INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
MacDougal, Paul Duncan

GENERATION AND MANAGEMENT OF OBJECT DESCRIPTION HIERARCHIES
FOR THE SIMPLIFICATION OF IMAGE GENERATION

The Ohio State University

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106

Copyright 1984 by
MacDougal, Paul Duncan
All Rights Reserved
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark √.

1. Glossy photographs or pages ✓
2. Colored illustrations, paper or print ✓
3. Photographs with dark background ✓
4. Illustrations are poor copy
5. Pages with black marks, not original copy
6. Print shows through as there is text on both sides of page
7. Indistinct, broken or small print on several pages
8. Print exceeds margin requirements
9. Tightly bound copy with print lost in spine
10. Computer printout pages with indistinct print
11. Page(s) ________ lacking when material received, and not available from school or author.
12. Page(s) ________ seem to be missing in numbering only as text follows.
13. Two pages numbered _________. Text follows.
14. Curling and wrinkled pages
15. Other

University Microfilms International
GENERATION AND MANAGEMENT OF OBJECT DESCRIPTION
HIERARCHIES FOR THE SIMPLIFICATION OF IMAGE GENERATION

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Paul D. MacDougall, B.S., M.S.

*****

The Ohio State University
1984

Reading Committee:
Dr. Franklin C. Crow
Dr. Edwin R. Tripp III
Prof. Charles A. Csuri

Approved By

Adviser
Department of Computer and Information Science
Acknowledgements

I would like to thank Frank Crow, Chuck Csuri, and Ed Tripp for their guidance and review during my research. I would also like to thank all of the staff and students at CGRG whose support and encouragement made this all possible. I learned a great deal from the Art Education students who bring such a different perspective to Computer Graphics. I would also like to thank my wife, Mary, for the love and support she has given me during this long process.
Vita

September 28, 1956 ... Born - Schenectady, New York

1978 ................. B.S. Botany, University of North Carolina, Chapel Hill, North Carolina

1978-1980 ............ Biochemist, Research Triangle Institute, Research Triangle Park, North Carolina

1980-1981 ............ Graduate Teaching Associate, Department of Computer and Information Science, Ohio State University, Columbus, Ohio

1981 .................. M.S. Computer and Information Science, Ohio State University, Columbus, Ohio

1982-1984 ............ Graduate Research Associate, Department of Computer and Information Science, Ohio State University, Columbus, Ohio

1983-1984 ............ Graduate Administrative Associate, Department of Computer and Information Science, Ohio State University, Columbus, Ohio

PUBLICATIONS

## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>ii</td>
</tr>
<tr>
<td>Vita</td>
<td>iii</td>
</tr>
<tr>
<td>Table of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>Chapter 1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Management of Detail</td>
<td>2</td>
</tr>
<tr>
<td>1.2. Historical Perspective</td>
<td>4</td>
</tr>
<tr>
<td>1.3. Description Hierarchies</td>
<td>7</td>
</tr>
<tr>
<td>1.4. Overview</td>
<td>8</td>
</tr>
<tr>
<td>Chapter 2. Coordinating the Transition Between Descriptions</td>
<td>9</td>
</tr>
<tr>
<td>2.1. Instantaneous Transition</td>
<td>10</td>
</tr>
<tr>
<td>2.2. Blended Transitions</td>
<td>11</td>
</tr>
<tr>
<td>2.3. Additional Problems</td>
<td>20</td>
</tr>
<tr>
<td>Chapter 3. Display of the Transition</td>
<td>22</td>
</tr>
<tr>
<td>3.1. Using Transparency</td>
<td>22</td>
</tr>
<tr>
<td>3.2. Using Pixel Coverage</td>
<td>28</td>
</tr>
<tr>
<td>Chapter 4. Staging the Transition</td>
<td>31</td>
</tr>
<tr>
<td>4.1. Choosing transition type</td>
<td>32</td>
</tr>
<tr>
<td>4.2. Choosing appropriate transition distance</td>
<td>35</td>
</tr>
<tr>
<td>Chapter 5. Generation of Object Descriptions</td>
<td>37</td>
</tr>
<tr>
<td>5.1. Object Types</td>
<td>37</td>
</tr>
<tr>
<td>5.2. Individual Description Generation</td>
<td>38</td>
</tr>
<tr>
<td>5.3. Description Modification</td>
<td>41</td>
</tr>
<tr>
<td>5.4. Automatic Description Generation</td>
<td>43</td>
</tr>
<tr>
<td>5.5. Comparison of Techniques</td>
<td>52</td>
</tr>
<tr>
<td>Chapter 6. Animation Environment</td>
<td>55</td>
</tr>
<tr>
<td>6.1. Data Generation and Manipulation</td>
<td>55</td>
</tr>
<tr>
<td>6.2. Scene Description</td>
<td>56</td>
</tr>
<tr>
<td>6.3. Viewing and Rendering</td>
<td>56</td>
</tr>
<tr>
<td>6.4. Post-processing</td>
<td>57</td>
</tr>
<tr>
<td>6.5. Modifications to the Pipeline</td>
<td>57</td>
</tr>
<tr>
<td>Chapter 7. Results and Conclusions</td>
<td>70</td>
</tr>
<tr>
<td>7.1. Further Research</td>
<td>74</td>
</tr>
<tr>
<td>List of References</td>
<td>75</td>
</tr>
<tr>
<td>Appendix A: INSPECT User's Manual</td>
<td>78</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>A.1 Introduction</td>
<td>78</td>
</tr>
<tr>
<td>A.2 Subsystems in INSPECT</td>
<td>79</td>
</tr>
<tr>
<td>A.3 Inspect&gt; Subsystem Commands</td>
<td>80</td>
</tr>
<tr>
<td>A.4 Colormod&gt; Subsystem Commands</td>
<td>86</td>
</tr>
<tr>
<td>A.5 Long Commands</td>
<td>90</td>
</tr>
<tr>
<td>A.6 Edit&gt;</td>
<td>93</td>
</tr>
<tr>
<td>A.7 Knob&gt;</td>
<td>96</td>
</tr>
<tr>
<td>A.8 Expression Syntax</td>
<td>98</td>
</tr>
<tr>
<td>A.9 Conditional Phrases</td>
<td>98</td>
</tr>
<tr>
<td>A.10 Assignment Phrases</td>
<td>100</td>
</tr>
</tbody>
</table>
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Three Descriptions of a Sphere</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Sample Blending Functions</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Linear Blending Functions</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Net Transmittance of Blending Functions</td>
<td>24</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Combination of Instantaneous and Linear Blending Functions</td>
<td>25</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Overlapping Linear Blending Functions</td>
<td>28</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Hierarchy With Each Level Generated Independently</td>
<td>42</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Simplified Plane Curves Using Equal Arc Length</td>
<td>46</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Simplified Plane Curves Using Equal Segment Count</td>
<td>47</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Simplified Plane Curves Using Smallest Edge First</td>
<td>48</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Hierarchy Generated Using 3-D Grid Constraint</td>
<td>51</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Hierarchy Generated Using Small Edge Deletion</td>
<td>53</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Edge Length Graph of a Three Level Description Hierarchy</td>
<td>66</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Edge Count in Docking Simulation</td>
<td>71</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Still Frames From Docking Simulation</td>
<td>73</td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction

One of the current goals in computer graphics research is the generation of more realistic imagery. New lighting models and texture mapping have been used to get a less plastic appearance. One thing which has gained little attention is the overall complexity of images. While fractalized mountains make nice backgrounds, the middle ground is usually barren. A constant density of objects (trees, rocks, buildings, ...) over the entire visible space would result in much more realistic imagery. In general, this would result in a much greater screen space concentration of objects along the horizon. If all objects were displayed using a very detailed description, most of the computation would be done displaying objects which are far away, and quite possibly occluded. But, if objects were displayed using a very simple description, the foreground objects would not look realistic. The ideal situation would be to have the complexity and display time of an object be proportional to the screen area covered by the object. As an object's screen space
dimensions become small, the amount of work necessary to display that object should diminish accordingly.

1.1. Management of Detail

The goal of the research presented in this dissertation is to develop a method of managing the problem of displaying the proper amount of detail. A software system was developed which allows easy generation and management of a collection of object descriptions. When an object is to be displayed, the proper amount of detail is determined and the proper description is accessed. Although the user is aware that each object has more than one description associated with it, this system is no harder to use than one which has only one description per object. The addition of multiple descriptions allows users to create more complex images without needing more computation time.

Such a system could be used in any application which creates many images of complex scenes. Flight simulators and other real time interactive systems, such as video games, lack detailed imagery due to the constraints of having to complete an image in a small fraction of a second. The level of detail management proposed in this dissertation can increase the image complexity without requiring additional computational power or speed.
Commercial animation for advertising and entertainment also can benefit from level of detail management. Individual frames of a commercial animation may take many minutes or hours to calculate. The proper management of detail can increase image complexity or reduce computation time for such animations.

Consider a typical polygon display program which goes through the following steps when displaying an object description (not necessarily in this order):

1. Transform polygons to screen space.
2. Cull backfacing polygons.
3. Determine polygon occlusion.
4. Calculate shading for polygons.
5. Scan convert polygons.

The only step which is currently dependent on screen size is the final step of scan converting. As the area covered becomes smaller, the amount of time spent scan converting also diminishes. To make the total computation load proportional to screen size, the work done in the first four steps must also be proportional to screen size. The only way to accomplish this is to reduce the number of polygons displayed as a function of screen size (or similarly of distance from the eye).
1.2. Historical Perspective

1.2.1. Subdivision Algorithms

A number of approaches which control the amount of detail displayed have been developed in the past. Catmull introduced a patch subdivision algorithm [3] which relies on screen area of the subpatch as the stopping condition of the subdivision. This approach renders a patch using computation time proportional to the screen area. Although this was not intended to be a level of detail sensitive algorithm (it was merely a convenient patch display algorithm) it has this property. A variation of Catmull's subdivision algorithm was proposed by Lane and Carpenter [16] which tested the planarity of the sub-patches and approximated them with a polygonal tile when the match was sufficiently close. This requires many fewer subdivisions than Catmull's original algorithm for gently curved patches.

