pairs used in the experiment, 800 were identical pairs. These were further divided into 400 unique pairs, twenty depth pairs differing by each of the ten 20° increments between 0° and 180°, and twenty picture plane pairs differing by each of the 20° increments between 0° and 180°. Each of these unique pairs was presented twice to make up the total of 800 identical pairs. The second group of 800 consisted of 400 unique isomorphic pairs that were each presented twice. Each of the isomorphic pairs corresponded to one of the identical pairs from the first group, except that one of the objects had been reflected about one of the three principle planes. The identical pairs and the isomorphic pairs were randomly mixed throughout the entire group of 1600 drawings.

The experiment was administered in eight to ten one hours sessions for each subject. The exact number of sessions depended on the individual subject's ability. Each session was limited to no more than 200 pairs, and used either depth rotations, picture plane rotations, or an equal number of both types of rotation.

The results of this initial experiment showed that as the angular displacement increased, so did the time required to judge whether the two objects were the same or different. The mean reaction times increased from approximately one second for pairs in which the objects were identically oriented to 4.4 seconds for pairs in which the objects differed in orientation by 180°. This data tended to suggest that subjects compared objects by mentally rotating them at a rate that traversed 180° in an average of 3.4 seconds. This rate is calculated from the 4.4 seconds required to make determinations at 180° difference minus 1.0 seconds to make determinations for identical (non-rotated) objects. From this a rotation rate of approximately 53° per second was implied. Polynomial regression lines were computed for each of the subjects for each type of rotation. In every case a highly significant linear component was present. No significant quadratic or higher order effects were found. Another observation made by Shepard was that there were no significant differences between rotations in depth and rotations in the picture plane.

The treatment of isomorphic pairs was different from that of the same pairs, since the angle through which they are rotated to congruence is not defined. In general it was found that the response time for isomorphic pairs was approximately a second longer than the response time for the corresponding same pair. Subjects indicated that they rotated one end of one of the objects into congruence with the corresponding end of the other object, and then compared the free ends. If the free ends did not match the objects were deemed isomorphic.

Other methods for discriminating between objects, such as developing a coded representation for each of the figures and then comparing the codes, were dismissed by Shepard. Such methods would not depend on the orientation of an object, and thus should display a constant response time.

The description to this point has focused on the initial work done by Shepard and Metzler. In order to gather further evidence that the process of mental rotation was analogous to the corresponding physical operation, Shepard continued his study. A brief description of more recent work, done by Shepard and Cooper, follows.

In order to show the correspondence between mental rotation and physical rotation it needed to be demonstrated that the internal process passed through states that corresponded to the intermediate orientations of the physical process. Towards this end a second series of experiments was devised. These tests were single-stimuli experiments, and used polygons instead of three-dimensional objects.

Before this second set of tests could begin, it had to be established that the single-stimuli test caused the subjects to imagine the same progression of mental rotations as the paired drawings had. The subjects were first required to differentiate between a polygon and its mirror image, for each of eight polygons. Once this learning phase had been completed the actual test began. During each trial the subject was shown one of the polygons in an orientation that either matched the training position, or differed from it by some increment of 60°. The subject was asked to determine as quickly as possible whether the polygon was the standard version or a mirror image. As with the two-stimulus test, a linearly increasing relationship was found between angular displacement and response time. Once again subjects showed a consistently higher response time when mirror images were displayed. Of interest here is the fact that the angular sweep time for two-dimensional polygons was approximately 450° per second, while the three-dimensional figures used in the initial study were only rotated at a rate of 53° per second.

The next experiment in this series was designed to establish that subjects actually imagined the stimulus turning until it matched the orientation of the learned shape. During these sessions subjects were asked to imagine the rotation of a learned polygon. They were then presented with another polygon for comparison. Initially a vector was displayed indicating the orientation that was to be imagined. The subject then mentally rotated the specified test polygon to the desired orientation. Once this task was completed the subject indicated readiness and a polygon, which was either a standard image or a mirror image, was displayed. The subject was asked to decide whether the polygon was a standard or mirror image. Two times were recorded for each trial, the first was the set up time (the time that was required to perform the mental rotation), and the other was the response time (the time required to determine whether the polygon was a standard or mirror image). The preparation time once again exhibited a linearly increasing relationship between angular displacement and time. The response time was consistently fast, and averaged less than 0.5 seconds for all angular differences. These results once again supported Shepard's initial conjecture about the nature of the mental rotation process.

The final set of trials done by Shepard and Cooper was designed to show that the process of mental rotation passes through stages corresponding to the intermediate angles of physical rotation. Shepard speculated that if such a relationship did exist, the orientation that would elicit the fastest response would change steadily and progressively with time, and in step with the actual mental rotation process itself. In order to show that this was the case subjects were asked to imagine a specified polygon rotating at the rate that was natural for that particular subject. In should be noted that the same subjects were used throughout this second series of experiments, so there was extensive information about each individual's rate of mental rotation. At random intervals this shape or its mirror image were displayed. The subject was asked to identify the object as standard or mirror as quickly as possible. In half of the trials the stimuli was shown in a position to match the subject's presumed current

mental image; these were called "probe-expected" trials. In the other half the stimulus was displayed at varying angles from the subject's current mental image, these trials were called "probe-unexpected".

In half of the probe-expected trials the orientations were increments of 60°. These orientations were the same as the those used in the earlier single-stimulus experiment. In the other half of the probe-expected trials objects were shown at odd multiplies of 30°. These orientations were unfamiliar to the subjects. If Shepard's conjecture that mental rotation progresses smoothly through all angles rather than in jumps is true, then the results from these unfamiliar trials should be approximately the same as those from the familiar trials. Analysis of the test data showed that the results were almost identical.

In the probe-expected trials the subject's answers were uniformly fast, with a mean response time of approximately 0.5 seconds. In the other trials the response time increased linearly as a function of the angular displacement. This suggested that the mental process passes through intermediate angles in a continuous manner as does the physical process.

All of this empirical data seems to support Shepard's initial hypothesis. However, a number of questions remained. For example, how much physical detail do the mental images preserve as they are transformed? What is the neurophysiological basis of the mental images? How are the internal manipulations performed? The next section will describe the goals of the current research being done in the Psychology Department at the University of Waterloo. Readers who are interested in a more detailed description of Shepard's work are referred to Shepard and Metzler (1971) for a description of the initial study, and to a Scientific American survey article by Shepard and Cooper (1984) for a description of both sets of experiments. The description of Shepard's work given here is largely based on the latter article, it, however, provides more detail.

2.2. Current Research

Researchers in the Department of Psychology at the University of Waterloo have been interested in investigating the process of mental rotation for some time. Work in this area has been done by Bryden and Tapley (1977). Their study was designed to determine if there were any significant sex differences in the ability to perform mental rotation. The results of this study confirmed Shepard's initial findings about the linearity of the relationship between reaction time and angular displacement, although there were also significant quadratic and cubic components to the rotation effect. The study also indicated that men were superior at the mental rotation task, both in terms of speed and accuracy.

The current work being done by Bryden is aimed at designing an experiment that will test a subjects' ability to perform mental rotation on a number of different stimuli. The first set of trials are using shaded objects composed of seven blocks. Two types of seven-block stimuli are used, one is the Shepard and Metzler seven-block object. It has three right angle

twists. The other is a seven-block object with only two right angle twists. Two ten-block objects are being used as well. Again there is a Shepard and Metzler three-twist figure and a two-twist figure. For the final second set of trials, each of the objects will be displayed as outlined (wire-frame) drawings. Those figures more closely correspond to the original stimuli used by Shepard and Metzler.

The use of this broader range of stimuli will allow comparisons based on a number of physical features of these objects. The study will be primarily concerned with comparing the time taken to respond to large objects and small objects (ten-block objects versus seven-block objects), simple objects and complex objects (two-twist objects versus three twist-objects), and shaded objects versus outlined (wire-frame) objects. The comparison between shaded objects and outlined objects is of particular interest to computer graphics users, since it may indicate which representation is easier to understand. Information of this nature might be used to determine which representation is more suitable for particular applications.

The study performed by Shepard and Metzler was done with a select group of subjects, chosen from among the Stanford University undergraduate population, and screened for high spatial ability. They were trained specifically for the tasks that they were expected to perform. A goal of the current study is to investigate the mental rotation process in people who have not been specifically selected and trained to perform this task. A further difference between this study and Shepard's initial study is that Shepard used only a small number of subjects and a large number of trials. This study will use a larger number of subjects and a smaller number of trials.

The final goal of this study is to again investigate possible sex differences in the ability to perform mental rotation, and to see if any differences that are present become more pronounced with different sets of stimuli.

2.2.1. The Mental Rotation Experiment

As in Shepard's original study, subjects are shown pairs of three-dimensional objects and are asked to decide as rapidly as possible whether the two objects are the same or different. Objects are considered to be the same if they can be rotated into congruence.

This study is comprised of two major sections of four phases each. The major division is between shaded and outlined (wire-frame) objects. Each of the four phases corresponds to one of the test objects. A phase is further subdivided into three viewing rounds, each of which consists of 104 pairs of objects.

The subjects in this experiment are undergraduate students from the University of Waterloo. Since one of the objectives of this study is to see how average people perform the mental rotation task, no prescreening is done on the basis of spatial ability. The only requirement is that subjects be right handed. The reason for this stipulation is that the subject's use the second and third fingers of their right hand to press the buttons that give

their response to the current stimuli. Once this has been assured, the subject is given the Vandenberg and Kuse mental rotation test. Subject's are given fifteen minutes to complete this test. The test is given to obtain an independent measure of how good a particular subject's spatial abilities are.

A set of 104 slides showing pairs of objects is used as the basic stimulus set for the experiment. A set consists of 52 "same" slides, (slides in which the two objects can be rotated into congruence with each other) and another 52 slides showing two isomorphic objects. The isomorphic pairs differ by a reflection as well as a rotation, and therefore the two objects cannot be rotated into congruence with each other. Two duplicate sets of the 104 slides are used for each phase of the experiment. Each set of slides is randomly arranged so that there is no discernible pattern that might aid the subject in making determinations. One of the slide sets is used for the first and third viewing rounds, the other set is used for the second viewing round. The three viewing rounds are separated by rest breaks. Subjects generally take about two hours to view all 312 slides.

Subjects were seated at a table 90 inches from the screen. The slide projector was placed 114 inches from the screen, and was located to the left behind the subject. The response box was placed on the table, and the subject was allowed to position it so that it was comfortable. Before starting the experiment subjects were given eight practice trials. During a trial a subject is presented with one slide at a time, the flash of a slide being displayed triggers a photocell, which starts an electronic timer. The electronic timer measures the subject's response time to millisecond accuracy. The subject has two buttons, one to indicate that the objects are the same, the other to indicate that they are different. Depression of either button stops the timer and signals the experimenter of the subject's choice. The experimenter records the subject's response and time and then proceeds to the next slide.

