COMPUTER RECOGNITION OF PARTIAL VIEWS OF THREE-DIMENSIONAL CURVED OBJECTS*

by

J. W. McKee and J. K. Aggarwal
Department of Electrical Engineering

Technical Report No. 171
May 15, 1975

INFORMATION SYSTEMS RESEARCH LABORATORY

ELECTRONICS RESEARCH CENTER

THE UNIVERSITY OF TEXAS AT AUSTIN

Austin, Texas 78712

*Research supported by the Joint Services Electronics Program under JSEP Contract F44620-71-C-0091 and by the National Science Foundation under Grant GK-42790

ABSTRACT

This paper describes two similar algorithms: one for learning descriptions of objects, the other for recognizing partial views of known objects. The input to either algorithm is a two-dimensional array in which the boundaries of the surfaces of the object are marked. Both algorithms use a library of known objects. The learning algorithm sees the whole object to be learned and computes a description of the object. The output of the learning program is an updated library of known objects in which the description and name of the new object have been added to the names and descriptions of all the other known objects in the library. The recognition algorithm sees only a partial view of a known object. The algorithm computes a description of the partial view and compares this description to the descriptions of known objects. The output of the recognition algorithm is a list of possible names of the partial view with a measure of how well each object named matches the partial view. Figures illustrating the descriptions of the objects in the library of known objects are given. To help the reader evaluate the performance of the algorithm, figures of partial views are given. These figures illustrate how the recognition algorithm matched the partial view of an object to different known objects. A table summarizing the results of tests of the recognition algorithm is given.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES
(continued)

LIST OF FIGURES
(continued)

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This paper is concerned with some of the aspects of the general problem of recognizing an object from its gray scale image. One approach to analyzing a scene, often called a bottom-up method, is to process the gray scale information in a series of steps: boundary detection, object description and model matching. Two of the more common methods of boundary detection are edge marking and tracing, Roberts [1] and Underwood [2], and region growing, Brice and Fennema [3] and Barrow and Poplestone [4]. Object descriptions seem to be divided into two classes, depending on whether all the edges in the scene are assumed to be straight lines or not. If they are assumed to be straight lines, then the object is described by relations between the straight lines, [1], [2], [3]. If the edges may be curved, then other descriptions are used [4] such as different moments of the surfaces and their spatial relationships. Depending on the type of descriptors used, the description of the scene is "matched" to a library of descriptions of known objects.

This paper will deal with the last two steps of the bottom-up method: object description and model matching. The input data for the object description program is a two-dimensional edge array in which all the boundaries of the surfaces are marked. This edge array is generated by another algorithm, described in McKee and Aggarwal [5], which converts a gray scale picture into the two-dimensional array in which the boundaries are marked. The objects viewed are not restricted to be planar; in fact, most of the objects used are curved in two or more dimensions. This curvature and uneven lighting introduce small "noise surfaces" and false edges in the input data which must be contended with.

This paper will describe two very similar algorithms: one for computing and learning the description of an object, the other for computing and comparing a description of a partial view of an object to the descriptions of known objects. A library of known objects is constructed by the learning algorithm. The learning algorithm sees the whole object to be learned, computes a description of the view, and adds the description to a library of known objects. The learning algorithm does not compute a three-dimensional description of the object but forms a description of the object from the two-dimensional projection it "sees". At the present time only one view of each object is being learned. The recognition algorithm uses the library of known objects and a partial view of a known object to derive a list of names of objects which could cause the partial view. The recognition algorithm also computes a measure of how well each named object matches the partial view. The partial view is a part of the same projection of the object used in learning the object.

CHAPTER 2

DEFINITIONS

Before discussing the algorithms, we should define some terms used in the rest of the paper. The definitions of chain-connected points and of nodes are found in [5]. We will give only a general definition here. A set of points is chain-connected if they form a chain in which each point has only two neighbors. A node is a set of points that have three or more neighbors.

(i) The perimeter of a scene is a union of sets of chain-connected points which forms a window about a portion of the scene. Only the interior of the perimeter is considered to be visible to the algorithm.

(ii) Types of Nodes.

A) A perimeter node is a node containing points on the perimeter of the scene.

B) A real node is a node common to three or more surfaces, but not containing points on the perimeter of the scene.

C) A false node is any node that is not a perimeter node or a real node, e.g., a node common to only two surfaces.

(iii) Types of Edges.

A) In the context of this paper an edge is a set of chain-connected points which forms the complete boundary between two surfaces.

B) An edge segment is a set of chain-connected points between two nodes. An edge contains one or more edge segments.

Let us illustrate these definitions by pointing them out in Fig. 1. The perimeter is the rectangular outside set of points. Two perimeter nodes, two real nodes, and a false node are pointed out in Fig. 1.