A subdivision approach to the problem of level of detail management would start with a description of an object and subdivide it into constituent pieces. This suggests a detail hierarchy for each object to be rendered.
1.2.2. Hierarchical Representation

Clark suggested such a hierarchical object description [6]. He suggested that an object could be represented as a tree with the root of the tree having a very gross representation of an object. The branches of the tree divide the object into more detailed descriptions of the constituent parts. A display program using this description would be able to traverse the tree and only include that portion of the tree which is at the correct level of detail. Clark used the term "resolution clipping" for the process of terminating the tree traversal when the proper amount of detail was found.

1.2.3. Volume Representation

When representing an object, there are two main approaches; describe the surface of the object, or describe the volume occupied. Oct-trees [14] can be used to represent the volume an object occupies. The node at the end of each of the eight branches specifies that portion of the respective octant of 3-space which is occupied by the object. Each node (octant) is further divided into eight octants. This process is repeated until a sufficient resolution is achieved. This tree can be considered a description hierarchy. A display program referencing
this type of object description can traverse the tree until the volume (or cube root of the volume) of the octant visited matches the size of a pixel. The pixel is then appropriately illuminated to match the amount of volume occupied by the object in that octant.

1.2.4. Mip Mapping

The mip mapping [23] approach to antialiasing textures manipulates a texture image at multiple levels of complexity. These levels are accessed as needed to get the proper sampling rate of the texture image. The filtering of the map is done once. It is then accessed as needed to generate an image. As with any successful table look-up scheme, the savings on each of many accesses to the table offsets the expense of creating the table. With mip mapping, two levels are appropriately blended to get any intermediate level. This provides a "continuous" range of samplings from the most detailed texture image to a solid color representing the total average color. The mip mapping technique was also applied to polygonal meshes representing elevations above a regular grid. This technique is not directly extensible to general three dimensional objects as noted [23].
1.3. Description Hierarchies

The general approach to displaying the proper level of detail has been to describe an object in a hierarchical manner. Mayer and Cosman [18] reported on a real-time simulator in which objects were described at various levels of detail. By having a collection of object descriptions at various complexities, the proper level of detail could be displayed by accessing the proper object description. Figure 1 shows wire frame views of three different polygonal descriptions of a sphere.

Figure 1. Three Descriptions of a Sphere
1.4. Overview

The approach investigated in this research is the object description hierarchy approach suggested by Clark [6] and refined by Mayer and Cosman [18]. An object description hierarchy will be a simple linked list of descriptions with each node representing a full description of the object at a particular level of complexity. Object descriptions may be polygonal meshes, collections of patches, or any other representation. The type of representation may vary among the nodes within a hierarchy.

The problem of detail management is one of generating and managing the collection of object descriptions. Generation of object descriptions will be discussed in Chapter 5. Management of object descriptions will be divided into three parts: Chapter 2 discusses how to gracefully change between descriptions of an object. Chapter 3 discusses the technical aspects of blending object descriptions into one final image. Chapter 4 discusses how to decide when such transitions are to be performed. The remaining chapters deal with the results of the research and the system developed at the Computer Graphics Research Group (CGRG) at Ohio State University to support level of detail management.
CHAPTER 2

Coordinating the Transition Between Descriptions

When creating single images, the choice of which object description to display is simply a function of the screen size of the object in question. Associated with each description is a range of screen sizes for which this description has the appropriate amount of detail. Chapter 3 contains a discussion of how to select the appropriate screen size range for an object. The upper bound of applicable sizes of one description is used as the lower bound of the next more detailed description.

The screen size range corresponds to a range of eyepoint-to-object separation distances when displayed on a given resolution screen. Since objects are manipulated in object space rather than screen space, the separation distance is usually easier to deal with than screen size, although the transformation from one metric to the other is quite simple. The current viewing parameters are used to determine separation distance and choose which description to use.
When creating animated sequences, as in flight simulators, the distance between objects and the eyepoint is usually changing. As an object changes size, it may require the use of different level descriptions at different times. This implies that a method of changing between different object descriptions is needed. Three methods of changing object descriptions are presented below.

2.1. Instantaneous Transition

The simplest way to change between two descriptions is to do it all at once. Consider the view from a plane flying toward a building. When the plane is far from the building, the building is rendered using the least complex description. As the plane gets nearer, it becomes necessary to use a more complex description. When this transition becomes necessary the low level description is dropped from the display list and the next more complex description is added to the display list. This type of transition is termed an instantaneous transition. It is the least expensive method of changing levels, requiring no changes to the display software. There does, of course, need to be some software which decides which description to use and when to change. Control software will be discussed in later chapters.
The major drawback to this scheme is the noticeability of the transition. Methods for hiding or softening the visual impact of instantaneous transitions are discussed in Chapter 4.

2.2. Blended Transitions

Other methods of changing object descriptions involve blending between the different descriptions to get a smoother transition. This is usually done by using transparency to blend the images [18], but this research introduces a new method which solves certain drawbacks associated with this approach. The exact methods used are covered in Chapter 4.

It should be noted that during a transition, two descriptions are being displayed at the same time. This implies that during the transition, more work is being done than if only the more detailed description were displayed. The savings on subsequent frames of displaying the simpler object description must outweigh the increased computation during the transition for this scheme to net a decreased work load.
2.2.1. Blending over distance

The first method of coordinating the blending of object descriptions is called blending over distance. It is done by specifying a transition zone during which the blending will occur. This zone may be specified in either screen size or eyepoint-to-object distance (again assuming constant screen resolution). In either case, the transition zone is centered about the lower bound of applicable distances of the more detailed description (call this distance $T$). Assume that the zone specified for a transition is two units long. The near edge of the zone is encountered when the eyepoint-to-object distance is one unit less than the transition point (i.e. $\text{NEAR} = T - 1$). The far edge of the zone is encountered when the eyepoint-to-object distance is one unit more than the transition point (i.e. $\text{FAR} = T + 1$).

Only the more complex description will be accessed at the near edge of the transition zone. Similarly, only the less complex description will be accessed at the far edge. Inside the zone, both descriptions are accessed and blended together. A blending function, $F(x)$, where $x$ is the separation distance, is defined so that for $x$ less than $\text{NEAR}$, $F(x)$ is 1.0; for distances greater than $\text{FAR}$, $F(x)$ is 0.0; and between $\text{NEAR}$ and $\text{FAR}$, $F(x)$ drops from 1.0
to 0.0. Figure 2 shows two possible blending functions. Linear blending as shown on the left is easier to calculate and produces a good result. The blending functions $F[x]$ and $G[x] = 1.0 - F[x]$ are used to determine the weighting applied to the descriptions being blended. $F[x]$ determines the weighting of the more complex description, while $G[x]$ is used for the less complex description. These functions can be applied when the eyepoint-to-object distance is decreasing as well as when it is increasing.

Figure 2. Sample Blending Functions
Figure 3 shows a graph of linear blending functions used to generate the correct weighting of each description during the transition. In practice, when a description is to be displayed with a weighting of 0.0, it is dropped from the display list.

One potential problem with blending over distance is that the distance from the eye to the object may get stuck in the middle of a transition zone. In a chase scene, it
is quite common to have the object being pursued remain at a relatively constant distance from the pursuer. If the separation distance happens to fall in a transition zone, then two descriptions of the pursued object will be displayed (potentially forever) at the appropriate weighted intensity. Unfortunately, displaying an object description using a small weighting is just as expensive as displaying it using a large weighting. Thus displaying both descriptions represents an increased work load on the system and is worse computationally than not changing levels at all! One solution is to have a timeout feature on each transition. If the transition is not complete when the timer expires, the remainder of the transition is forced to occur.

Another problem which might arise in a chase sequence is the eyepoint-to-object separation may oscillate through the transition zone. This would cause the object being pursued to undergo frequent transitions. The frequent change in descriptions being displayed might be distracting, and would cause an increased workload for the system. It was stated earlier that the smallest applicable screen size of one description was used as the largest applicable size of the next lower description. If this restriction is relaxed and the applicable sizes are allowed to over-
lap, a buffer zone is created. The location of the transition point is then determined by which description is being phased out. An object which is getting closer waits longer to become detailed, while an object which is getting farther away waits longer to become more simple. This creates a much larger zone through which the separation distances must oscillate before the problem arises.

One additional problem arises when the eyepoint-to-object distance is changing very quickly. For very large relative velocities, blending over distance is identical to an instantaneous transition. If the separation distance is on one side of the transition zone in one frame and on the other side of the transition zone in the next frame, the transition is equivalent to an instantaneous transition. The problems associated with instantaneous transitions apply in this case also.

Rapid movement of an object can be used as a justification for lowering the amount of detail needed when displaying an object [18]. The human eye is not capable of discerning great detail on fast moving objects. With large relative velocities, temporal aliasing becomes more apparent. If temporal antialiasing is done, a less detailed description can be used because of the blurring caused by the motion. If a single point on an object cov-
ers a path four pixels in length when blurred, the effect is a four to one decrease in the resolution of the screen along the path of the object. The resolution perpendicular to the path is not changed\(^1\). Once the amount of blurring due to temporal aliasing is known, it is possible to calculate the amount of resolution lost and to compensate appropriately when determining which object description to display.

In order to implement blending over distance, there must be a control program which monitors the eyepoint-to-object distance, evaluates the blending functions, and directs the display of the appropriate descriptions.

Although not detailed in their paper, blending over distance is probably the technique used by Mayer and Cosman [18]. Hardware simulator manufacturers are understandably tight lipped about the internal workings of their proprietary software.

\(^1\) This problem of skewed compression was encountered by Williams [23] when antialiasing texture mappings. As a texture mapped plane is viewed obliquely, the texture map is compressed in one dimension (depth) only. Williams' solution was to use the greatest compression value and ignore the extra fuzziness generated along the uncompressed axis.
2.2.2. Blending over time

The second method of coordinating the blending of object descriptions is called blending over time. It is done by specifying a transition interval during which the blending will occur. In an animated sequence, a time interval is usually specified as a number of frames. The transition between descriptions is done by blending two descriptions over a specified number of frames. Again, there are blending functions, \( F[x] \) and \( G[x] \), which specify the weighting of each description. In this case, the blending functions are a function of frame number (or time), rather than of distance.

This approach eliminates the potential for getting stuck between descriptions because time cannot be stopped. The transition will complete within the specified number of frames.