The preliminary results of this first experiment tend to confirm the results of Shepard's initial study. Discussion of the preliminary results from this experiment is given in Chapter Five.

3. The Hidden Surface Algorithm

The hidden surface package described in this section is based on the scanline algorithm for hidden surface removal developed by Watkins at the University of Utah. The primary reason for choosing a scanline algorithm to solve the hidden surface problem is that it facilitated the implementation of an anti-aliasing capability. It was necessary to anti-alias the images being produced for the Psychology Department because it was thought that rastering phenomena might affect the results of the experiment. Slide 3 gives an example of the aliasing effects that are present in images rendered using a z-buffer. Before discussing the implementation of the current package, a description of Watkins's original algorithm will be given.

3.1. Watkins's Algorithm

Scanline techniques generate an image on a line-by-line basis. This type of algorithm differs from other rendering algorithms in that it processes sections of the screen at one time, rather than the individual objects that make up a scene. Another difference is that Watkins's algorithm solves the hidden surface problem at screen resolution. These differences allow for greater efficiency since scanline coherence can be exploited.

Scanline coherence is a property of most raster images. *Scanline coherence* means that adjacent scanlines appear very similar. This allows one to reduce the computation required to display a particular scanline to an incremental calculation based on the preceding scanline.

Decisions about visibility are also simplified. Hidden surface calculations can be made in two-dimensional space rather than in three-dimensional space, since the intersection of a polygon with a scanline is a line segment. Thus, the scanline algorithm can make depth decisions in the x-z plane. This reduces the hidden surface problem to deciding, for each scanline, which segments or portions of segments should be displayed. The remainder of this section describes Watkins's algorithm as given in Newman and Sproull (1979).

To solve the hidden surface problem Watkins's original implementation used four basic data types. These were polygons, edges, segments, and spans. A scene is described by planar polygons. Polygons are, in turn, described by their edges, and edges are described by their end points. Each edge contains pointers to the two polygons that it separates. If an edge is part of only one polygon then one of these pointer fields is empty. A *segment* describes a section of a polygon sliced along a particular scanline, and is itself described by its left and right edges. This edge description consists of values for XLEFT, XRIGHT, ZLEFT, and ZRIGHT. XLEFT and XRIGHT give the x-coordinates of the left and right edges of a segment, ZLEFT and ZRIGHT give the depth of each of these edges. The YLEFT and YRIGHT fields are used to count the number of scanlines until the left or right edge, respectively, terminates. In addition to edge information, segments contain shading information. Depending on the user's preference, Gouraud or constant shading can be used.

Gouraud shading, shades a polygon by linearly interpolating intensity along each edge and then between edges along each scanline. *Constant shading* calculates a single intensity value for shading an entire polygon. The final data type, the *span*, is used to described a visible portion of a scanline.

Table 1 – POLYGON data type	
Field	Meaning
SEGLIST	Points to a list of segments associated with this polygon.
CHANGING	Used to link together all polygons that are changing on a given scanline.
SHADING	Intensity levels may be stored here if constant shading is used.

Along with vertex and polygon descriptions, each edge also has an ENTERLIST field. This field is used to create a linked list of edges that enter on a particular scanline. For each scanline there is a YENTER list, this list contains all of the edges that enter on that scanline. These edges are linked together using the ENTERLIST field of each of the edges.

Table 2 – EDGE data type	
Field	Meaning
POLY1	A pointer to one of the polygons that the edge bounds.
POLY2	A pointer to the other polygon that the edge bounds. (may be NULL)
VERTEX1	A pointer to one of the edge's endpoints.
VERTEX2	A pointer to the edge's other endpoint.
ENTERLIST	Used to link together all edges that enter on a scanline.

Table 3 – SEGMENT data type	
Field	Meaning
POLYGON	A pointer to the polygon to which this segment belongs.
POLYSEGMENT	Used to link together all of the segments that belong to a polygon.
XLEFT	X-coordinate of the left edge.
ZLEFT	Z-coordinate of the left edge.
XRIGHT	X-coordinate of the right edge.
ZRIGHT	Z-coordinate of the right edge.
DXLEFT	Increment added to the x-coordinate of the left edge.
DZLEFT	Increment added to the z-coordinate of the left edge.
DXRIGHT	Increment added to the x-coordinate of the right edge.
DZRIGHT	Increment added to the z-coordinate of the right edge.
YLEFT	The number of scanlines remaining before the left edge exits.
YRIGHT	The number of scanlines remaining before the right edge exits.
SHADELEFT	Shading for the left edge. (optional: For Gouraud shading.)
SHADERIGHT	Shading for the right edge. (optional: For Gouraud shading.)
DSHADELEFT	Shading increment for the left edge. (optional: For Gouraud shading.)
DSHADERIGHT	Shading increment for the right edge. (optional: For Gouraud shading.)

Each segment has a POLYGON field and a POLYSEGMENT field. The POLYGON field is used to point to the polygon of which this segment is a part. The POLYSEGMENT field is used to maintain a linked list of all segments for a particular polygon. The POLYSEGMENT list is sorted on the increasing value of XLEFT for each of the segments belonging to a particular polygon. It should be noted that segments are not part of the original scene description, and are derived from the edge descriptions contained in the YENTER list for each scanline.

The primary data structure used in Watkins's implementation of the scanline algorithm is the XSORT list. The XSORT list links all active segments together in sorted order of the increasing value of XLEFT. On each scanline two major tasks are performed. The first task to update the XSORT list. The second task is the solution of the hidden surface problem for that scanline.

The first of these tasks is conveniently divided into four steps. The first step is to incrementally update the XSORT list. This list will consist of the segments that were considered on the previous scanline. For each segment the values of XLEFT, ZLEFT, XRIGHT, ZRIGHT, must be computed. This is done by adding DXLEFT, DZLEFT, DXRIGHT, DZRIGHT to XLEFT, ZLEFT, XRIGHT, ZRIGHT. YLEFT and YRIGHT must also be incremented on each scanline. If YLEFT, or YRIGHT equals zero after incrementing, then that edge is terminating. When this happens that particular edge's polygon is marked as "changing". The CHANGING field is used to create a linked list of all polygons that are changing on this scanline. If Gouraud shading is being used, the shading information must also be updated. This is done by adding DSHADELEFT and DSHADERIGHT to SHADELEFT and SHADERIGHT.

The second step is to process the YENTER list for this scanline. The YENTER list contains all the edges that enter on this scanline. Processing at this stage involves the creation of new segment blocks, if necessary, and the insertion of these blocks into the segment list for the appropriate polygon. All polygons that have entering edges are marked as changing and added to the CHANGING list.

The third step is to process the changing polygons. All of these polygons will have been marked in steps one or two. This processing involves the filling of any holes that may have been left in the segment list by actions that were taken in the first two steps. Holes will be formed where there are incomplete segment blocks. Incomplete segment blocks are blocks that are missing a left edge, a right edge, or both edges.

The final step is to sort all segments, thus creating a new XSORT list. This new list consists of the segments retained from the previous scanline and the new ones that were added on this scanline.

Once the preliminary work to update the XSORT list has been done, the hidden surface problem can be solved. The scanline is divided into spans, and the hidden surface problem is solved for each of these individual spans. Two routines are used to solve the hidden surface problem for a span, the LOOKER and the THINKER.

The hidden surface problem for spans can be divided into four categories. The first category consists of those spans in which there is only one segment. This segment will be visible throughout the span. The second type of span is one which contains a "spanner" that obscures all other segments in the span. A *spanner* is a segment that extends to or beyond the edges of a sample span. The third category consists of spans which contain a simple intersection. If only two segments fall inside a span then there may be a simple intersection. In this case, the intersection point, DIV, can be computed. One segment is visible in the region $SPAN_{left} \leq X_s \leq DIV$, the other is visible in the region $DIV \leq X_s \leq SPAN_{right}$. The forth type of span is a complicated span. Spans are categorized as complicated when they do not fall into one of the first three classes. A complicated span is handled by dividing it and then testing the smaller spans that are created as a result. Subdivision occurs at the leftmost segment endpoint within the span. If there are no segment endpoints within the span, division occurs at the midpoint of the span. The reason for choosing to divide at the leftmost endpoint is that it hastens the resolution process, since successful spans tend to start and end at segment boundaries.

The LOOKER processes each segment that lies wholly or partially within the current sample span, by retrieving the information necessary for the THINKER to decide which category that span belongs to. If the THINKER decides that the current span is a simple case, (one of the first three categories), it generates the information necessary to display that span. If the thinker detects a complicated case (the fourth type of span), it fails and subdivision occurs. The XSORT list serves as the source of spans for the THINKER. In addition to the XSORT list two other lists are maintained. The SEGOUT list contains segments whose right edge lies within the span currently being considered. Spans to the right of the current span do not need to know about these segments. The SEGACT list contains segments whose right edge extends beyond the right limit of the current span. These segments should be considered by subsequent spans on this scanline. If processing of a span succeeds, the segments in the SEGOUT list do not need to be considered again on this scanline. The segments in SEGACT are retained for the processing of subsequent spans. If processing of the current span fails, then all segments considered in the original span must be considered in the subspans. Therefore, the SEGOUT list is merged with the SEGACT list. The LOOKER, THINKER, subdivision of spans, and segment bookkeeping, are managed by an overall CONTROLLER.

This implementation of the scanline algorithm does not take advantage of scanline coherence. To use scanline coherence a list of $SPAN_{right}$ values (sample points) are used to divide a scanline into spans. The successful subdivisions on one scanline are used as the

sample points on the next scanline. The purpose of sample points is to provide the THINKER with a high number of spans that are easy to evaluate.

3.2. Complexity of Watkins's Algorithm

Watkins's algorithm has been widely used since it was first described, Watkins (1970). However, analysis of the complexity of the algorithm has proven to be difficult. Some of this difficulty has arisen from an inability to find a precise formulation for the complexity of this algorithm. A term that is often used to describe the running time of Watkins's algorithm is "visual complexity". According to Newman and Sproull (1973) Watkins's scanline algorithm is quite fast; they claim that the computation grows roughly as the visible complexity increases. However, no precise relationship is given.

Beatty, Booth, and Matthies (1981) made the following observations about the operations that are performed in Watkins's algorithm. Work proportional to the number of edges in a scene is being performed during the x and y bucket sorts at the start of each frame, and on various scanlines as each edge enters and leaves the scene. Initialization of the scene requires work proportional to the number of polygons that describe that scene. Incrementing the x and z values for each edge on each scanline requires work proportional to the number of edges×scanlines. The work needed to maintain the edges in an x-sorted order is proportional to the number of edge crossing. Finally, work proportional to the linear screen resolution is required to initialize the bucket sorts, and to perform the outer loop of the top to bottom scan of the scene. Actual pixel setting is proportional to the area (number of pixels) on the screen. Since this is independent of the algorithm, it is ignored. These observations can be used to give some insight on how the efficiency of this algorithm can be improved.