As with blending over distance, there are some problems with blending over time at very high relative velocities. With high relative velocities, the distance traveled during the transition may approach or exceed the distance between transitions of different levels. This creates the possibility of having two or more transition zones overlap. When two transition zones overlap, there
are three descriptions being accessed and blended at the same time. This could create a sudden large work load which might exceed the capacity of a real-time simulator. As was stated previously, a lower complexity description can be used on objects with high relative velocities.

The control program needed for this blending method is more complex than for either instantaneous transitioning or blending over distance. In order to set up the time interval, the control program needs to know when the eyepoint-to-object distance will be appropriate for a transition. It then will add half the number of frames to both sides of this point and apply the blending functions appropriately. It is obvious that this requires some knowledge of the future (when the object will be the proper distance away) or at least a good predictor of future behavior. In some settings such as animation, this future knowledge is very easy to get. In other settings such as real time simulators, future knowledge is very hard to predict. Current velocities can be used as a good predictor for the near future.
2.3. Additional Problems

There are some pitfalls to using eyepoint-to-object distance as the metric for determining appropriate level of detail. This metric assumes that the distance to the object is large in comparison to the size of the object itself. If this is not the case, the near part of an object may be rendered in lower detail than appropriate while the far part of the object may be rendered in higher detail than needed. A single object representing a railroad track going toward the horizon is a good example. The railroad ties in the front should be rendered in more detail than those at the horizon. It seems awkward to require that each railroad tie and each section of rail be considered a separate object.

This is the one major drawback of having each node in the hierarchy consist of an entire object description. Clark's [6] object hierarchies are described by a tree structure in which each node contains a small part of an object. Lower levels of the hierarchy had more complex descriptions of the parent parts. This organization would allow different parts of the object to be displayed at different complexities. Blending portions of the hierarchy to join different complexity descriptions of neighboring parts would be necessary.
In the case of the railroad track, a tree structured hierarchy is simply a different organization for having each tie and each rail a separate object. Each tie is either a different object or a different node in a hierarchy. The generation of the treelike description hierarchy is as difficult as describing each part separately.
CHAPTER 3

Display of the Transition

3.1. Using Transparency

In order to display the transition between two descriptions of varying complexity, the display program must be able to display polygons (or other surfaces) with less than full intensity. One possibility is to use existing software which displays transparent objects. Both Mayer [17] at Evans & Sutherland and Evans and Fadden [9] at General Electric have used transparency to blend between object descriptions, but neither group has published the details of their methods.

This research investigated the use of transparency to blend object descriptions and found a variety of problems associated with it. An alternative method was developed using the concept of pixel coverage. This new method will be discussed after pointing out some of the failings of using transparency.
Recall the linear blending functions \( F(x) \) and \( G(x) \) introduced in the last chapter. To derive transparency from the blending functions, transmittance is calculated. A transmittance of 1.0 is totally transparent, while a transmittance of 0.0 is opaque. Transmittance can be derived from the blending function by subtracting the weighting from 1.0 (e.g. \( F(x) = 0.2 \) implies that the transmittance should be \( 1.0 - 0.2 = 0.8 \)); The transmittance of one description will go from opaque to transparent, while the other description will change from transparent to opaque. In the middle of the transition, the two blending functions have the same value. The two descriptions are displayed using 0.5 transmittance for each.

Transparency, however, is a multiplicative process. The amount of light absorbed when passing through two 0.5 transmittant surfaces is not 1.0 (0.5 + 0.5), but .75 (0.5 + (0.5*0.5)). Figure 4 shows the net transmittance when using the linear blending functions from Figure 3. For most of the transition, the net transmittance of the object will not be 0.0 as desired. This means that background objects will appear through the object in transition. This is definitely undesirable.
There are other combinations of blending functions which will eliminate this problem. Figure 5 shows a linear blending function combined with an effective instantaneous drop. The description which is to be added is displayed with increasing opacity according to the
standard $F[x]$ blending function. The description which is to be removed is displayed as totally opaque. Once the new description reaches full opacity, the old description is dropped from the display list. It is critically important that the new object description which is fading in be displayed as if it were in front of the old description
which is disappearing. This is required even though spatially they may intersect, or even overlap in the other order. If the objects are displayed in the wrong order, the new description which is fading in is hidden by the opaque description. When the opaque description is finally dropped from the display list, the new description would become visible. This would appear similar to an instantaneous transition. If the two objects are not forced to be one in front of another, then a proper hidden surface elimination algorithm would calculate the intersections of the two descriptions and a combination of the two effects would occur.

Many display programs do not have this facility to rearrange the depth sorting of objects. With a clustering scheme such as described by Crow [8], the ordering of the objects is easily modified. This scheme also greatly facilitates the mixing of surface representations among the various descriptions. A bi-cubic patch based display program can display the high level description while a polygon based display program can display the next lower description. In scanline based systems which must have all of the surface elements on hand to do the sorting, this mixing of representations is very difficult. The span buffer approach used by Whitted [22] also allows mixed surface representations.
Another problem with this scheme is that the instantaneous drop of the disappearing description will be visible if the new description does not totally cover the old description at the time of the drop. Since this method must be used for transitions which change to more complex and to less complex descriptions, the silhouette of the two levels would have to be identical in order to hide the instantaneous drop going both ways. Having identical silhouettes would defeat the purpose of having multiple descriptions, namely to have less complex descriptions available for display when the object is far from the eye.

The best solution when using transparency is to have blending functions like those in Figure 6. This method replaces the instantaneous drop with another linear blend. The position of NEAR, FAR and T are slightly different in this scheme. NEAR is now $T - 2.0$ and FAR is $T + 2.0$. This places T in the middle of the transition as desired and keeps the duration of the blends the same as in the other schemes. It should be noted that there is no net transmittance during the transition between descriptions.
3.2. Using Pixel Coverage

A new approach developed in this research was to use the concept of pixel coverage. One way of thinking about displaying a weighted description is to consider the weighting factor as the maximum amount of each pixel that the description will cover. When two descriptions are at
equal weights (0.5), each pixel covered by the object's image should be influenced equally by the two descriptions. Or, similarly, one half of the pixel is covered by one description while the other half of the pixel is covered by the other description.

Using pixel coverage, it is possible to implement the obvious blending function of two simple linear blends as in Figure 3. This solution does not have the drawbacks associated with transparency because coverage is additive. 0.5 covered plus 0.5 covered equals 1.0 covered. Also, the order of processing the polygons in the various descriptions is not important as it was in the more complex blending schemes proposed for using transparency.

The display programs at CGRG use pixel coverage to do spatial antialiasing. When tracking edges of polygons while tiling, the amount of each pixel covered by the polygon is calculated and stored. When a pixel is accessed, the new pixel color is blended with the old color according to the old and new coverage values. Once a pixel is totally covered, no subsequent accesses to the pixel will change the color of the pixel. By scanning in from front to back, the proper hidden surface elimination and spatial antialiasing is accomplished. Once the image generation is complete, any pixels which are only
partially covered must be proportionally blended with the background to finish the antialiasing of the image.

To implement the blending of object descriptions using pixel coverage, each description uses the blending function value as a scalar to be multiplied to the calculated coverage. This limits the maximum contribution to any one pixel by any one object description to be exactly equal to the desired weighting.

Crow and Howard [7] developed the frame buffer at CGRG. It has 32 bits: 8 each for red, green, blue, and coverage. This allows hardware support of the pixel coverage algorithms discussed above. A hardware blending function which will do the pixel blending is currently being developed.
As stated in Chapter 1, one way to create more realistic imagery is to maintain a constant density of objects in the entire field of view. This requires that objects in the distance be rendered in very low detail. As the eyepoint approaches an object, the amount of detail must increase so that a realistic image is maintained.

In real-time systems, there is a limit to how much computation can be done before the next image must be displayed. The longer a low complexity description can be displayed, the less computation is required. This savings must be weighed against image quality. Using lower complexity descriptions when an object is far from the eye provides visual complexity without the associated computational burden of processing vast quantities of surface elements. Using lower complexity descriptions when an object is far from the eye also eliminates the spatial aliasing of small details present in complex descriptions.
4.1. Choosing transition type

Since instantaneous transitions present the smallest overhead of any of the proposed transitioning schemes, they should be used when appropriate. The major problem with this method is the sudden change in the image at the time of the transition. If instantaneous transitions are used, they should be staged so that the sudden transition is least noticeable. There are a number of ways to diminish this sudden change. One is to have a large number of descriptions which are quite similar in detail. The noticeability of the transition is proportional to the amount of change which occurs in the transition. The large number of descriptions must be generated and stored, or generated at run time. The Lane-Carpenter [16] patch subdivision technique mentioned in Chapter 1 generates a polygonal object description based on the planarity of the resultant sub-patches. This planarity measure can vary with the eyepoint-to-object distance. In an animation using this display algorithm, there would, in effect, be an instantaneous transition of polygonal object descriptions at each frame.

Another possible way of minimizing the blink caused by an instantaneous transition is to stage the transition at a great distance from the eye. At a great enough
distance, a large change in object description can be made without a noticeable change in the image. Having large distances between the transitions implies that there are very few levels in the object hierarchy.

This technique of staging transitions at a great distance was used in one of the GE [9] simulators. The gaming area was divided into 16 circular rings representing increasing distance from the eye. Each object description had associated with it a 16 bit word which designated in which rings that description was to be used. When an object passed from one ring to another, a different description would pop in. In practice, there were only two or three descriptions of each object. It was reported that the objects were never far enough away to totally hide the transition.

One way of eliminating the blink entirely is to schedule the transitions to coincide with occlusions. The occlusion may be by another object, or with a cloud, explosion, or other phenomenon.

One major drawback to this method is that occlusion information is not usually present. Most hidden surface algorithms do not maintain the necessary back pointers to determine occlusion. Some algorithms can be modified to
get this information (scan-line algorithm), while many cannot (depth buffer, painter's algorithm). Occlusion can be determined by an animator when staging an animation, but for complex scenes this is an undesirable burden. In video games and war simulations, explosions often fill the entire screen with fire and smoke. This would be an easy occlusion to detect and use.

In the general case though, a blend over a number of frames will be necessary. Whether the blend is cued to time or distance of separation, the problem of determining when to schedule the transition remains. There is a trade-off between the distance at which the transition takes place and the noticeability of the transition. Mayer and Cosman [18] report that blended transitions often can be done twice as close to the eye as instantaneous transitions without changing the noticeability.

With blended transitions, the choice of where to stage the transition becomes, appropriately, one of having the appropriate level of detail displayed, rather than trying to hide the transition.
4.2. Choosing appropriate transition distance

Assigning the appropriate screen size to a description is the final problem to be solved. A variety of factors influence the distance at which an object description does not present enough visual detail. Mayer and Cosman [18] identify four factors: Size of the object on the screen, color of object relative to background, shape of object's silhouette, and difference between the current description and the new description. Additional factors include relative velocity of the object, type of transitions being used, and amount of detail in the description. Tracking the viewer's eyes and reducing the detail on the periphery was suggested [9] as another factor which might allow a lower level description to be used.

The brute force approach to assigning the appropriate screen size is to display the object description at various screen sizes and pick the largest size which seems to have enough visual detail. This marks the upper bound of appropriate screen size. The lower bound is normally taken from the next lower level description's appropriate upper screen size. In Chapter 6, a method of automatically picking appropriate size ranges based on the complexity of the description will be discussed. Factors such as relative velocity and position on screen are not
constant for a description. They must be calculated at display time and be used to modify the appropriate screen size as needed.
5.1. Object Types

Before looking at the process of generating object descriptions, it is useful to consider the nature of various objects and how they should be represented. Some objects, like checkerboards, signs, and buildings are mostly flat, but have a variety of colored areas which blend into a solid color when viewed from a distance. The less complex descriptions should mimic this color blending by having fewer surface elements with an average of the colors in the corresponding part of the detailed object.

Some objects, like humans, animals, and wheat fields resemble blobs with small projections which cannot be seen from great distances. The detailed description will have all of the protuberances, while the lower level descriptions will omit them.

Other objects, like crane booms, towers, and lace doillies look solid when viewed from a distance, but are
in reality full of holes. The most detailed description will define all of the holes as they exist in the object. The lowest level description might be a single solid form. The use of transparency in such lower level descriptions can simulate the airy nature without having to model it in detail.

This classification of object types is not meant to be exhaustive or exclusive, rather it is meant to suggest some of the different problems associated with developing object description hierarchies. A car, for example, might have qualities from each category. A car might have racing stripes which should be color blended into the body color in the lower descriptions. There might be a radio antenna in the most detailed description, but none in the lower ones. Finally, the grill might be modeled as one slightly transparent surface in the lower level descriptions.

5.2. Individual Description Generation

The biggest problem with using object description hierarchies is not the software development necessary to display the proper detail. The biggest problem is the generation of the various object descriptions. This requires much more manpower for a modestly large database
than the software development. And, unfortunately, the data generation process is not bounded by a certain set of useful objects. New areas of the world (airports, etc.) need to be modeled whenever the gaming area changes. New planes, cars, and other man made objects are being manufactured and will need to be modeled. Finally, the objects in commercial animation are limited only by the animator's imagination. Object description generation is a process which will never cease.

The sophistication of data generation programs has increased greatly since computer graphics first became a field of study. The simplest automated tool for data generation is a text editor. The data generator usually makes a model or detailed plans of the object to be described and then measures the model or plans to get the object space coordinates of all of the points involved. These points are entered into an ASCII point dictionary using the text editor. The points are then connected to define the polygonal mesh describing the surface of the object. This list of polygon definitions is also entered into the ASCII file. The potential for making mistakes is large. Obviously this is not the preferred method for generating complex object descriptions.
A simpler method is to use digitizing hardware to measure the plans (2-D digitizer) or the model (3-D digitizer) and automatically generate the point dictionary. The polygon dictionary can be generated by hand or automatically from some implied ordering in the point dictionary. Sutherland [20] developed a system which could reconstruct a three dimensional representation of an object using various views of an object. This was accomplished by using a large digitizing tablet with multiple pens. Each pen was located at the same location on the different views of the object. The three dimensional location of the point was derived from the various pens' locations.

More sophisticated methods allow the user to generate object descriptions interactively. Carlson [2] outlines a variety of methods of generating three dimensional object descriptions using an interactive system. Solids of revolution can be created by specifying the profile to be revolved and the path of revolution. Extruded solids can be created by specifying the shape of the extrusion. Tubular objects are defined by defining the centerline which the tube is to follow and the diameter of the tube along the centerline. Lofted objects can be created by defining various cross sections and linking them to form
polygons. There have been many papers [5, 11, 12] dealing with the best way to link up the cross sections to get the most accurate surface reconstruction.

If the data generator has access to a solid modeling system [1, 10, 19, 21], then the process of generating detailed object descriptions is much simpler. By using set operations on primitive solids, the final shape can be approximated. It is possible that some of the early approximations will suffice for less complex descriptions.

In general, to generate multiple descriptions, each level must be described independently. The amount of overlap between the description of one level and the description of another level is, in practice, small. This means that the data generator must create two, three, or more representations of the same object. Figure 7 shows an object description hierarchy generated by describing each level independently.

5.3. Description Modification

One very important tool for anyone generating or modifying data is an object editor. Without such an editor, manipulation of data is even more difficult than the original description of it. Without an object description
Figure 7. Hierarchy With Each Level Generated Independently

editor, users would dump the binary point/polygon dictionaries to ASCII, modify the dictionaries using a text editor, and convert back to binary. After making the changes, the object description was then displayed to view the changes made. This process usually required many iterations to make even minor changes in the structure of an object.
Another useful tool is a description verification program. Hand built object descriptions are quite prone to having a variety of inconsistencies such as missing or reversed polygon definitions. A description verification program cleans up object descriptions and points out any violations of display program restrictions such as concave polygons.

5.4. Automatic Description Generation

One of the goals of this research was to investigate methods of automatic object description simplification. Available, the data generator can concentrate on the final, highly detailed object description and then automatically generate the lower levels of the hierarchy. While others have used the object hierarchy approach for image generation, this represents the first general investigation of automatic simplification techniques.

5.4.1. During generation of high level object

A number of automated methods of generating object description hierarchies were investigated. The first method is to generate multiple descriptions during the generation of the most detailed description. By doing the simplification during this phase, certain features of the
object's structure can be easily accessed and used to facilitate the simplification.

To investigate this approach, the data generation system, DG [2], at CGRG was modified. To create an object using DG, the data generator specifies one or more two dimensional plane curves. Actually, the plane curve is approximated by a connected set of line segments. These plane curves are used as profiles for solids of revolutions, paths for solids of revolutions, mid-lines for tubular extrusions, and outlines for extruded objects. Having access to these plane curves, which are intimately related to the final structure of the object description, makes the simplification process much simpler.

The modifications made to DG involve a plane curve simplifier which creates a less detailed approximation of a plane curve from the original. By simplifying the plane curves before doing the revolution or extrusion, a less complex description is made. The number of times and degree to which the plane curve is simplified can be adjusted to get reasonable sets of object descriptions. Three separate methods of simplification were investigated.

(1) The first method is called the equal arc length method. The original approximation to the desired
plane curve is sampled by measuring the total arc length of the curve, dividing the length by the desired number of segments, and then sampling the curve at the appropriate distances. This method does not guarantee that the resultant segments will be equal length, just that the endpoints are the same distance apart when measured along the curve. If the object being modeled is supposed to have a curved surface, a Catmull-Rom spline (piecewise cubic interpolating spline) [4] is fitted through the original plane curve approximation. The spline is approximated by a small number of segments between each of the control points on the original plane curve. This more detailed set of points is then sampled to get a plane curve consisting of the desired number of segments. This allows the sampled points to lie along a smoother profile. Figure 8 shows a simplification using this technique.

(2) The second method is called the *equal segment count* method. The original segment count is divided by the desired number of segments. This number, N, is the sampling frequency. The original plane curve is then sampled at every N points along the original plane curve.
Figure 8. Simplified Plane Curves Using Equal Arc Length

curve approximation. Fractional portions of segments are calculated by dividing the segment in which the fractional part lands according to its length. This technique can be considered a normalized equal arc length scheme where each segment is given the normalized length 1.0. Figure 9 shows a simplification using this technique.
(3) The final method is called the \textit{smallest edge first} method. The shortest segment in the original plane curve approximation is selected and the endpoints are averaged together. This results in a new description with one fewer segment. This process is repeated until the desired number of segments is reached. Figure 10 shows a simplification using this technique.
Figure 10. Simplified Plane Curves Using Smallest Edge First

In practice, none of the three techniques gave the best results consistently. The user of the modified DG is asked to select which of the three simplified plane curves comes closest to the desired shape.

This technique of generating multiple descriptions at the time when the most detailed description is created can also be used in traditional solid modeling systems which use geometric primitives, such as blocks, wedges, cones,
cylinders, hemispheres, and fillets [1,10,21]. If each of
the curved geometric primitives (planar primitives such as
blocks can be represented exactly using a few polygons)
are represented at more than one level of detail, and all
set operations are performed concurrently at each level, a
collection of descriptions will be created.

5.4.2. After Generation of Detailed Object Description

Other methods of simplifying object descriptions take
a previously described object and simplify it directly.
By doing the simplification after the fact, the structure
of the object must be discerned directly from the descrip-
tion. This makes the process much more difficult. It
might be possible to reconstruct the profiles and paths
used to create a solid of revolution by examining the
topology of the polygonal mesh. This would probably
require techniques borrowed from Artificial Intelligence.
There are a few methods which are less elegant, but easier
to program and quite effective in generating less detailed
descriptions.

5.4.2.1. 3-D Grid Constraint

One approach to object simplification constrains the
possible positions that the points of an object may
occupy. The point dictionary of the most detailed
description is modified to lie on a three dimensional grid. This is done by rounding each coordinate of each point to the closest grid position. In the original description, points may occupy any location (Actually, points are constrained to lie on a grid having spacing of the precision of the computer in which the description is stored). By increasing the size of the grid elements, the possible positions are increasingly constrained. This reduces the complexity of the object by coalescing adjacent points. The smallest edges are removed at smaller grid sizes, while the large edges remain.

After the point dictionary is processed, duplicate points are removed and the polygon descriptions are appropriately modified. The final step is to examine each polygon definition for degenerate edges (edges between points which have been coalesced). This clean-up operation can be done using an object description validator as mentioned earlier.