From these measures of work Beatty, Booth, and Matthies proposed that the *visual complexity* of a scene is the sum of the number of edges in the scene, the number of edges×scanlines, and the number of visible edge crossings. This definition was chosen because any algorithm that solves the hidden surface problem must examine every edge at least once. Since these edges must be incrementally updated on each scanline, a term proportional to the number of edges×scanlines is included. The final term arises from the observation that each visible edge crossing requires output.

Beatty, Booth, and Matthies have further suggested that the running time of Watkins's algorithm is not proportional to the visible complexity. The reason for this is that the complexity of Watkins's algorithm is proportional to the total number of edge crossings, not just the visible edge crossings. This is due to the fact that the active edge list is kept in x-sorted order, even though only the visible edges need to be ordered. This ordering is maintained by using a bubble sort, which does work proportional to the total number of inversions in the data (each edge crossing is an inversion). For n data objects a bubble sort

has a worst case running time of n^2. It is this that leads to the quadratic behavior of Watkins's algorithm.

Based on these observations the following improvements were suggested. First, scanline coherence is not only exhibited by adjacent scanlines, but also by whole groups of scanlines. Therefore, the algorithm need only be concerned about scanlines on which "critical events" occur. *Critical events* are the visible entry, exit, or crossing of edges in the scene. For scanlines on which no critical events occur it is sufficient to incrementally update the spans that were visible on the preceding scanline. The next scanline on which a critical event occurs can be easily predicted. Since the span list must be traversed on every scanline in order to generate display information, future edge crossings and exits can be predicted while this information is being checked. The next scanline on which an edge enters can be determined by examing the entering edge list. Since the crossing of invisible edges does not affect the visibility of the scene being rendered, those crossings can be ignored. A final observation is that when a critical event is known to occur on a given scanline between two edges, the processing need only occur in that specific region.

These improvements increase the average efficiency of Watkins's algorithm, however, the running time is still not proportional to the visual complexity of the scene being displayed. This is easily seen by observing that an invisible edge that crosses many visible edges will still cause a number of scanlines to be unnecessarily processed. The current version of Watkins's algorithm must perform a depth comparison when an invisible edge crosses a visible edge even though there may be no change in the visibility of the image. Even with these improvements quadratic behavior seems possible. More complex data structures which maintain z-sort information may be a way of avoiding this quadratic behavior for scenes whose visual complexity is not quadratic in the number of objects.

3.3. Current Implementation

This section describes the implementation of Watkins's algorithm used to produce the images that were used as stimuli in the experiment. The implementation is largely based on the improvements to Watkins's original scanline algorithm that were suggested by Beatty, Booth, and Matthies (1981). It is designed to be integrated into an existing graphics package, therefore a brief description of the existing software will be given before going on to the details of the new implementation.

The Computer Graphics Laboratory currently uses the GR graphics package. This system is a CORE-like package that has evolved, over a period of seven years, from a set of programming assignments that are given in the introductory course on computer graphics. This package is now the model used for teaching the introductory graphics course. Students are expected to implement their own version of the GR package in stages that correspond to the assignments in the course. The package was initially written in FORTRAN, and was

rewritten in Pascal when it replaced FORTRAN as the principle programming language at Waterloo. This change allowed the addition of some sophisticated data structures. In addition to the improvement to the capabilities of the package, it was found that student productivity also benefited from the change of languages. The final stage in this evolutionary process was a move from a Honeywell 66/60 running GCOS-8 to a VAX 11/780 running Berkeley UNIX. The research graphics package used by the Computer Graphics Laboratory is conceptually very similar to the Pascal teaching version. The major differences are that the research package is implemented in the C language and it is capable of interacting with a larger number of input and output devices.

Design of the GR package is based on modularity, and device independence. The package consists of a five-level hierarchy. Levels one to four form the basic graphics package, level five is used for application routines. This type of organization is ideally suited to incremental implementation, as is done in the assignments for the introductory graphics course.

The following is a brief description of the functions performed at each level of the package. Level one is the bottom level, it contains the drivers for the supported input and output devices. Calls to this level are in the form of device independent display requests. The virtual device has 15 bits (0 – 32767) of resolution in each of x, y, and z. The primary purpose of this level is to translate the "virtual coordinates" for output into the appropriate device specific coordinate system, and to then issue the proper commands to perform the desired operations. This process is reversed for input. Input coordinates are translated from the device specific coordinate system to the virtual coordinate system.

Level two performs the viewport transformation. A viewport is established by calling the routine Viewl2. If no viewport is specified, a default viewport is provided. This level maps floating point "clipping coordinates" to the fixed point virtual coordinates that are passed to level one.

The primary function of level three is to perform clipping. Clipping is the process of removing those portions of the scene that do not appear within the specified viewing volume. The package is capable of performing both line and polygon clipping. Level three accepts as its input points in a "homogeneous coordinate system". The points are clipped by level three, and passed to level two. The output from level three is in clipping coordinates in the ranges $-1.0 \leq x,y \leq +1.0$, and $0.0 \leq z \leq +1.0$.

Level four is the user interface. At this level the user has primitives for drawing points, lines, polygons, and text. The user is also provided with facilities for the specification of colour, light sources, and geometric transformations. A text facility similar to the GSPC CORE implementation is provided. Input to level four is in non-homogeneous three dimensional "world coordinates". Level four maps an arbitrary window in world coordinates to a 2×2×1 window centered on the origin. Input coordinates are transformed

to homogeneous coordinates to allow for a uniform matrix representation of the perspective and translation transformations.

Level five consists of application routines. These routines are primarily used for creating and traversing data structures which are used to describe scenes.

The current implementation of Watkins's algorithm is tailored to replace portions of the level one output section. Input to level one is in the form of polygons which have been processed by the higher level routines. These polygons all lie within the bounds of the virtual device's screen coordinates. Given this input there are two major tasks to be performed. The first of these is to create the appropriate data structures to allow for the solution of the hidden surface problem. The second task is to render the scene using these data structures and to then display it on the appropriate output device. The next section describes the data types and structures used in this package. This will be followed by a detailed description of the actual implementation of the scanline algorithm.

There are three main data types: polygons, edges, and spans. A polygon node is created for each polygon that is passed down from the upper levels. Level one calculates the coefficients of each polygon's planar equation. These coefficients are used when it is necessary to calculate the depth of a polygon. This depth information is required to solve the hidden surface problem. In addition to the planar coefficients, each polygon node contains shading information. The current implementation supports constant shading only. Each polygon node contains a pointer, OPEN, to any span belonging to this polygon that is currently active. This pointer is used to simplify the retrieval of spans from the depth sort structure that is maintained while visible spans are being calculated. Polygon nodes are stored in a polygon structure which is an array of buckets of pointers to polygon nodes. When a polygon node is created it is given a unique number that corresponds to the next free bucket in the polygon array. This number is also given to all of the edges that belong to that particular polygon. When one of these edges needs to reference some information about its polygon, this number can be used to address the appropriate polygon node within the polygon structure.

The second data type is the edge. When a polygon is passed to level one the edges associated with that polygon are maintained in a doubly linked ring. This ring is traversed, and the edges are unlinked into chains of edges which are in turn placed in an edge structure. An old chain is ended and a new one started when there is a change in the vertical direction of the edges in the ring. Thus a chain consists of edges that continue from one to the next such that each edge's starting y-coordinate is greater than the starting y-coordinate of the next edge. A polygon node's pointer into its linked ring of edges points to the edge with the highest starting y-coordinate. Given the direction of traversal of the ring, one can tell whether a given edge is a left or right edge of that particular polygon. The edge structure is an array of buckets of pointers to lists of chains. The number of buckets corresponds to the

Table 4 – POLYGON data type	
Field	Meaning
RED	Red component of the polygon colour.
GREEN	Green component of the polygon colour.
BLUE	Blue component of the polygon colour.
A,B,C,D	The coefficients of the polygon's planar equation.
OPEN	A pointer to any open spans belong to this polygon. (used in determining visible spans)

number of scanlines. For each scanline the lists are ordered by increasing value of the x-coordinate of the first edge of each edge chain. A chain is placed in a particular bucket of the edge structure according to the starting y-coordinate of the first edge of the chain. If there is already a non-null list of chains at the bucket where a new chain is being placed, the new chain is inserted into the list according to the value of the starting x-coordinate of the first edge in that chain. In cases where two chains have the same value for their starting x-coordinate, the slope of the first edge in each chain is used to determine the correct placement. If the bucket in which a new chain is being placed has a null list of chains, the pointer in that bucket is made to point directly to the new chain.

Each edge contains coordinate information. This information consists of the starting and ending y-coordinate, the starting x-coordinate, and an x-increment. The x-increment is calculated from the slope and is the change in the x-coordinate per scanline. The starting x-coordinate is updated on each scanline with the x-increment, to give the x-coordinate for that edge on the next scanline. In addition to coordinate information, each edge contains a polygon number to indicate which polygon contains this edge's depth and shading information. Two flags LORR and VIS are used to record whether the edge is a left or right edge, and whether it is currently visible. The LINTER and RINTER fields are used to tell the next scanline on which an edge crossing involving this edge occurs. Finally, there are a number of pointer fields. The NEDGE field is used to link edges together into chains of edges. The PREV and NEXT fields are used to create doubly linked lists of edges. These fields can be used in two contexts, to link together lists of visible edges, or to link together lists of invisible edges. The final two pointer fields are LSPAN and RSPAN. LSPAN points to the span that has this edge as a left bounding edge. RSPAN points to the span that has

Table 5 – EDGE data type	
Field	**Meaning**
POLY	Number of the polygon to which this edge belongs.
X	Current x-coordinate.
INCX	Increment to be added to X before processing the next scanline.
LINTER	Next scanline on which this edge intersects a visible edge to its left.
RINTER	Next scanline on which this edge intersects a visible edge to its right.
NEDGE	A pointer to the next edge in this edge chain.
NEXT	Used to create doubly linked lists of edges.
PREV	Used to create doubly linked lists of edges.
LSPAN	A pointer to the span that has this edge as a left bounding edge.
RSPAN	A pointer to the span that has this edge as a right bounding edge.
LORR	Indicates whether the edge is a left or right edge.
VIS	Indicates whether the edge is visible or invisible.
LASTY	Gives the scanline on which this edge exits.

this edge as a right bounding edge. If an edge is invisible then both of these fields are null.