This method of constraining the point locations is similar to the quantization which occurs when an object is displayed on low resolution screens or when an object's screen size diminishes. The one problem which arose when using this scheme was the tendency of polygons to twist and become nonplanar. This problem can be overcome by
restricting the most detailed description to be composed entirely of triangles. Figure 11 shows a collection of object descriptions which were generated using this method.

Figure 11. Hierarchy Generated Using 3-D Grid Constraint

5.4.2.2. Small Edge Deletion

Another method of description simplification attacks the most detailed portion of the description first. The polygon dictionary is read and a list of edges sorted by length is created. Since the most detailed portions of an object description have smaller edges than less detailed
portions, the edge list is roughly sorted by amount of detail. The edge list is traversed starting at the smallest edge. As an edge is visited in the list, the points defining that edge are coalesced. This coalescing can be done by averaging the coordinates of the two points or picking the one furthest from the center of mass.

Again, the polygon dictionary must be cleaned up after edge deletion. In this case, the clean-up must be done after each edge removal or the coalesced points might get separated by subsequent coalescing of other neighboring points. When two points are coalesced, the edge lengths of all edges referencing those points change (the one between the points becomes zero length, while the others generally become longer). The edge list must be re-sorted after each edge removal. Since the edges are already sorted and only a few values have changed, a complete sort is not needed. There are a variety of sorts which efficiently sort an almost sorted list [15]. Figure 12 shows an object description hierarchy generated using this approach.

5.5. Comparison of Techniques

The concurrent generation of multiple descriptions using the modified version of DG gives the best results.
Unfortunately, the remainder of the data manipulation stage of the pipeline is not set to handle multiple descriptions. The user must send each description through the description verification system. The user must also pay attention to any manipulation which is performed to an object. If two objects are combined to make a third object, the user must be sure that the lower levels are appropriately processed. Similarly, any modifications made with the object editor such as deleting faces, moving vertices, or setting up coloration files, must be done on each level of the hierarchy independently.
While the post generation methods of grid constraint and edge deletion can work on an object after all modifications have been made, the results are not quite as pleasing. The user must often fix certain problems in the new descriptions using the object editor. There is, however, a great deal of time saved doing small modifications rather than completely designing each lower level individually.

The small edge deletion method works best on objects which have a great variety of edge lengths. Checkerboards and other regular polygonal meshes have a very narrow range of edge lengths. When the threshold gets into this range, the entire object may disappear.

A final problem is the proper selection of the grid spacing or the threshold for describing each of the lower level descriptions. The proper distance can be arrived at by interactive intervention by the user, or by comparing the complexity of the resultant object with the desired complexity. The metric for complexity can be the appropriate screen size of the object description as determined by the method outlined in Chapter 6 in the section on the Automatic Hierarchy Manager.
CHAPTER 6

Animation Environment

The Computer Graphics Research Group (CGRG) at Ohio State University has an animation environment which is organized into a four stage pipeline:

1. Data generation and manipulation
2. Scene description
3. Viewing and Rendering
4. Post-processing

6.1. Data Generation and Manipulation

Most of the object descriptions generated at CGRG are made using DG [2]. It allows users to create object descriptions using solids of revolution, projected objects, lofted objects, or tubular objects. There are also a number of special purpose programs for generating grids, gears, spirals, and other shapes.

Once a description is created, it can be combined with other descriptions, knifed into two portions along a plane, or arbitrarily modified using an object editor.
There is an object verification program to ensure that display program restrictions are met in each object description made.

6.2. Scene Description

After a set of object descriptions has been created, an animation script can be developed using one of the animation systems developed at CGRG. There is TWIXT, an event-driven system for animating single objects and simply systems of articulated objects [13] and SA, a procedural based system for controlling the motion of complex articulated figures [24]. Each of these systems produce a series of single scene descriptions which are passed to SCN_ASSMBLR [8]. SCN_ASSMBLR is a program which controls the placement, orientation, and interaction between object descriptions and light sources. This program produces an output file which is passed on to the display program in the next stage of the pipeline.

6.3. Viewing and Rendering

The display program receives the object description, light source information, and viewing parameters necessary to generate an image. The display program is responsible for doing the shading, occlusion, and tiling of the
polygons in each object description. The decoupling of the display program from the scene description program allows easy substitution of display programs. This creates a very flexible environment for research in new display algorithms.

6.4. Post-processing

Once an animation is calculated, a variety of post-processing programs are available for making titles, fades, dissolves, mattes, and other effects. By having a pipeline of animation programs, the purpose of each stage can be rigidly defined. This allows more flexibility in creating new programs to experiment with new ideas. Post-processing packages should be designed to work on frame buffer images without regard to the method used to generate them. Incorporating responsibilities from one stage into another might make the process more efficient, but much less flexible.

6.5. Modifications to the Pipeline

The main goal in incorporating level of detail management into the above pipeline was to make as few changes in the structure as possible. As much as possible, the level of detail management features were kept
invisible to the user. Obviously, the data generation process must be modified. Also, the rendering process will change. The other stages of the pipeline were not effected.

6.5.1. Changes to the Data Manipulation Stage

6.5.1.1. Object Editor

In order to generate and manage a collection of object descriptions, a number of software tools were developed. The most important was an object editor called INSPECT. INSPECT allows the data generator to easily inspect and modify an object description. INSPECT reads in the object description and displays it on a vector-refresh display. This display has hardware transformation capabilities which allow the user to rotate, scale, and translate the object so that the area of interest is easily viewed. Along with the display of the current point or polygon is a description of the point or polygon on the terminal. For points, the current position, color, and a list of polygons in which it is referenced are listed. For polygons, the polygon definition, current color, neighboring polygon numbers, and texture coordinates are listed. The editor also works using only a terminal, but there is, of course, no vector display of the object.
The commands in INSPECT are single letter commands which are usually the first letter of the function of the command (e.g. 'a' is for add point or polygon). The display of the object can be either a wire frame of the entire object with the current point or polygon highlighted, or a zoomed in version of the current point or polygon and neighboring points or polygons. The user interface includes a consistent help facility. At any time, the user may type 'h' and get appropriate help about whatever operation is being performed. For most of the operations, there is an easy way to undo any changes made (e.g. 'd' for delete and 'u' for undelete). As with most editors, there is a guarded exit so that the program cannot be exited accidentally without writing out changes made.

The primitive operations on points include: add, color, copy, delete, and move. The polygon operations include: add, break, color, copy, delete, edit, join, move, reverse, and texture map. The edit operation on polygons allows easy reorganization of the point list of a polygon. Individual points in the polygon description can also be copied and/or moved from within the edit command. All movement of points in this mode is constrained to lie in the plane of the original polygon. This allows easy
modifications to be made without ruining the planarity of the polygon.

Users can set up sets of expressions which will do conditional deletion of polygons and conditional coloration of points and polygons. Expressions consist of an assignment phrase and a conditional phrase. The variables which can be accessed are red, green, blue, and transparency of each point or polygon, and the point or polygon number (dictionaries are numbered sequentially) or any modulus thereof. A color mnemonic dictionary is maintained so that commonly used color coordinates can be referenced using convenient names (e.g. (1 0 0 0) can be the definition of the mnemonic 'red'). The syntax of the expressions was modeled after that of the C programming language. A grid can be colored as a checkerboard using a single expression: (c = red: n%2 == 0).

Appendix A contains a complete user's manual for INSPECT. It contains a complete description of all of the commands and the complete syntax for expressions. Some of the details are necessarily site dependent to the hardware at CGRG, but the majority of the manual is still useful for getting a better feel of the capacities and capabilities which were determined to be useful in our environment.
6.5.1.2. **Object Description Validator**

Once an object description is created, it must be validated to insure that the restrictions of the display programs are satisfied. A data validation program called CHECK was developed. It performs a variety of tests which detect and in some cases correct inconsistencies in an object description. The user can easily select all or any subset of these tests to perform on an object description.

The tests which CHECK will perform are:

**Test b - backfacing test.** This test looks for neighboring polygons which are defined inconsistently. Polygons are supposed to be defined by a clockwise traversal of the points when viewed from the outside of the polygonal mesh. If one polygon is defined in part by an edge from point \( a \) to point \( b \), then its neighbor across that edge should be defined in part by an edge from point \( b \) to point \( a \). If neighboring polygons have the same directed definition, then one has the wrong orientation. The first polygon in the object is assumed to be in the correct orientation. All of its neighbors are tested and any inconsistent ones are reversed. This process is repeated recursively for each of the neighbors. The one problem
with this test is that some objects, such as Moebius strips and Kleine bottles, do not have well defined insides and outsides. This situation is easily detected by marking each polygon as it is processed. If a marked polygon is inconsistent with the polygon currently being processed a Moebius strip of polygons has been traversed!

Test c - concave polygon test. This test looks for concave polygons in the object description. Concave polygons are reported to the user. It is up to the user to split any such polygons into a set of convex ones.

Test d - degenerate edge/polygon test. If a polygon contains an edge between two points which are redundant (see test r), that edge is considered degenerate and will be removed by this test. If a polygon ends up with fewer than three good edges, it is considered degenerate and removed.

Test i - identical polygon test. This test looks for polygons with the same definition and removes one. Note that two polygons may use the same points, but be oriented in the opposite direction. In this case, both polygons are kept.
Test 1 - collinear edge test. This test looks for points in a polygon definition which are collinear. Since many display programs calculate polygon normals using the first three points in the definition, this test will rotate the polygon definition so that they are no longer at the front. The sensitivity of this test is adjustable, so that "almost" collinear also passes.

Test n - neighbor test. This test looks for polygon edges with no neighboring polygons and for polygon edges with more than one neighboring polygon. Normal closed polygonal meshes will have two polygons sharing each edge. This test reports the existence and location of any such edges, but it is up to the user to investigate further.

Test o - open edge closure test. This test looks for open edges and tries to find multiple neighbors across the edge. For example if polygon one has an edge from point a to point c, but no other polygon has an edge from point c to point a, then this edge from point a to point c is considered an open edge. If there is a polygon which has an edge from point c to point b and another polygon which has an edge from point b to point a, and if point b is collinear with
and between point a and point c, then the definition for polygon one can be changed from a - c to a - b - c, thereby closing the open edge! Again, the sensitivity of the collinearity test is user adjustable.