The final data type is the span. A span corresponds to a visible section of a polygon. Each span has left and right visible edges that bound the span. The EDGES field is used to point to a doubly linked list of edges that are obscured by this span. The PREV and NEXT fields are used to create doubly linked lists of spans. The following fields contain information that is used to exploit scanline coherence. LINTER and RINTER give the next scanline on which one of the obscured edges crosses the left or right bounding edge. The EXIT field gives the next scanline on which an edge associated with this span exits. Finally,

each span contains the number of the polygon that is visible within its bounds. This number allows a span to access shading and depth information.

Table 6 – SPAN data type

Field	Meaning
POLY	Number of the polygon that is visible within this span.
LEFT	Pointer to the left bounding edge.
RIGHT	Pointer to the right bounding edge.
LINTER	Next scanline on which an edge intersects a the left bounding edge.
RINTER	Next scanline on which an invisible edge intersects a the right bounding edge.
EXIT	The next scanline on which an edge associated with this span exits
EDGES	Pointer to the invisible edges associated with this span.
NEXT	Used to create doubly linked lists of spans.
PREV	Used to create doubly linked lists of spans.

Spans and edges are used to create the visible span structure. This is the main data structure used in solving the hidden surface problem. The visible span structure is a doubly linked list of the spans that are visible on the current scanline. This structure is updated as edges enter the scene, exit the scene, or cross. Each of the spans in this structure has a left and right bounding edge. These edges are linked together throughout the entire structure to form a doubly linked list of the visible edges which are maintained in x-sorted order.

The processing done to the visible span structure on a given scanline depends on whether a critical event occurs on that scanline. As mentioned previously, the next scanline on which a critical event occurs is easily predicted using the coherence information that is contained in the individual spans.

For scanlines on which there are no critical events a minimal amount of processing needs to be done. As the visible span structure is traversed the visible edges are incrementally updated, and display information is retrieved. It should be noted that the use of this type of structure allows a saving since invisible edges are only updated when it becomes necessary to do so.

Processing for scanlines on which a critical event occurs is more complicated. Using the coherence information it is possible to tell which critical events have occurred. Processing for the three types of critical event occurs in the following order. First, the exiting edges are deleted using the routine Remove. Next, any crossing edges are processed by the routine Crossover. Finally, if there are any entering edges they are placed in the visible span structure by the routine AddEdges. If one of the types of critical events does not occur on a given scanline, that phase is omitted. After a critical event has occurred the coherence information must be updated. The next section gives a description of the routines that are used to solve the hidden surface problem. The first of these routines, Resolve, is used to calculate visible spans.

3.3.1. Calculation of Visible Spans

Resolve is used to set up the initial visible span structure, and is called by other routines to calculate the spans that are visible in any gaps that may be formed by crossing or exiting edges. As input, this routine accepts a doubly linked list of edges in x-sorted order, and a list of polygons that are active at the current position on the scanline. The list of active polygons is used to set up the initial span information. Given this input, the routine then calculates the spans that are visible over the range defined by the edge list. If the edge list is null the routine will calculate the polygon (span) that is visible at the current position on the scanline.

To solve the hidden surface problem Resolve maintains two structures. The first is a visible span list. This list contains visible spans whose right bounding edge has already been encountered. The second structure, the currently active tree, contains spans whose right bounding edge has not yet been encountered. The elements of this structure are hierarchically arranged based on the depth information that is currently available. If a span is known to be hidden by a particular span, then it along with all the other spans obscured by that span are *children* of this span. If the depth relation between two spans is not known then these spans are "siblings". *Siblings* are spans that are on the same level of the hierarchy. By maintaining such a structure, one reduces the number of depth comparisons that must be made to find the closest span. When one is trying to find the closest span only those spans at the highest level of the hierarchy need to be considered. This allows one to immediately dismiss all of the children of these spans. This allows a saving in the number of computations that need to be made since a smaller set of spans need to be tested to find the closest span. By ignoring the children, until their level has become the top (visible level) of

the active span structure one saves further on the number of computations, since some of these spans may terminate before it becomes necessary to test this level of the hierarchy.

Resolve starts by checking the active polygon list that it received as input. For each element in this list a span node is created and placed in the active span structure. The first element of the active span structure is the closest and thus currently visible span. The second element is the next closest span, the rest of the spans are maintained in unsorted order. These unsorted spans become the children of the second closest span, and will all be siblings of each other.

Once the initial span information has been processed the input edge list is scanned. When a left edge is encountered a new span node is created for the polygon that it starts. The OPEN pointer in the polygon node associated with that edge is made to point to the newly created span. When the corresponding right closing edge is encountered this pointer is retrieved and the span can be deleted. The depth of the newly created span is compared with the depth of the currently visible span. If the new span is closer than the currently visible span, the currently visible span is given the left edge as a right bounding edge. This span is then removed from the active span structure and attached to the end of the visible span list. Since the span's visibility was terminated by a new closer span, it remains active as the second closest span. In order to keep the active span structure correct another span must be created with the same polygon information as the span that was just placed in the visible span list. This new span is the immediate child of the new visible span and represents the continuing section of the previously visible span. All of the children of the previously visible span are pushed down one level as a result. The closer span is placed at the top of the active span structure, and becomes the new currently visible span. The currently visible span will have the left edge as its left bounding edge. If the new span is deeper than the currently visible span, it becomes a sibling of the children at the level immediately below the currently visible span. When the currently active span structure is empty, the new span can be installed as the currently visible span without any depth comparisons.

When a right edge is encountered, one of the currently active spans must be terminating. If it is the currently visible span that is terminating, then this right edge is made the right bounding edge of the currently visible span. This span is removed from the currently active span structure and added to the end of the visible span list. The children at the level immediately below the currently visible polygon must now be checked to find the new currently visible span. To do this the depth of each of the sibling spans at this level must be checked. The closest of these spans is then moved from its current position to the top of the visible span structure. This check will also find the second closest span, it becomes the immediate child of the currently visible span, and the rest of the sibling spans become children of this span. If the right edge terminates one of the hidden spans then there is no change in the visibility of the scene. The terminating span is deleted from the structure and its immediate children are moved up to the level that it had previously occupied. This

must be done since it is possible that a child of the deleted span may be closer than the deleted span's siblings.

The hidden surface section of this routine terminates when all of the input edges have been processed. Before the routine can finish some work must be done to the visible span list. If the currently active span structure is non-null at this point, then the currently visible span must be added to the end of the visible span list. This may occur when Resolve is being used to fill gaps in the visible span structure, since the terminating edge of the last visible span may not be among the input edges. After the visible spans have been determined all of the input edges are still doubly linked. This must be changed so that the visible edges are connected in a doubly linked list, and the invisible edges are associated with the span that obscures them. This is easily accomplished, the edges that fall between the left and right bounding edges of a span are the edges that are obscured by that span. These edges can be unhooked from the doubly linked list and associated with the appropriate span by using the EDGES field. Once the invisible edges have been removed, the left and right bounding edges are reconnected. The final processing to be done involves the calculation of coherence information for each individual span.

3.3.2. Adding Entering Edges

The AddEdges routine is used to add entering edges to the visible span structure. AddEdges accepts a doubly linked list of x-sorted chains of edges. Since edges are linked together in chains, the edges of new polygons are guaranteed to enter on a given scanline in pairs. This makes the process of adding and removing edges much easier. The fundamental purpose of AddEdges is to compare the depths of the entering polygons with the depths of the currently visible polygons at points where the two overlap.

This routine starts by checking if there are any edges that fall to the left of the leftmost edge in the visible span structure. If there are any such edges, they are unlinked from the rest of the input edges and passed directly to Resolve so that the visible spans defined by the edges can be determined. The resulting visible spans are attached to the front of the visible span structure. When connecting the end of the new spans to the front of the visible span structure one must check the two spans at the connection point to see how they should be properly joined. If the right bounding edge of the last of the new spans is non-null then the spans and the visible edges can simply be connected. If the right edge of the last span is null, however, a depth comparison must be performed to see if this span or the first span of the visible span structure is closer. If the new span is closer it is flagged as continuing (this is denoted as the "open" condition), and the rest of the entering edges are processed. The visibility of this open span will be terminated in one of three ways, the right closing edge of this span will be encountered, a closer left edge will be encountered, or a currently visible span that is closer will be encountered. If the first span of the visible span structure is closer than the last of the new spans then the left bounding edge of this span becomes the right

bounding edge of the new span. In this case the spans and the visible edges can be reconnected. For the remainder of this section the term "current span" will refer to the span that is currently visible at this point in the visible span structure. The term "active span" will refer to any new span that is in the open condition.

The main loop in this routine scans over the visible span structure until the section of the structure is reached where the next entering edge is active. There are a number of checks that must be performed during this scanning process. When there is an active span the following checks must be performed. If the left bounding edge of the current span is a left edge, then this span may be closer than the active span. Thus a depth comparison must be performed at the x-coordinate of the left bounding edge of the current span. If the active span is closer it remains active and the left bounding edge of the current span becomes an invisible edge associated with the active span. This edge is added to a list of invisible edges that are obscured by the active span. Depending on where the active span terminates the rest of the edges belonging to the current span may also become invisible edges associated with the active span. If the current span is closer than the active span, then the active span terminates at the left bounding edge of the current span. The left bounding edge of the current span becomes the right bounding edge of the active span. The open condition is no longer true, so the span and visible edge lists can now be reconnected. The list of invisible edges associated with the active span are now attached to that span, and its coherence information is calculated. The active span's polygon number is added to a list of currently active polygons. This is done because the span's visibility was terminated by a closer span rather than by the right closing edge of that polygon. This active polygon list is maintained in order to keep track of the polygons that may become visible if any gaps exist in the visible span structure. The depth of these polygons must also be checked whenever a deeper polygon emerges from behind a closer polygon.

If there is no open span then the following checks must be performed. If the current span's left bounding edge is a right edge then a deeper polygon is emerging from behind a polygon that had previously obscured it. In this case one of the entering polygons may be closer than this emerging polygon. Thus the depth of the current span must be checked against the depth of the closest of the active entering polygons. If the current span is closer, no changes occur. However, if one of the new polygons is closer then it becomes active. This active span starts at the right bounding edge of the previous polygon, and will obscure at least part, and possibly all of the current span. As mentioned previously, some or all of the invisible edges associated with the current span may become associated with the active span.

If a new polygon does not become active because of the previous check, then another check is done to see if there is a gap between the current span and the next span. If there is no gap then nothing needs to be done. If there is a gap, then the closest of the active entering polygons will become visible in this region. A new active span must be created and

inserted after the current span. This active span will have the right bounding edge of the current span as its left bounding edge. If the previous two checks do not apply then no further processing needs to be done.

This scanning loop terminates when the span that the next entering edge overlaps is reached. If there is an active span at this point, the entering edge is processed with respect to this span. The visible span structure is ignored because it is being obscured by the active span at this point on the scanline. If there is no active span then the following situations may occur. If the entering edge falls in a gap between spans, the scanning loop terminates at the span that lies to the immediate right of the entering edge. If the next entering edge falls beyond the rightmost edge of the visible span structure, then the rest of the input edges can be passed directly to Resolve. In this case processing becomes much simpler. The visible spans returned by Resolve are appended to the visible span structure. The connection of these two sets of visible spans presents problems that are analogous to those encountered in attaching new spans to the front of the span structure. If the left edge of the first span of the new visible spans is non-null and there is no active polygon then the spans and edges need only be connected. If there is an active span, then the left edge of the first of the new spans becomes the right bounding edge of the active span. If the left edge of the first span of the new spans is null and there is no active span, then the right bounding edge of the last span of the span structure becomes the left bounding edge of the first span of the new spans. If there is an active span, then these two spans represent the same polygon and must be merged.

If the next entering edge falls within the visible span structure, the processing considers the current span. The nature of the scanning loop assures that the current entering edge will fall either in a gap to the left of the left bounding edge of the current span, or within the current span itself. With these stipulations the following cases may occur.

If there is no active span and the left bounding edge of the current span is a right edge, then a test must be made for emerging polygons. This test may cause the creation of an active span. If this happens the rest of the processing is the same as for an active span.

When the left bounding edge is a right edge the entering edge must fall within the bounds of the current span, since there can be no gap between the current span and the span to its immediate left. Once this check has been done, processing of the entering edge can start. If the entering edge is a left edge then a new span may be starting, depending on the depth of the entering polygon. If there is an active span then the new span's depth is compared with the depth of the active span. If the entering edge is deeper than this span it becomes one of the invisible edges associated with the active span. If the new edge is closer than the active span, then the active span's visibility is terminated with the new edge becoming its right bounding edge. A new active span is created with the entering edge as its left bounding edge. The polygon visible in this region is the one that the entering edge

starts. Care must be taken to ensure that the invisible edges associated with the current span are properly placed with either the old active span, the new active span, or the remaining portion of the current span.

If there is no active span then the depth of the entering edge is compared with the depth of the current span. In this case it is possible for the entering edge to fall in a gap to the left of the left bounding edge of the current span. If this is the case a new active span is started, with the entering edge as its left bounding edge. This span is attached to the preceding span, the remaining connections to be made later in the processing.

If the entering edge does not fall to the left of the current span, it must fall within the bounds of the current span. The depth of the entering edge is compared with the depth of the current span. If the entering edge is deeper, it becomes an invisible edge associated with the current span. If the entering edge is closer, then a new active span is started. The current span must be split into two sections. One section is for the part of the span that lies to the left of the new span. This section is bounded on the right by the entering edge. The other section is used to represent the rest of the current span, it will remain visible if the right closing edge of the entering polygon occurs before the right bounding edge of the current span. The new edge becomes the left bounding edge of the newly created active span. Once again care must be taken to ensure that the invisible edges associated with the current span are properly distributed among the new arrangement of spans.

If the entering edge is a right edge, then one of the entering polygons must be terminating. If there is an active span and the entering edge is the right closing edge of that span then its visibility is terminated. If the entering edge is not the right closing edge of the active span it becomes an invisible edge associated with the active span. In both cases the terminating polygon's identification number is removed from the active entering polygon list. When closing the active polygon and reattaching it to the rest of the visible span structure the following cases may occur. If the entering edge falls to the left of the current span's left bounding edge, then the visibility of the current span is not affected. The entering edge becomes the right bounding edge of the active span, and this span and the current span are connected. If the entering edge overlaps the current span then there may be a change in the visibility of this span depending on the depths of the two spans. If the current span is closer than the active span, the left bounding edge of the current span becomes the right bounding edge of the active span, and the right closing edge of the active span becomes an invisible edge associated with the current span. If the active span is closer than the current span, then a change occurs in the visibility of the current span. The entering edge becomes the right bounding edge of the active span, and the left bounding edge of the current span. The old left bounding edge of the current span becomes an invisible edge associated with the active span. Once again care must be taken to ensure that the invisible edges associated with the current span are properly reassigned to either the active span or the current span.

If there is no active span then the entering edge becomes an invisible edge associated with the current span. Since the entering edge is a right edge it must be invisible, or its polygon would be visible to its immediate right. As in the previous cases the closing edge's polygon identification number is removed from the active polygon list. The AddEdges routine terminates once all of the entering edges have been added to the visible span structure.

3.3.3. Crossing Edges

The second routine used for updating the visible span structure is called Crossover. It processes visible edge crossings and ensures that the span structure remains correct. The Crossover routine is based on the analysis of edges crossings done in Hamelin and Gear (1977). Tables 7 and 8 give a brief synopsis of the type of edge crossings that can occur, and the actions that should be taken in each case.

Table 7 – Analysis of Edge Crossings

2nd edge / 1st edge	LV	RV	LI	RI
LV	I1	E	C1	H
RV	C2	I2	NC	NC
LI	NC	H	NC	NC
RI	NC	C3	NC	NC

LV	Left visible edge.
RV	Right visible edge.
LI	Left invisible edge.
RI	Right invisible edge.

1st edge is the leftmost edge before the crossing.
2nd edge is the rightmost edge before the crossing.

Table 8 – Actions	
Occurrence	Action
E	Error; the edges belong to the same polygon
NC	No change in visibility
I1	First edge becomes invisible
I2	Second edge becomes invisible
C1	If the second edge is in front of the polygon visible to the left of the first edge, make the second edge visible
C2	Compare the depths of the polygons to which the edges belong. The edge with the forward polygon remains visible
C3	If the first edge is in front of the polygon visible to the right of the second edge, make the first edge visible
H	Hard crossing. Compare the depths of all polygons under the crossing to see which is visible

Crossover is invoked when the coherence information indicates that a crossing has occurred that involves a visible edge. Processing of the visible span structure proceeds from span to span until all of the spans have been checked. If changes are required they are made locally to the section of the visible span structure where the crossing has occurred.

Four checks are applied to each span. The first check is to see if the left bounding edge of the current span has crossed the preceding visible edge in the visible edge list. If this has occurred some action must be taken to correct the visible span structure. In general an edge will be tested against the edges to its left in the visible edge list until the edge to its immediate left has a smaller x-coordinate, or until it becomes the first edge in the visible edge list.

The second test checks to see if any of the invisible edges associated with the current span have crossed the left bounding edge. As in the previous case, if an edge crosses more than one edge it need only be compared against those edges that are visible. The third check tests if the right bounding edge has crossed any of the invisible edges associated with the

current span. The final check tests if the right bounding edge of the current span has crossed the left bounding edge.

Any crossings involving visible edges are passed to the next section of the routine. This section consists of two nested switch statements which implement the Hamelin and Gear edge crossing table. This section tries to adhere as closely as possible to the terminology used by Hamelin and Gear. First refers to the edge that was leftmost before the crossing, and second refers to the edge that was rightmost before the crossing.

Before any processing can be done the types of the crossing edges must be determined. This is done by checking the LORR and VIS fields. The LORR field tells whether an edge is a left or right edge. The VIS field indicates whether or not an edge is visible. The types of the first and second edges are then used to determine the particular type of crossing that has occurred.

The following sections give a detailed description of the processing done for each of the sixteen types of crossings. The format for each of the following subsections is "type of the second edge" crossing "type of the first edge".

3.3.3.1. Left Visible Edge Crosses Left Visible Edge

The 1^{st} edge becomes invisible. The following changes must be made to the visible span structure. If the 1^{st} edge is the right bounding edge of the preceding span, then the 2^{nd} edge becomes the new right bounding edge of that span. One must now check the preceding span to see if any of the invisible edges associated with that span now belong with the current span. Once these edges have been transferred the RINTER value for the preceding span can be calculated. In general when a left or right bounding edge is replaced by another edge, either LINTER or RINTER will have to be recalculated. From this point on, this computation will be assumed to be implicit in the bounding edge replacement process. Once the preceding span has been processed the span that was just obscured can processed. The left bounding edge and the invisible edges associated with this span are transferred to the obscuring span. The coherence information for the obscuring span can now be updated. Once this has been done the span and visible edge lists can be reconnected.

3.3.3.2. Left Visible Edge Crosses Right Visible Edge

This case requires a depth comparison. The closer of the two spans remains visible. The following changes must be made to the visible span structure. If the 2^{nd} edge (left visible) is closer than the 1^{st} edge (right visible), then the 2^{nd} edge becomes the right bounding edge of the preceding span. Once again the invisible edges of the preceding span must be checked to see if there are any that should be moved. Once this has been done the coherence information can be calculated and the span and visible edge lists reconnected. If the 1^{st} edge is closer, then the 1^{st} edge becomes the left bounding edge of the next span. If

the 2nd edge falls within the bounds of the 1st edge's span then it is added to the invisible edge list associated with this span. If the 2nd edge does not fall within the bounds of this span it must cross one or more additional visible edges. Evaluation of these crossings will determine the 2nd edge's final placement. The final step is to update the coherence information and reconnect the span and visible edge lists.

3.3.3.3. Left Visible Edge Crosses Left Invisible Edge

There is no change in the visibility of the scene. No depth comparisons need to be made since it is already known that the polygon that the 1st edge (left invisible) starts must be deeper than the polygon that the 2nd edge (left visible) starts. Therefore, LINTER and RINTER are calculated and the 1st edge becomes an invisible edge associated with the 2nd edge's span.

3.3.3.4. Left Visible Edge Crosses Right Invisible Edge

There is no change in the visibility of the scene. No depth comparisons need to be made since it is already known that the polygon that the 1st edge (left invisible) starts must be deeper than the polygon that the 2nd edge (left visible) starts. Therefore, LINTER and RINTER are calculated and the 1st edge becomes an invisible edge associated with the 2nd edge's span.

3.3.3.5. Right Visible Edge Crosses Left Visible Edge

This case cannot occur since both of these edges would have to belong to the same polygon.

3.3.3.6. Right Visible Edge Crosses Right Visible Edge

The 2nd edge becomes invisible. The 1st edge becomes the new left bounding edge of the span that the 2nd edge starts (if such a span exists). The visible edge list can now be reconnected and the coherence information updated. If the 2nd edge falls within the bounds of the 1st edge's span, it is placed in the invisible edge list associated with this span. If the 2nd edge does not fall within the bounds of this span then it must cross one or more additional visible edges. These crossing must be evaluated to determine the final placement of the 2nd edge.