Test p - planarity test. This test looks for non-planar polygons. The sensitivity of this test is user adjustable. Non-planar polygons are reported to the user. It is up to the user to split any such polygons into a set of planar ones.

Test r - redundant point test. This test looks for points which are at the same or nearly the same location in object space. The sensitivity of this test is user adjustable. This test is sometimes performed prior to other tests to get the proper neighbor relationship. The redundant points will be coalesced into one point, with all polygons referencing only the one single point.

Test s - sliver polygon test. This test looks for polygons which contain very small angles at vertices. Some display programs have problems with very thin polygons. This test flags such polygons, but no changes are made to the object.
Test u - unreferenced point test. This test looks for points which are not mentioned in any polygon definition and removes them.

6.5.1.3. Automatic Hierarchy Manager

After a set of descriptions of an object at different levels of complexity is created, the descriptions must be joined together into a hierarchy. Also, the applicable size of each description must be established. Object descriptions at CGRG consist of an ASCII header file containing pointers to the point and polygon dictionary file, coloration files, and texture mapping files. To set up a collection of descriptions for a single object, the ASCII header file is updated to point to the next lower detail object description. This sets up a singly linked list of ASCII header files. A program, SETLOD, was developed which automates the ASCII header file modifications which are needed to link descriptions into a hierarchy. It also guides the selection of appropriate sizes. This program examines the object description and measures complexity by edge length. As an object gets further from the eye, the screen size of the object diminishes. Similarly, the screen size of each edge of the object description also diminishes. This program measures the distance at which each edge length becomes the size of one pixel (this
requires knowing the screen resolution). Figure 13 shows a graph of percentage of edge lengths which are smaller than a pixel at a given distance from the eye. Note that the edges themselves become subpixel length earlier because most edges are not oriented perpendicularly to the line of sight.

Figure 13. Edge Length Graph of a Three Level Description Hierarchy
By examining the edge length graph, an appropriate separation distance is derived. The distance at which a description is too detailed is derived by finding a distance at which "most" of the edge lengths are subpixel size. The distance at which the description is too detailed is not as useful as the distance at which the description is not detailed enough. This point is derived by finding a distance at which "most" of the edge lengths are greater than pixel size. The definition of "most" was derived by comparing the output of SETLOD with distances determined by visually choosing the distance at which the lack of detail became evident. The percentage used is in the 70-80% range for blended transitions.

Once the proper transition distance is established for one description, the user is queried for the name of the ASCII header file of the next lower description. This process of linking together descriptions and assigning an appropriate size is continued until the last description is reached. At this point, the user may specify that this lowest complexity description should remain at all distances or fade out completely. This fade out is done by blending the lowest level description with a null or empty description.
This method of assigning appropriate size does not consider most of the factors mentioned in Chapter 4. These factors might indicate that an object description can be displayed much closer to the eye without a noticeable lack of detail. The distance given by SETLOD is to be considered an upper bound of possible choices. The user has the final say in modifying the suggested size range specified by SETLOD.

6.5.2. Scene Description

Due to the modularity of the animation pipeline, no changes were required in this stage. The user simply references the most detailed description when setting up a scene or animation. The level of detail management is done at the time of the actual rendering.

6.5.3. Changes to the Rendering Stage

The output from the scene description stage lists the most detailed description for each object. The display program must display the proper level description or the blend between two descriptions. To accomplish this, a filter program, LOD, was created. LOD examines the viewing parameters and calculates the separation distance between the eye and each object in the display file. This information, along with the user specified transition zone
in the object descriptions is used to decide which level is appropriate. LOD outputs another display file which has the proper level descriptions referenced. This method of separating the level of detail management from the display program helps maintain the flexibility of the animation pipeline. LOD can be told to do blending over distance, blending over time, or instantaneous transitions (actually instantaneous transitioning is a special case of blending with a zero-width transition zone). In order to get the relative velocities of the objects, a movement vector is passed down to the display file. This information could be used for temporal antialiasing as well.
CHAPTER 7

Results and Conclusions

One fruit of this research is a simulation of a submarine docking maneuver (sponsored by NAVY contract N61339-81-C-0073) which uses level of detail management to maintain a constant scene complexity throughout the entire simulation.

A database of object description hierarchies was created. Most of the object descriptions were made by hand simplification of existing object descriptions using the object editor, INSPECT. The display programs to handle atmospheric perspective (haze) and texture mapping in perspective were developed. The software needed to stage the transitions and to do the blending of object descriptions was developed as described earlier.

Figure 14 shows the success of this sequence. The graph in Figure 14 represents visible polygon edge counts in each frame of the docking maneuver. The top line is for the simulation without the level of detail management
Figure 14. Edge Count in Docking Simulation

(all objects are rendered using the most detailed description). The bottom line represents the edge count of the simulation produced with the level of detail management. The "skyscrapers" in the lower line of the graph represent the increased work load when an object is undergoing a transition. Both the top and the bottom lines fall off
because the area containing objects is a thin strip along the coastline. Behind this strip, there are no objects in the database. In any object database, the object count will decline in views which near the edges of the modeled area.

As can be seen, the level of detail management produced a savings of more than 70% in edge count through more than half of the simulation. The total savings in edge count was about 74% for the entire simulation.

Without the level of detail management, this project could not have been completed in the time specified in the contract. Figure 15 shows two images in the docking simulation.

This research has shown that hierarchies of object descriptions are quite useful in reducing the time needed to compute an image, and more importantly, useful in raising the complexity of computable images. The major burden is still the generation of multiple descriptions of an object. Having a set of software tools such as an object editor and a description verification program make hand generation of object description hierarchies much simpler. Automated methods for simplifying object descriptions such as those proposed in Chapter 5 can make the simplification
Figure 15. Still frames from Docking Simulation
process almost trivial. The most detailed descriptions will still have to be modeled by hand using a data generation program or solid modeling system.

7.1. Further Research

One area which needs further research is the question of how the nature of the object effects the quality of automatically generated object description hierarchies. It was noted that grids and other regular objects do poorly in the small edge deletion simplification scheme. It is not clear what other factors effect the results. More work is also needed in refining the automated simplification and further data modification so that less user interaction is needed.

Having a linear hierarchy limited the display program to having to display each level at one single weighting. Facilities for allowing variable blending within the hierarchy to handle objects which are very large are needed. This might be accomplished with variable blending within a description, or by expanding the hierarchy to a finely branched tree structure. This expansion would need to be automated in order to be of any assistance, since it is currently possible to create more complex hierarchies using object instancing facilities.
List of References


APPENDIX A

INSPECT User's Manual

A.1 Introduction

INSPECT is an object editor which allows viewing and manipulating object definitions in the standard CGRG binary file format. It is intended to be run using one of the vector display devices, but can be run on an ASCII terminal alone by sacrificing the vector display. It provides a method of changing vertex locations, face definitions, vertex colors, face colors, and texture coordinates.

When using INSPECT, there are a few internal states of which the user must be aware. First, there is the notion of the current face and the current vertex. When INSPECT is first entered, the current face is face 1 and the current vertex is vertex 1. There are commands which change the current face/vertex. See the command summaries below.

The second internal state determines face or vertex mode. When in face mode, the current face is displayed and available for modification. Similarly, the current vertex can only be displayed or modified when in vertex mode. Again, there are commands which switch between face mode and vertex mode.

The final internal state determines zoom or where mode. When in where mode the current face/vertex is highlighted on a vector display of the entire object. Zoom mode gives a view of the current face/vertex and its neighboring faces/vertices. There are commands which switch between where and zoom mode.

Along with the vector display of the object, there is a brief description of the current face/vertex printed out on the ASCII terminal from which the editor is being invoked. This information will include the face/vertex
number (the vertex and face dictionaries are each numbered sequentially starting at 1), and other information about neighbors, coloration, texture coordinates, and location.

A.2 Subsystems in INSPECT

INSPECT has four major subsystems. They are identified by their prompts: Inspect>, Knob>, Edit>, and Color-mod>. The editor starts in Inspect> subsystem, but can be made to enter any of the other subsystems by using the appropriate command.

In the Inspect> subsystem, the user can add, copy, delete, and move vertices and add, break-up, copy, delete, join, move, reverse, and modify texture coordinates of faces. There are also commands to write out changes made to an object, read in an object, show the status of an object, and enter another subsystem.

The Knob> subsystem changes the orientation of the object relative to the eyepoint. This new orientation is maintained until it is again changed by the user in the subsystem.

The Colormod> subsystem is used to modify the face/vertex coloration files associated with an object. This subsystem maintains a color mnemonic dictionary for associating color names with color coordinates (red, green, blue, and transparency). The color coordinates of each face/vertex can be modified individually, or a series of expressions containing assignment and conditional phrases can be defined which globally modify the coloration of an object.

The Edit> subsystem is used to easily rearrange a face definition. New vertices can be added along edges of the face, or at any location within the plane of the face.

Commands in INSPECT are generally one letter long. Commands are entered from the terminal in response to the current prompt. No <cr> (carriage return) is needed after entering a one letter command. Many commands request a face/vertex number, color coordinate, etc. A <cr> is needed after entering such information.

There is a consistent help facility built into INSPECT. At any time, to any prompt, the user may enter either 'h' or '?' and get help on what responses are appropriate. This facility helps the naive user quickly learn how to use the editor.
Each subsystem has somewhat different bindings of the keys to the commands executed (e.g. 'r' in Knob> means rotate, while in Inspect> it means reverse). The command bindings of each subsystem are discussed below.

A.3 Inspect> Subsystem Commands

a - add a face/vertex

If in face mode, INSPECT will ask the user to enter a new face definition. This is done by first entering the number of vertices desired in the new face and then entering that many vertex numbers. The user may exit this mode early by entering a <cr> when asked "How many vertices?" The new face defined is added at the end of the face dictionary and becomes the current face.

If in vertex mode, INSPECT will ask the user for the x, y, and z coordinates of the new vertex. INSPECT will prompt the user for each axis individually, but the user can enter all three values at the "X -" prompt. The new vertex is added at the end of the vertex dictionary and becomes the current vertex.

b - break up current face

If in face mode, the user will be queried for two vertices in the face definition which define a new edge to be used to split the face. If the vertices supplied by the user define an edge which is already in the definition, the command is aborted. The coloration of the face/vertices is duplicated in each of the two new faces created. The new faces are added at the end of the face dictionary, and the last one becomes the current face. The face which was broken in two is deleted.