3.3.3.7. Right Visible Edge Crosses Left Invisible Edge

If the 1st edge (left invisible) is the left starting edge of the polygon that is visible to the right of the 2nd edge (right visible), then a "hard crossing" has occurred. A *hard crossing* is a crossing where all polygons that are active under the crossing must be examined in order to determine which polygon or polygons are visible. This must be done because a gap is left

between the 1^{st} edge and the 2^{nd} edge. The 1^{st} edge becomes the left bounding edge of the span that is visible to the right of the 2^{nd} edge. The next step is to fill in the "visibility gap" that is left between the 1^{st} edge and the 2^{nd} edge. This is done by passing an active polygon list to Resolve, and placing the resulting visible spans in the gap. If the 1^{st} edge is not the left starting edge of the polygon that is visible to the right of the 2^{nd} edge, then the 1^{st} edge is added to the list of invisible edges that are obscured by this span.

3.3.3.8. Right Visible Edge Crosses Right Invisible Edge

If the 1^{st} edge (right invisible) is closer than the polygon that is visible to the right of the 2^{nd} edge (right visible), then the 1^{st} edge's polygon becomes visible. If there is no polygon visible to the right of the 2^{nd} edge, then the 1^{st} edge's polygon is known to be visible, without performing any depth comparisons. If the right bounding edge of this newly created span overlaps the left bounding edge of the next span in the visible span structure, then this will be resolved when this crossing is evaluated. If there is a polygon visible to the right of the 2^{nd} edge, then a depth comparison must be made between the currently visible polygon and the emerging polygon. If the emerging polygon is closer, then the 1^{st} edge becomes the left bounding edge of the currently visible polygon. A new span is created and inserted to indicate that the emerging polygon is now visible in the region that is bounded by the 1^{st} edge and the 2^{nd} edge. If the emerging polygon is deeper than the currently visible polygon, then the 1^{st} edge becomes an invisible edge associated with the currently visible span.

3.3.3.9. Left Invisible Edge Crosses Left Visible Edge

If the 2^{nd} edge (left invisible) is closer than the polygon that is visible to the left of the 1^{st} edge (left visible), then the emerging polygon is visible. There are two cases to be considered. If the span in which the 1^{st} edge is the left bounding edge is the first span in the span list then the emerging polygon is known to be visible without any depth comparisons, however, it must be ensured that this new span is designated as the new first span of the visible span structure. The second case occurs when there is a gap to the left of the 1^{st} edge. Once again the new span is known to be visible without any depth comparisons. Therefore, a new span is inserted in the span list immediately to the left of the span in which the 1^{st} edge is the left bounding edge. If the left bounding edge of this new span overlaps the span to its left, then this will be corrected when that visible edge crossing is evaluated. If there is a polygon visible to the left of the 1^{st} edge, a depth comparison must be performed. If the emerging polygon is closer than the currently visible polygon, the 2^{nd} edge becomes the right bounding edge of the preceding span. A new span is created and inserted between the 1^{st} edge and the 2^{nd} edge to indicate that the emerging polygon is now visible in this region. Care must be taken to ensure that the previously visible span's invisible edges are properly distributed between the remaining portion of that span and the emerging span. If the

emerging polygon is deeper than the currently visible polygon, and if the 2^{nd} edge falls within the bounds of the span to the right of the 1^{rt} edge, it becomes an invisible edge associated with that span. If this is not the case, the 2^{nd} edge must cross one or more additional visible edges. Analysis of these crossings will determine the final placement of the 2^{nd} edge.

3.3.3.10. Left Invisible Edge Crosses Right Visible Edge

There is no change in the visibility of the scene. If the 2^{nd} edge (left invisible) falls within the bounds of the span to the right of the 1^{rt} edge (right invisible), then the 2^{nd} edge becomes an invisible edge associated with this span. If the 2^{nd} edge does not fall within the bounds of this span, it must cross one or more additional visible edges. Analysis of these crossings will determine the final placement of the 2^{nd} edge.

3.3.3.11. Left Invisible Edge Crosses Left Invisible Edge

The crossing of invisible edges does not affect the visibility of the scene. Therefore this case is not tested.

3.3.3.12. Left Invisible Edge Crosses Right Invisible Edge

The crossing of invisible edges does not affect the visibility of the scene. Therefore this case is not tested.

3.3.3.13. Right Invisible Edge Crosses Left Visible Edge

If the 2^{nd} edge (right invisible) is the closing right edge of the polygon that is visible to the left of the 1^{rt} edge (left visible), then a hard crossing has occurred. Second becomes the right bounding edge of the span that lies immediately to the left of the current span, and a gap is left between the 1^{rt} edge and the 2^{nd} edge. This gap is filled by calling Resolve with a list of active spans, and a list of the edges from the preceding span that now fall in the newly created gap. The visible spans returned by Resolve are inserted in this gap. If the 2^{nd} edge is not the closing right edge of the polygon visible to the left of the 1^{rt} edge then this edge is obscured by the span visible to the left of the 1^{rt} edge. If the 2^{nd} edge falls within the bounds of this span it is added to the list of invisible edges associated with that span. If the 2^{nd} edge does not fall within the bounds of this span it must cross one or more additional edges. Analysis of these crossings will determine the final placement of this the 2^{nd} edge.

3.3.3.14. Right Invisible Edge Crosses Right Visible Edge

There is no change in the visibility of the scene. If the 2^{nd} edge (right invisible) falls within the bounds of the span to the left of the 1^{rt} edge (right visible), it is added to the list of invisible edge associated with this span. If the 2^{nd} edge does not fall within the bounds of

this span it must cross one or more additional visible edges. Analysis of these crossings will determine the final placement of the 2^{nd} edge.

3.3.3.15. Right Invisible Edge Crosses Left Invisible Edge

The crossing of invisible edges does not affect the visibility of the scene. Therefore this case is not tested.

3.3.3.16. Right Invisible Edge Crosses Right Invisible Edge

The crossing of invisible edges does not affect the visibility of the scene. Therefore this case is not tested.

Crossover terminates once all of the spans in the visible span structure have been evaluated, and if necessary updated.

3.3.4. Removing Edges

The final type of updating that needs to be done to the visible span structure involves the removal of exiting edges. There are two types of exiting edges. If there is another edge following the exiting edge in its chain, this new edge will enter the visible span structure at the exact point that the current edge is exiting. This new edge can replace the exiting edge at its current position in the visible span structure. Exiting edges of this type are denoted as "continuing" edges. If the current edge is the last edge in its edge chain then the polygon that this edge belongs to is terminating. Exiting edges of this type are denoted as "terminating" edges. The use of chains guarantees that terminating edges will only occur in pairs. Remove processes both continuing and terminating edges.

Remove is called when the coherence information indicates that an edge or edges are exiting. If the removal of any edge causes gaps to be formed in the visible span list, Resolve is called to generate a new set of visible spans (if any are visible) for that region. The following discussion gives a detailed description of the processing done by Remove.

Remove examines the visible span structure in a span by span fashion. A distinction is made between exiting edges that continue and exiting edges that terminate. The former can be replaced by the next edge in its chain. If the exiting edge is invisible when it exits only the coherence information for the newly entering edge needs to be calculated. If the exiting edge is visible either LINTER or RINTER will have to be recalculated for the entire span. The processing for terminating edges is more complex and is outlined in the rest of this section.

The first section of the Remove routine determines the status of the current span. There are five types of spans. The first type of span has no change in visibility. The fact that the visibility does not change does not necessarily mean that no changes have occurred in

this span. Any invisible edges that have exited will not affect the visibility of this span, however, the coherence information will have been updated. Left terminating edges must be remembered until their closing right edge has been encountered, since the visibility of other spans may be affected by the termination of these polygons. The reason for this is that it is possible for a span's visibility to terminate without either of its bounding edges terminating. This type of span will have a right visible edge as its left bounding edge and a left visible edge as its right bounding edge.

The second class consists of those spans in which the left bounding edge terminates. It is guaranteed that the left bounding edge of these spans will be a left visible edge. If the left bounding edge had been a right visible edge then its polygon would have been visible to its immediate left. Thus there would have to have been a span with this edge as a right bounding edge. This span will precede the current span and will also have to be processed since its right bounding edge is terminating. Therefore, any right edges will be processed before the current span is encountered. When the left bounding edge is removed a gap is formed in the visible span structure. This gap is filled by calling Resolve with a list of the currently active polygons, and the edges that were obscured by the exiting span. Resolve will return the spans that are visible in this section. The process used to insert the new spans into the gap is essentially the same as the one that was outlined previously, with one additional case. This case occurs when the last span before the gap in the visible span structure has no right bounding edge, and the first span of the newly created spans has no left bounding edge. These two incomplete spans represent the same polygon, and must be merged. The same situation arises when the last span of the newly created spans has no right bounding edge, and the first span of the remainder of the visible span structure has no left bounding edge. This case is also solved by merging the two spans.

Spans with only the right bounding edge exiting make up the third category. If the right bounding edge is a left visible edge, nothing is done. Processing for this edge is left for the next span, since the exiting polygon is actually represented by the subsequent span. If the right bounding edge is a right edge then this span's polygon is exiting. First the terminating edge is removed. The resulting gap is filled by calling Resolve with a list of the currently active polygons, and the edges that were obscured by the exiting span. The new spans are inserted by the previously outlined procedure.

The fourth type of exiting span is one in which both bounding edges terminate. This case is very similar to the two previous cases. Once again the left bounding edge is guaranteed to be a left visible edge. If the right bounding edge is a left visible edge then this edge is left for processing with the next span, since it is this span that represents its polygon. The left bounding edge is removed, if the right bounding edge is a right visible edge it is also removed. Resolve is then called to determine the spans that are visible in the gap that is formed by the exiting polygon. These spans are inserted in the visible span structure by the previously outlined procedure.

The final type of exiting span is one in which the polygon represented by the span terminates but the two bounding edges do not. The gap left here is once again filled by calling Resolve.

The final section of this routine performs the insertion of new spans into any gaps that have been formed in the visible span structure. The procedure used to do this has already been outlined. There are, however, two special cases. When the first span of the new spans becomes the first span of the visible span structure, care must be taken to ensure that the pointer to the visible span structure is properly updated. The second of these special cases occurs when Resolve returns a null list. Resolve will return a null list when there is nothing visible in the region that has analyzed. In this case the spans that are on the left and right sides of the gap must be connected. The cases that arise in this situation are similar to those that occur when inserting a list of new spans into the visible span structure.

Remove terminates once all of the spans in the visible span structure have been checked.

4. The Slide Sets

A total of nine different sets of 104 slides each were produced. Two copies of each of the sets were made. One of these nine sets used a four-block object as the stimulus, this set was used primarily for calibration. The other eight sets consisted of four objects, each rendered in shaded and outlined (wire-frame) versions. Two of the objects are composed of ten blocks, and two are composed of seven blocks. One of these ten-block objects has two right angle twists and the other has three right angle twists. There are also two and three twist seven-block objects. The right angle twists on each object are strategically located so that an object cannot be rotated into congruence with itself. This stipulation is necessary so that there is only one axis through which an object can be rotated into congruence with similar objects that are in different orientations. For each type of object there are two specific objects, which are the enantiomorphic versions of that object. *Enantiomorphic objects* are isomers that differ only in handedness. Therefore, they are related to each other as a object is related to its mirror image object. Slides 4 to 12 show the nine objects that were used for the experiment. On each of these slides the lefthand figure depicts a standard object, the righthand figure depicts a mirror object.

Each of these objects was rotated about the x-axis to produce a set of unique orientations. The set of orientations used for this experiment corresponded to the 24 increments of 15° that lie between 0° and 360°. These orientations were used to construct pairs of objects that differed in angular displacement by each of the thirteen unique 15° increments between 0° and 180°. Each slide set contains eight pairs that differ by each of these 15° increments. Four of these slides depict same objects. Two of these examples depict two similar standard objects, and the other two examples depict two similar mirror objects. The other four pairs depict differing objects. Two of these different pairs have the standard object on the right and the other two pairs have the standard object on the left. Table 9 gives the specific orientations that were used to create each pairing. This table can be divided in half, with each half consisting of 26 pairs. Pairs in the left half were used to create slides that had two standard objects. These pairings were than re-used with a mirror object on the left and a standard object on the right. These two groups total 52 slides. The pairs in the right half of the table were used to create slides with two mirror image objects, and then re-used to create slides with a standard object on the left and a mirror object on the right. These two groups also total 52 slides, and together with the previous two groups make up the overall total of 104 slides.

The standard orientation for an unrotated object is +22.5° in each of the three primary axes. The angular displacement of a given orientation from the standard orientation can be calculated as follows, (orientation number − 1) × 15°.

Table 9 – Pairs of Orientations								
Displacement	left	right	left	right	left	right	left	right
0°	4	4	22	22	7	7	12	12
15°	9	10	15	16	5	6	1	2
30°	11	13	18	20	1	3	19	21
45°	5	8	11	14	19	22	23	2
60°	3	7	8	12	13	17	20	24
75°	1	6	13	18	21	2	23	4
90°	3	9	17	23	10	16	18	24
105°	10	17	22	5	8	15	14	21
120°	1	9	4	12	11	19	16	24
135°	4	13	17	2	14	23	6	15
150°	5	15	8	18	10	20	21	7
165°	3	14	9	20	11	22	16	2
180°	1	13	7	19	18	6	24	12

In addition to the rotation about the x-axis, each pair of objects may also have been rotated about the z-axis. This rotation is used to adjust the orientations of the two objects so that the results of the experiment would not be affected by poor views. Any adjustment in the z-axis was applied equally to each object so that the angular displacement between the two objects remained the same.

The purpose of introducing this second rotation is to eliminate three artifacts that might cause ambiguous pictures. The first of these artifacts is the total occlusion of more distant parts of an object by nearer parts of the objects. This occurrence can give the impression

that a particular object is composed of fewer blocks than it actually is. The second problem with some of the orientations was that one or both of the objects appeared two-dimensional. Objects tended to appear two-dimensional when one of the principle axes of the objects was in a line with the sight vector. It is desirable to have all of the views easily identifiable as three-dimensional objects for two reasons. First, the sudden appearance of an object that is visibly different from the majority of objects tends to confuse subjects, and thus affect their reaction time. Second, the rate at which human beings can rotate two-dimensional objects seems to be significantly higher than the rate at which they can rotate three-dimensional objects. It was not known what affect the two-dimensional appearance of these objects might have on the rotation rate so these views were eliminated. The final criterion used in the adjustment process was only used for the shaded objects. It was found that the presence of two adjacent surfaces that were very similarly shaded made the object much harder to discern. As two surfaces become more closely shaded the internal edge that separates them becomes fainter, and thus it becomes harder to determine where one surface ends and the other begins.

Each of these artifacts can be eliminated by rotation in one of the other principle axes. In the case of occlusion, the hidden part of the object can be rotated back into view. In the case of a two-dimensional appearing object, it can be rotated out of alignment with the sight vector. Finally, in the case of poor shading, the object can be rotated so that different lighting is cast on the two similarly shaded surfaces.

Only the shaded slides were previewed. The reason for this was that the line drawings were rendered in exactly the same orientations as the shaded drawings. Thus, all of the occlusions and two-dimensional views had already been eliminated, and shading was no longer a criterion. Of the images that were previewed, about half needed some adjustment. These adjustments ranged from 5° to 180°. This adjustment process was done using the z-buffer device of the GR graphics package. The reason for using the z-buffer for previewing was that it took less time to display an image than the method that was used to render the production images. Since these images were only being previewed and adjusted, picture quality was not important.

The images used in the production of the slides were rendered using the GR graphics package and the additional software described in Chapter Three. The test objects were specified using the Read_Scene scene description language. Read_Scene reads a file describing an instance of a directed acyclic graph (DAG), builds the corresponding DAG, and returns a pointer to that structure. A scene file contains a one line description for each node in the DAG. The node description lines are indexed, starting at 1, and incrementing with every successive node. Each node is identified by a single character field, an is followed by a number of numeric fields that specify parameters. In addition to these fields are pointers to siblings and children. These pointers consist of the index of the node that is the sibling or the child. The pointers are used to link the individual node descriptions into a

scene.

The Read_Scene language has the following capabilities.

rotation of objects
translation of objects
scaling of objects
text objects
polygon objects
colouring of objects
Bezier curves and surfaces
B-spline curves and surfaces
Beta-spline curves and surfaces

There are many shortcomings in the Read_Scene language. The worst of these is the difficulty in adding or removing nodes from a scene file once it has been created. This requires the renumbering of all the node indices. Since the stimuli used in this experiment where so simple this did not prove to be a problem. However, if more complex stimuli were desired the Fécit scene description language could be used. Fécit is a hierarchical language for describing scenes that was developed by Sylvia Lea, Lea(1983).

A specific stimuli is created by the following method. First a unit cube centered on the origin is described. This cube is replicated and translated to form the desired arm-like structure. This much will describe the geometry of the object. Addition scaling and x, y, z rotation nodes are added so that the object can be scaled to the desired size, and rotated into any orientation. Rotation and scaling are accomplished by changing the value specified in the scaling or rotation node. This can be done under program control once the structure has been created by Read_Scene and passed to the renderer. Each object used in the experiment was given an initial rotation of 22.5° in each of the three principle axes, and an initial scaling of 0.90. Two scene files are used for each type of object, one describes the standard object and the other describes the mirror image object. Read_Scene provides a mirroring facility. However, it mirrors an object about the origin, and this application only required mirroring about the x-axis. Thus, the need for two scene files. The following is an example of a Read_Scene file, which describes the Shepard and Metzler seven-block object.

```
rotation of objects
* points - nodes root

8 0 20 20

* Vertices of a cube
1 -0.5 -0.5 -0.5 0 0 0
2 -0.5 -0.5 +0.5 0 0 0
3 -0.5 +0.5 +0.5 0 0 0
4 +0.5 +0.5 +0.5 0 0 0
5 +0.5 -0.5 +0.5 0 0 0
6 +0.5 -0.5 -0.5 0 0 0
7 +0.5 +0.5 -0.5 0 0 0
8 -0.5 +0.5 -0.5 0 0 0
* Faces of a cube
1 p 00 02 4   2 3 4 5
2 p 00 03 4   6 7 8 1
3 p 00 04 4   1 8 3 2
4 p 00 05 4   5 4 7 6
5 p 00 06 4   1 2 5 6
6 p 00 00 4   8 7 4 3
* Scale all cubes
7 s 01 00 +0.90 +0.90 +0.90
* Position individual cubes
8 t 07 09  1.0  1.0  3.0
9 t 07 10  1.0  2.0  3.0
10 t 07 11  1.0  2.0  2.0
11 t 07 12  1.0  2.0  1.0
12 t 07 13  2.0  2.0  1.0
13 t 07 14  2.0  3.0  1.0
14 t 07 00  2.0  4.0  1.0
* Orient reference figure
15 t 08 00  -1.5 -2.5 -2.0
16 z 15 00  +22.5
17 y 16 00  +22.5
18 x 17 00  +22.5
19 t 18 00  +0.0 +0.0 -3.75
* Scale and translate entire picture
* to fit, putting in two copies of the
* object, one rotated, the other not.
```

20 s 19 00 +1.0 +1.0 +1.0

A slide rendering program was written to read in scene description files and to perform the appropriate rotations in order to display a given pair of objects in the desired orientations. Each of these images consisted of two objects, each displayed in its own viewport. The use of two viewports means that when two identically oriented objects are displayed, identical images will appear in each of the viewports. If there was only one viewport, the perspective transformation would make the objects appear as if they had been slightly rotated with respect to each other. The previous arrangement was chosen, because it was thought that this apparent rotation might affect the results of the study.

A number of experimental parameters were considered in the production of the slides. These included projector distance, subject distance, projection width, viewport width, and object radius. The projector distance is the distance, measured in inches, from the lens of the projector to the viewing screen. The subject distance is the distance, in inches, from the subject to the viewing screen. The projection width is the width, in inches, of the image displayed on the screen when it is projected from the projection distance. The viewport width is the width of each of the viewports measured in virtual coordinates. The object radius is the radius of the largest test object measured in block widths. All of these parameters, except for the object radius, are used to ensure that the displayed objects appear to be the proper size for the viewing distance. The object radius is used to make sure that clipping of the test stimuli does not occur, regardless of the rotations employed. The values these parameters had for this experiment are given in Table 10.

Table 10 – Rendering Parameters

Parameter	Value
Projection distance	114.0 inches
Projection width	25.5 inches
Subject distance	90.0 inches
Viewport width	13720 virtual units
Object radius	3.75 block units

It is important to know the relationship between the block size and the screen width. Using this information the displayed objects can be appropriately scaled so that each of the test objects will fit within the viewport. The blocks used in creating the experimental objects were scaled so that the screen width is 17.2 when measured in block widths. This ratio means that each of the viewports is 7.6 block units wide. Therefore, any object with a radius of less than 3.8 block units can be displayed without clipping. This block size along with the object radius guarantees that the two objects depicted on each slide will be separated by at least 2.1 block units.

The final consideration in the creation of the stimuli was the inter-block spacing. A unit block is deemed to have 100% spacing. A block that has 80% spacing is one that is composed of edges that are 80% of the linear of size of a unit block. Thus 100% spacing would correspond to no inter-block spacing. As the volume of each block decreases the inter-block spacing increases. For the slides, 90% spacing was chosen.

Each slide has an identification code plotted in the lower right hand corner. This code is used to tell which orientations are represented on a given slide. The identification code consists of two groups of alphanumeric characters. The first group of characters is used to designate the set that a given slide belongs to. The second group consists of a three digit number in the range 001−104, which indicates the particular orientations that are depicted on the slide. An adhesive label with this identification number is also produced. This label is attached to the slide frame to make identification easier.