If in vertex mode, the user will get an error message: "Select a face first."
c - copy current face/vertex

If in face mode, a new face with the same definition as the current face is added to the end of the face dictionary. This new face becomes the current face.

If in vertex mode, a new vertex with the same location as the current vertex is added to the end of the vertex dictionary. This new vertex becomes the current vertex.

d - delete current face/vertex

If in face mode, the current face is deleted. It remains the current face and is drawn in with a dotted line on the vector display. A deleted face will not be considered a neighbor of nearby faces, although nearby normal faces will be considered neighbors of the current (deleted) face. It can be undeleted using the 'u' command.

If in vertex mode, the current vertex is deleted if and only if it is not referenced in any face. If a deleted face which references this vertex is later undeleted, this vertex will be automatically undeleted. Deleted vertices can also be undeleted using the 'u' command. The deleted vertex remains the current vertex.

e - edit current face

If in face mode, this command enters the Edit> subsystem. See the Edit> Subsystem Commands section for a full description of the key bindings in that subsystem.

If in vertex mode, the user will get an error message: "Select a face first."

f - face display

This command queries the user for a face number. The selected face becomes the current face. If a <cr> is
entered, the current face is redisplayed. This is useful for getting the terminal description back after it has scrolled off the screen. When a face has many vertices in its definition, the description will not all fit on the terminal screen. The definition is displayed a page at a time. The user must hit any key to get the next page. If the user hits 'q', the rest of the description is skipped. This command switches to face mode if in vertex mode.

h - help

INSPECT has a built in help facility. At any time the user may enter 'h' or '?' and get some help. At this level, the user gets a menu of all the legal commands and their meanings (very brief).

i - input new object

This command is used to input a new object. If any changes were made to the original object, the user is queried about writing out the changes.

j - join current face with one of its neighbors

If in face mode, the user is queried for the number of one the faces neighboring the current face. Then, the current face and the selected neighbor are coalesced across their common edge. This new face is added at the end of the face dictionary and becomes the current face. The two faces which were used to create the new face are deleted. If the new face is nonplanar or concave, this fact is noted. This command can be undone by undeleting the two faces used to create the new face, and deleting the newly created face.

If in vertex mode, the user will get an error message: "Select a face first."
k - knob the vector display

This command allows the user to interactively change the view transformation of the vector display. See the Knob> Subsystem Commands section for a description of the key bindings in this subsystem.

l - list faces/vertices

This command allows the user to enter a conditional expression. If in face mode, the face ID number of each face which satisfies the expression is listed. If in vertex mode, the ID number of all vertices which satisfy the expression is listed. See the Expression Syntax section for a description of how to specify conditional expressions.

m - move the current face/vertex

This command is really a super set of the Knob> subsystem. The object may be rotated, scaled, and placed as in the Knob> subsystem. In addition, the current face/vertex can be moved relative to the rest of the object. The 'm' key selects move face/vertex mode while 'r', 'p', and 's' select rotate, place, and scale mode for the whole object as in Knob> subsystem. The 'x', 'y', and 'z' commands apply the selected transform along those axes. The 'n' command applies the move face/vertex transform along the face/vertex normal when in 'm' (move face/vertex) mode.

If in vertex mode, the user is queried for exact coordinate values, if known. If none are given, then the interactive mode is entered. CAUTION: There is no "undo" command for any changes made after exiting the Move> subsystem. While still in Move> subsystem, the user may hit 'o' while in 'm' (move face/vertex) mode to get the original placement of the current face/vertex.
n - next face/vertex

This command increments the current face/vertex pointer. If at the last face/vertex, the first face/vertex becomes the current one.

o - output object description

This command is used to output a modified object description to a new file name. If no changes have been made to the object, a new copy cannot be written. The 'q' command is used to write changes made to the original files.

p - previous face/vertex

This command decrements the current face/vertex pointer. If at the first face/vertex, the last face/vertex becomes the current one.

q - quit and write out changes

This command exits INSPECT and asks the user about writing out changes made. It checks to make sure that all files have write permission, before writing.

r - reverse current face

If in face mode, this command causes the current face description's vertex order to be reversed.

If in vertex mode, the user will get an error message: "Select a face first."

s - show current status

This command lists out some information about the object, including the number of vertices, faces, edges, the bounding box, the names of the binary
files, and whether the definition has been modified. Also listed is the color of the object, if any.

t - texture coordinate modification

If in face mode, this command allows the user to modify the texture coordinates associated with each vertex of a face description. The user must type in the U and V coordinates of each vertex. As in most cases, a <cr> in response to a query for a value keeps the value which was previously there.

If in vertex mode, the user will get an error message: "Select a face first."

u - Undelete the current face/vertex

This command undoes the 'd' command. If the current face/vertex is currently deleted, it is undeleted.

v - display a vertex

This command queries the user for a vertex number. The selected vertex becomes the current vertex. If a <cr> is entered, the current vertex is redisplayed. This command switches to vertex mode if in face mode.

w - enter where mode

This command switches to where mode. The entire object is drawn with the current face/vertex highlighted.

z - enter zoom mode

This command switches to zoom mode. If in face mode, only the current face and its undeleted neighboring faces are displayed. If in vertex mode, only undeleted faces which reference the current vertex are displayed.
\(<\text{esc}\> - \text{enter Colormod} > \text{subsystem}\)

This command switches from \text{Inspect} > \text{to Colormod} > \text{subsystem. See the Colormod} > \text{Subsystem Commands section for a description of the key bindings in this subsystem.}\n
\(! - \text{shell escape}\)

This command introduces a system command. The string typed from the '!' to the <cr> is passed to the C shell to be executed.

\(: - \text{long command}\)

There are a few miscellaneous commands which are not used too frequently. The colon introduces these commands. The string typed from the ':' to the <cr> is used to select the long command to be executed. The available commands are described fully in the \text{Long Commands section.}\n
\text{A.4 Colormod} > \text{Subsystem Commands}\n
\(a - \text{modify all face/vertex colors}\)

If in face mode, this command queries the user for a set of assignment phrases and conditional expressions which direct a global modification of the face colors. If in vertex mode, the global modification is done to the vertex colors.

\(c - \text{modify current face/vertex color}\)

If in face mode, this command queries the user for a new face color mnemonic or coordinate quadruple. If
in vertex mode, this command queries for a new vertex color.

d - define a color mnemonic

This command is used to define a color mnemonic in the color dictionary. The user will be queried for the mnemonic and the definition (color coordinate). If the mnemonic is already defined, the user is asked if the old definition is to be removed. The user also has the choice of changing all previous uses of that mnemonic to the new definition. This provides a convenient method of changing all references of one color to a different color; simply redefine the mnemonic for the old color. If a definition of a new mnemonic is already entered in the dictionary, the user is queried about entering the new mnemonic as a synonym, or quitting.

f - face display

This command has the same function as it does in Inspect>. The user is queried for a face number. This selected face becomes the current face, and face mode is entered.

h - help

This command lists all the Colormod> command bindings.

i - input a color dictionary file

This command reads in a color dictionary file. When INSPECT is first entered, it looks in the current directory for a file called .colors. If this file is not found, it looks for ~/.colors. These are the user's personal color dictionaries. If the user wants to input another color dictionary the 'i' command is used. The user is queried for the name of a
color dictionary file. The format of a color dictionary file is:

<mnemonic> <red> <green> <blue> <transparency>

for example:
red 1 0 0 0
green 0 1 0 0
blue 0 0 1 0
white 1 1 1 0
black 0 0 0 0
tred 1 0 0 1
tgreen 0 1 0 1
tblue 0 0 1 1

k - knob the vector display

This command allows the user to interactively change the view transformation of the vector display. See the Knob> Subsystem Commands section for a description of the key bindings in this subsystem.

l - list faces/vertices

This command allows the user to enter a conditional expression. If in face mode, the face ID number of each face which satisfies the expression is listed. If in vertex mode, the ID number of all vertices which satisfy the expression is listed. See the Expression Syntax section for a description of how to specify conditional expressions.

n - next face/vertex

This command increments the current face/vertex pointer. If at the last face/vertex, the first face/vertex becomes the current one.

o - output color dictionary file

This command is used to output the current color mnemonic dictionary to a color dictionary file. The user is queried for the name of the file to write to.
p  - previous face/vertex

This command decrements the current face/vertex pointer. If at the first face/vertex, the last face/vertex becomes the current one.

r  - reverse current face

If in face mode, this command causes the current face description's vertex order to be reversed.

If in vertex mode, the user will get an error message: "Select a face first."

s  - show current status

This command lists out some information about the object, including the number of vertices, faces, edges, the bounding box, the names of the binary files, and whether the definition has been modified. Also listed is the color of the object, if any. The color mnemonic dictionary is also listed.

v  - display a vertex

This command queries the user for a vertex number. The selected vertex becomes the current vertex. If a <cr> is entered, the current vertex is redisplayed. This command switches to vertex mode if in face mode.

w  - enter where mode

This command switches to where mode. The entire object is drawn with the current face/vertex highlighted.

z  - enter zoom mode

This command switches to zoom mode. If in face mode, only the current face and its undeleted neighboring
faces are displayed. If in vertex mode, only undeleted faces which reference the current vertex are displayed.

<esc> - enter Inspect subsystem

This command switches from Colormod to Inspect subsystem. See the Inspect Subsystem Commands section for a description of the key bindings in this subsystem.

! - shell escape

This command introduces a system command. The string typed from the '!' to the <cr> is passed to the C shell to be executed.

: - long command

There are a few miscellaneous commands which are not used too frequently. The colon introduces these commands. The string typed from the ':' to the <cr> is used to select the long command to be executed. The available commands are described fully in the section.