When a slide is produced, its identification number and descriptive information are automatically written to a file. This descriptive information uniquely identifies each slide. These files are used to create reference sheets for each slide set. A reference entry for a particular slide contains the following information.

slide identification code
class of the object depicted
orientation of the left object
type of the left object
orientation of the right object
type of the right object
y-rotation
z-rotation

As mentioned before the slide identification code is the series of alpha-numeric characters that appears in the lower right hand corner of each slide, and on the adhesive label for that slide. The class of the object tells which object is being displayed. For example, 7(3) refers to the object composed of seven blocks, which has three right angle twists. The type of the object tells whether the depicted object is a standard or mirror image. The orientation is a number from 1−24 that gives the angular displacement of the object as

described in Table 9. The y-rotation gives the amount of the corrective rotation, in degrees, about the y-axis. The z-rotation gives the amount of corrective rotation about the z-axis. In each case the amount of corrective rotation about the y-axis is zero, since it was found that rotation about the z-axis was sufficient to eliminate all anomalies.

The production of all of the slide sets was done in the following manner. Images were first rendered on an Adage/Ikonas RDS-3000 framebuffer. These images were then photographed using a Dunn 631 color film recorder. Kodak 64 ASA bulk loaded film was used for all of the slides.

5. Conclusion

At this time the experiment using the preliminary slide set (the four-block object) has been completed and analyzed. The experiment using the set of seven-block objects with three twists has also been conducted, but data analysis is not yet complete.

5.1. Results

The following is a brief summary of the results obtained from this preliminary experiment. Subjects' responses were sorted primarily according to the angular displacement between the two depicted objects. Secondary sorting was done according to same/different pairs, and correct/incorrect answers. Only correct responses were analyzed. Accuracy was found to be quite high, about 90%. This compares favorably with Shepard's study where accuracy was approximately 96%.

The response times for same pairs ranged from about one second for identical objects to between four and five seconds for same pairs that differed by 180°. The response times for isomorphic pairs were found to be uniformly fast, falling in the four to five second range. The rate of mental rotation measured in these initial tests is very similar to the figure measured by Shepard in his original study.

When the response times for each subject for the same objects were plotted it was found that the slopes were primarily linear. For nine of the twelve subjects used in the experiment the slopes were exclusively linear. The other three subjects displayed some quadratic behavior. At approximately 120° the response times leveled off, and in some cases decreased slightly.

The results from the four-block experiment tend to confirm the results of Shepard's original study. There were, however, some differences. The largest discrepancy between the current results and the original results occurs in the time required to recognize that two images do not depict the same object. In Shepard's study it was found that the time needed to recognize that two pairs were different was about a second longer than the time required to recognize the cooresponding same pair. Initial results from the current study indicate that subjects take a uniform amount of time to recognize isomorphic objects, and that this time is about the same as the time required to recognize two orientations that differ by 180°.

Bryden has conjectured that this result might be due to the fact that a subject tries to do more to prove that two objects are the same, Bryden(1985). For example, if two identically oriented same objects are displayed, it is clearly evident that they are the same. However, if two identically oriented different objects are displayed it is not readily apparent that they are not the same. Thus the subject will mentally rotate the object until they have aligned some corresponding part of both objects. At this point it will become apparent that the remainder of the object does not match, therefore the objects must be different.

The other difference between Shepard's results and the current results was the presence of a quadratic component in some of the subjects' responses. As mentioned earlier these results were obtained using a four-block object as the stimuli. This object was not asymmetric, so there existed two possible rotations that would bring one orientation into congruence with another orientation. This led Bryden to speculate that some of the subjects were using the second rotation when the angular displacement of the first rotation became large. If subjects were actually using this second rotation it may account for the quadratic component present in their behavior.

Although analysis of the data from the first of the seven-block objects is not yet complete, some preliminary results have been obtained. Before the experiment was conducted it was speculated that larger objects would take longer to mentally rotate. This hypothesis was based on the belief that the larger objects would require the encoding and processing of more information. However, the results showed that the response times for the seven-block object were faster than the response times for the four-block object. This result, at first, seems counterintuitive. It now seems that the additional information provided by a larger object makes the mental rotation process easier.

Despite the differences mentioned above, the majority of the results support Shepard's original conjectures about the nature of mental rotation.

5.2. Further Work

The results of Shepard's original mental rotation experiment were first published in 1971. In the time since then this experiment has been performed using a variety of media. The original study was done using cards. The stimuli for this study were generated by A. M. Noll using a Stromberg-Carlson 4020 microfilm recorder. Bryden's 1977 study was done using wooden blocks. The current study is being done using photographic slides. A natural progression in this work would be to perform the experiment on line using interactive computer graphics displays.

Using the equipment and software presently available in the Computer Graphics Laboratory this new experiment could be readily performed. The main concern would be the time required to display the stimuli. The least amount of time between the presentation of two stimuli is one second. This will occur when two identical objects are displayed. This is not enough time for the images to be computed while the subject's decision is being made, since the average time needed to render a pair of these objects is about 30 seconds. However, if the objects are precomputed and run-length encoded, there is enough time to display them.

The Adage/Ikonas RDS-3000 has 32 bits of colour information, and a crossbar switch which enables one to select specific bit planes or groups of bit planes for display. Using this capability it is possible to display one image while another is being read in. Once the subject

has responded, the image that has just been read in can be flashed onto the monitor by changing the crossbar setting to display the bit planes that hold this new image. The process then repeats itself, as another image is read into another set of bit planes while the subject is responding to the stimulus that is currently being displayed.

This experiment would require the development of real-time data acquisition software, to measure and record the subject's replies. This software would relieve the experimenter of this duty, and would thus be more reliable. The final component of this experiment would be a user interface. This software allows the subject to respond to or interact with the stimulus. User interfaces are an area of active research within the Computer Graphics Laboratory. An experiment in human colour perception has already been performed by a member of the laboratory, Schwarz (1985), and as a result much of the data acquisition and interaction software is already available.

By moving to interactive computer graphics equipment a number of improvements can be made to the experiment. The study performed by the Psychology department used only right handed subjects. This constraint was necessary because of limitations in the data acquisition equipment. The subject uses the second and third fingers of their right hand to respond to the stimuli. Left handed subjects could be tested if the meaning of these buttons could be reversed. The use of interactive computer graphics equipment would allow the meaning of responses to be altered depending on the individual subject's characteristics. The meaning of a particular response can easily be changed under software control by specifying subject traits before the experiment. These traits can then be examined by the data acquisition software to determine what the meaning of particular responses should be. By doing this any right handed or left handed biases can be eliminated. By using graphics equipment a greater variety of stimuli can be used, for example, in the probe expected/unexpected trials a greater number of angles could be tested (rather than only multiples of 30°). Another advantage provided by the flexibility of this equipment is the ability to dynamically order the stimuli. In the probe expected/unexpected trials this would allow the investigator to vary the the times around the subject's expected time.

A long range goal of the Computer Graphics Laboratory is to create a number of tools that will allow experimenters to describe and run their own experiments. Such tools would include the ability to describe suitable user interfaces, the type of data to be recorded, and the type of stimuli to be used. By performing Shepard's mental rotation experiment on computer graphics equipment a number of lessons may be learned that will help to guide the development of these longer-term projects. Of more immediate significance would be information indicating how well graphics equipment compares with the other media used for this experiment. The first step in this direction is to reliably duplicate earlier results.

The main purpose of this project was to try to find novel applications for computer graphics. Towards this end a large set of stimuli was provided for one set of experiments. The success of this endevour has already spawned further interest in the use of computer graphics for psychology experiments. Current work is being done to create stimuli which will be used to test the fidelity with which features of a physical stimulus are retained when mentally rotated. Further work may be done on a variation of the single-stimulus experiment described in Chapter Two. Hopefully, the cumulative result of these collaborative efforts with the Psychology Department will be a complete experiment specification system that runs entirely on interactive computer graphics equipment.

6. References

Bryden M. P., Tapely M. S., An Investigation of Sex Difference in Spatial Ability: Mental Rotation of Three Dimensional Objects, Canadian Journal of Psychology, 1977, 31(3), pp. 122-130.

Bryden M. P., Personal Communications, 1985.

Beatty John C., Booth Kellogg S., Matthies Larry H., Revisiting Watkins Algorithm, Proceedings of the Canadian Man-Computer Communications Society 7th Conference, 1981, pp. 359-369.

Foley James D., Van Dam Andries, Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Company, 1982.

Gauroud H., "Continuous Shading of Curved Surfaces", IEEE Transactions on Computer Graphics, C-20(6), June 1971, pp. 623-628.

Hamelin G., Gear C. W., Raster-Scan Hidden Surface Algorithm Techniques, Computer Graphics 11(2), Summer 1977, pp.206-212.

Lea, Sylvia C., Fécit - A Structured Language for Describing Computer Generated Scenes (Masters Thesis), University of Waterloo, 1983.

Matthies Larry H., Scanning Algorithms for Computer Graphics (Masters Essay), University of Waterloo, 1981.

Newman William M., Sproull Robert F., Principles of Interactive Computer Graphics, 1st ed., McGraw-Hill Book Co., 1979.

Shepard Roger N., Metzler Jacqueline, Mental Rotation of Three-Dimensional Objects, Science,1971, 171, pp. 701-703.

Shepard Roger N., Cooper Lynn A., Turning Something Over in the Mind, Scientific American, Dec. 1984, pp. 359-369.

Schwarz M. W., An Empirical Evaluation of Interactive Colour Selection Techniques (Masters Thesis), University of Waterloo, 1985.

Watkins G. S., A Real-Time Visible Surface Algorithm, University of Utah,

Computer Science Dept., UTEC-CSc-70-101, June 1970, NTIS AD-762 004.

7. Appendix 1

Slide 1 l – (shaded) reg 10(3) r – (shaded) reg 10(3)	Slide 2 l – (shaded) reg 10(3) r – (shaded) mir 10(3)	Slide 3 l – (z-buffer) reg 10(3) r – (z-buffer) reg 10(3)
Slide 4 l – (shaded) reg 4(2) r – (shaded) mir 4(2)	Slide 5 l – (shaded) reg 7(2) r – (shaded) mir 7(2)	Slide 6 l – (outline) reg 7(2) r – (outline) mir 7(2)
Slide 7 l – (shaded) reg 7(3) r – (shaded) mir 7(3)	Slide 8 l – (outlined) reg 7(3) r – (outlined) mir 7(3)	Slide 9 l – (shaded) reg 10(2) r – (shaded) mir 10(2)
Slide 10 l – (outlined) reg 10(2) r – (outlined) mir 10(2)	Slide 11 l – (shaded) reg 10(3) r – (shaded) mir 10(3)	Slide 12 l – (outlined) reg 10(3) r – (outlined) mir 10(3)