A.5 Long Commands

beep - turn on beeping

This command turns on casual beeping. See 'nobeep' in this section for a more detailed description.
crease - crease the current face

This command copies the current face, adding the new face to the end of the face dictionary. It also copies the individual vertices which define the new face. This causes a crease when the object is smooth shaded. The new face becomes the current face.

delete - delete faces using expressions

This command allows the user to enter a conditional expression which determines which faces in the object will be deleted. This is similar to the 'l' list command in Inspect and Colormod. See the Expression Syntax section for a full description of conditional expressions.

expunge - remove deleted faces

This command removes all deleted faces permanently. Once expunged, they cannot be undeleted. This is useful if you need to know what the polygon ordering will be when the object is written out.

hit - 2D hit routine

This command turns on the cursor and allows the user to select a position on the screen. All vertices which are near the cursor are listed. The closest vertex has all faces listed in which it is referenced.

knife - knife the object along a plane

This command allows the user to position the object beneath a knife using the Knob subsystem and slice the object into two parts. This command only works when using the Megatek display. Once split, the faces are reorganized so that edges on the positive side of the knife are first, edges on the negative side are second, and deleted faces (usually those
which were split by the knife) are last. This facilitates using :delete to remove one of the two halves generated by knifing the object.

markedge - turn on edge type distinctions

This command instructs the Megatek display to show the number of faces joining an edge by varying the edge type. Normal edges shared by two faces are displayed using the dim solid line. Edges referenced in only one face are displayed as bright dotted lines. Edges at which more than two faces join are displayed as bright solid. Deleted faces do not contribute to the count of contributing faces for an edge.

nobeep - turn off beeping

This turns off the terminal beeping for nonimportant things like deleted faces/vertices. If something important goes wrong (like a file write failure), the terminal will still beep. This is best used when most of the faces in an object have been deleted and the constant beeping when viewing deleted faces becomes bothersome.

nomarkedge - turn off edge type distinctions

This command turns off the edge distinction feature of the vector display. All edges which are referenced by nondeleted faces are displayed using solid dim lines.

undelete - undelete faces using expressions

This command allows the user to enter a conditional expression which determines which faces in the object will be undeleted. This is similar to the 'l' list command in Inspect> and Colormod>. See the Expression Syntax section for a full description of conditional expressions.
A.6 Edit>

The Edit> subsystem adds the concept of current edge. When Edit> is entered, the current face is displayed. The current edge of the face description is highlighted. The current vertex of the face description is boxed. Any nearby vertices which lie in the plane of the face are listed and displayed. This can be somewhat of an inconvenience when displaying 2D data, but there is currently no way of selectively turning them off.

a - add a vertex

This command queries the user for a vertex number. This vertex is added into the face definition along the current edge. There is no restriction about the chosen vertex lying in the plane of the face.

b - break up the face

The name of this command is somewhat of a misnomer. The function is really to define new faces using the displayed vertices. The user uses the cursor to define a new face. Each time the cursor is poked, the closest vertex to the cursor location is added to the new face definition. To end the new face definition, the user selects the first vertex again. The new face is then displayed with all of its neighbors. As usual, it is added to the end of the face dictionary. Once the user has looked at the new face and acknowledged by hitting <cr>, the original face description is redisplayed. In a sense, this command can be used in conjunction with 's' slide to carve a face into as many pieces as desired. Then, the face can be deleted once Edit> subsystem is exited.

c - copy current vertex

This command duplicates the coordinates of the current vertex and adds a new vertex to the end of the vertex dictionary. The new vertex is not added to the face definition, but easily can be using 'r' (replace).
d - delete current vertex

The current vertex is removed from the face definition. If there are only three vertices in the face, no vertices may be deleted.

f - flatten vertex

This command queries the user for a vertex number. This vertex is projected into the plane of the face. This is used mainly to flatten out slightly nonplanar faces. If a face is sufficiently nonplanar, some of the vertices in the definition will not pass the coplanarity test and will not be numbered on the vector display. Note that flattening one face may make a neighboring face nonplanar.

h - help

This command lists all the possible commands available in this subsystem along with a brief meaning for each.

k - enter Knob> subsystem

This allows the user to change the viewing parameters of the display.

l - list coplanar vertices

This command lists those vertices which pass the nearby, coplanarity test. These numbers are displayed on the screen, but sometimes when they overlap it is hard to read them.

m - move vertex

This command is used to move one of the coplanar vertices. The movement is done using the cursor. The
new position is constrained to lie in the plane of
the face. The point does not need to be part of the
face definition.

n - next edge/vertex

This command moves the current edge and vertex around
the face definition in a clockwise manner.

o - on/off edges

This command turns on/off the edges comprising the
face definition. This is used to allow a better view
of the vertex numbers.

p - previous edge/vertex

This command moves the current edge and vertex around
the face definition in a counterclockwise manner.

r - replace current vertex

The current vertex in the face description is
replaced with one that the user specifies. This is
an add and delete all in one.

s - slide new vertex

This command is used to slide a new vertex along a
line in the plane of the face. The user is queried
for two vertices in the plane of the face. The cur-
sor then slides along the line (not segment) which is
defined by the two selected vertices. When the user
selects a position, a new vertex is added at the end
of the vertex dictionary with the appropriate coordi-
nates. This new vertex is not added to the face
description.
! - shell escape

This command introduces a system command. The string typed from the '!' to the <cr> is passed to the C shell to be executed.

<esc> - return to Inspect> subsystem

This command changes from the Edit> subsystem to the Inspect> subsystem.

A.7 Knob>

Most of the commands in this subsystem are executed silently. The one line displayed gives the current transformation mode and the amount of each application of the transform.

d - negative transform
- - negative transform
_ - negative transform

These three keys all set the current vernier to a negative value.

i - identity matrix

This command sets the view matrix to the identity matrix.

o - original matrix

This command sets the view matrix to the value it had when the Knob> subsystem was entered.
p - placement transform
. t - placement transform

This command sets up a placement or translation transformation. Each time the axis keys are pressed, the current vernier for placement is concatenated to the view matrix of the object.

r - rotation transform

This command sets up a rotation transformation. Each time the axis keys are pressed, a rotation of the current rotation vernier is concatenated to the current view matrix.

s - scale transform

This command sets up a scale transformation. Each time the axis keys are pressed, the object is scaled by the current scale vernier. No single axis scaling is allowed, so all axes are scaled regardless of which axis key is pressed.

u - positive transform
+ - positive transform
= - positive transform

These three keys all set the current vernier to a positive value.

v - set vernier
V - set vernier

Set the current mode's vernier. If a <cr> is entered, the vernier is doubled for 'v' and halved for 'V'. Each of the three transforms have their own verniers.
x - apply transform along X axis
X - apply 10*transform along X axis

This key applies the current transform along the X axis. The capital X applies the transform 10 times.

y - apply transform along Y axis
Y - apply 10*transform along Y axis

This key applies the current transform along the Y axis. The capital Y applies the transform 10 times.

z - apply transform along Z axis
Z - apply 10*transform along Z axis

This key applies the current transform along the Z axis. The capital Z applies the transform 10 times. A Z translation is not visible since the object is viewed in an orthographic projection.

<esc> - return to previous subsystem

This command exits the Knob> subsystem.

A.8 Expression Syntax

Expressions in INSPECT are of two forms: a conditional phrase or an assignment phrase followed by a conditional phrase. The second form is used when the coloration of an object is going to be modified, while the simple conditional is used elsewhere.

A.9 Conditional Phrases

A conditional phrase is a series of simple conditional expressions joined by the logical operators && (AND) and || (OR). A simple conditional is of the form <variable> <relational operator> <literal> or <variable> <relational operator> <variable>. The valid variables are 'r' for red, 'g' for green, 'b' for blue, 't' for transparency, 'n' for face/vertex number, 'm' for face/vertex
number modulo 2, and 'c' for color mnemonic. The relational operators are '==' (EQUALS), '!=' (NOT EQUALS), '>' (GREATER THAN), '=>' (GREATER THAN OR EQUAL), '<' (LESS THAN), or '<=' (LESS THAN OR EQUAL). Literals are real numbers or integers. Colors are reals in the range 0.0 to 1.0, but face/vertex numbers are integers in a much larger range. Simple conditionals are evaluated left to right. There is no parenthesis facility to rearrange the order of evaluation. Blanks are ignored, so they may be used to improve readability.

The best way to explain these expressions is to look at a few examples.

Example 1

\[ r = l && g == 0 && b == 0 && t == 0 \]

This conditional phrase evaluates true only for faces/vertices whose color is pure red. A similar method would be the conditional phrase \[ c == \text{red} \] (provided the mnemonic red is properly defined).

Example 2

\[ n >= 50 \]

This conditional phrase evaluates true only for faces/vertices whose identification number is greater than or equal to 50. This type of conditional can be used in the ':delete' command to remove large portions of an object.

Example 3

\[ m == 0 \]

This conditional phrase evaluates true only for faces/vertices whose identification number is even. \[ 'm' \] is the face/vertex number modulo 2. If the ID number is even, \[ 'm' \] is zero. If the ID number is odd, \[ 'm' \] is one. This type of conditional phrase is useful for making checkerboards.
Example 4

""

The double quote is a special character used to mean the last conditional phrase used. This can eliminate some typing when using a conditional phrase over again.

A.10 Assignment Phrases

Assignment phrases are used to assign a value to the color coordinates of a face/vertex. The form of an assignment phrase is <arithmetic operator> <literal> or <arithmetic operator> <variable>. The arithmetic operators are: '=' (BECOMES), '+=' (INCREASED BY), '-=' (REDUced BY), '*=' (MULtiplied BY), '/=' (DIVided BY). Literals are real numbers in the range 0.0 to 1.0. Variables are 'r', 'g', 'b', and 't'. The variables 'n' and 'm' are also available, but not very useful since colors are not allowed to exceed one or fall below zero.

The assignment phrase is separated from its associated conditional phrase by a ':'. The assignment is done only if the conditional phrase is true. If no conditional phrase is present, the assignment is done. Again, a few examples will be helpful.

Example 5

Red Expression - r = 0

This assigns the value zero to the red value of the color. When prompting for an assignment expression, INSPECT will provide the variable to which the value will be assigned (e.g. Red expression - r).
Example 6

Red Expression - $r \times 2 : r < 0.5$

This multiplies the red value by 2 if its value is less than 0.5

Example 7

Red Expression - $r = 0 : m = 0$
Green Expression - $g = 0$
Blue Expression - $b = 1$
Transparency Expression - $t = 0$

This set of expressions sets the color to blue on all faces/vertices which have an even ID number.