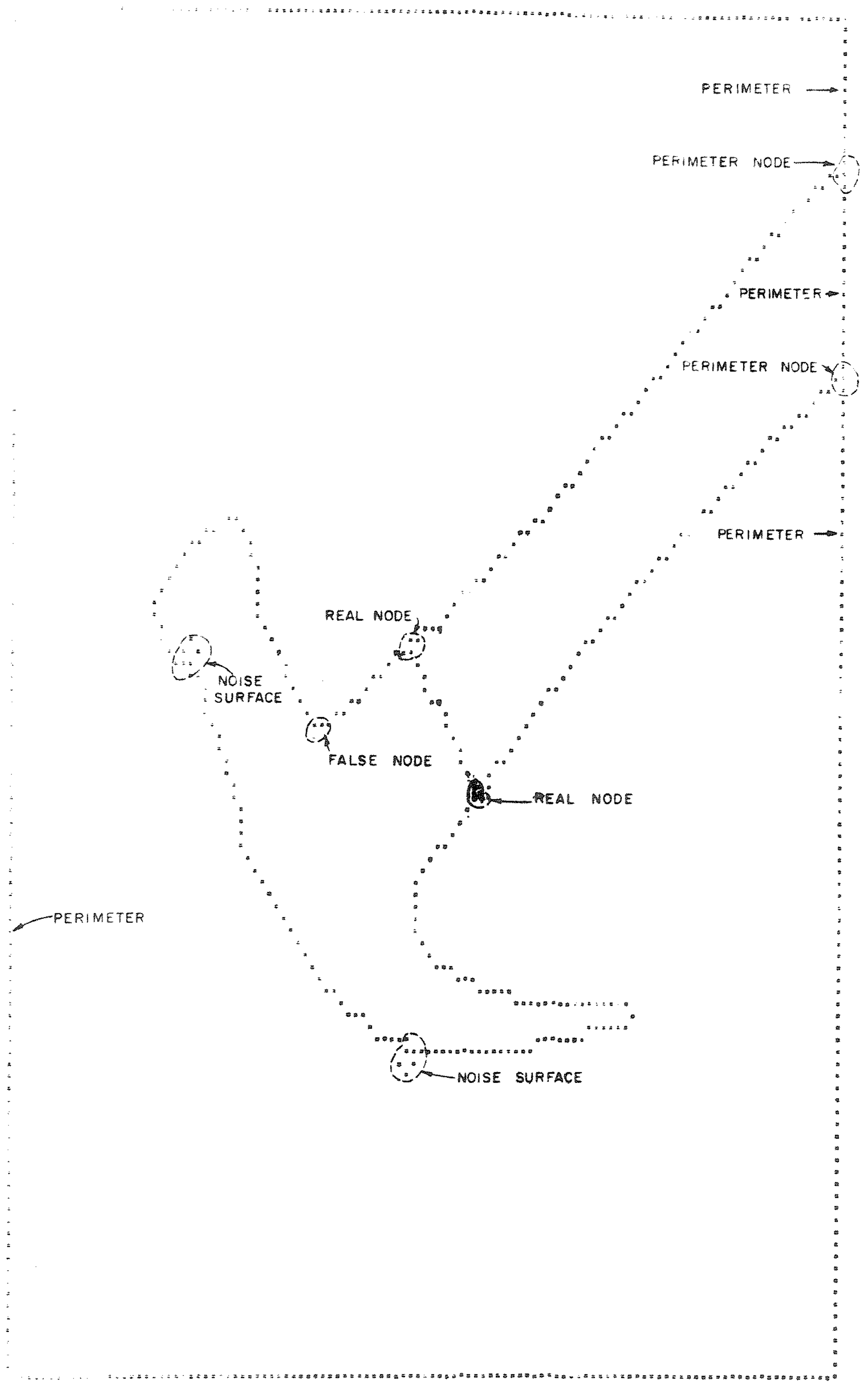


Figure 1. Partial View of a Hammer

Surfaces like the two noise surfaces shown in Fig. 1 are caused by glare or shadows. After these noise surfaces and the false node are removed and the gaps are filled with chain-connected points, the edge between the hammer head and the background would be the set of chain-connected points from one real node to the other real node. The edge segments are the sets of chain-connected points between nodes.

CHAPTER 3

THE ALGORITHMS

There are actually two closely related algorithms: the first is to learn a description and name of a viewed object, the second is to form and compare a description of a partial view of an object to the set of descriptions of known objects. Each algorithm can be divided into three major steps. The first two steps of the learning algorithm are the same as the first two steps of the recognition algorithm. The last step of the learning algorithm is similar to the last step of the recognition algorithm. Therefore, we treat learning and recognition as one algorithm with the following steps:

A) For each edge segment in the scene, create a list of the x-y coordinates of the points on the edge segment; and for each edge,create a list of the edge segments which form the edge.

B) Describe each edge in terms of a series of arcs of circles, and

CL) Learn the name and description of the object, or

CR) List the names of known objects which could have caused the partial view.

Before we give a more detailed discussion of the steps of the algorithm, let us describe the data structure of the input to the algorithm. The algorithm is designed to handle the output of the algorithm described in [5]. The input data is in the form of a two-dimensional rectangular array in which the boundaries of the surfaces of the object are marked. There may be edge segments due to shadows and edge segments connected to false nodes. There may also be surfaces caused by noise or glare on the object. These problems are shown in Fig. 1 and will be corrected in Step B. For each edge in the scene, Step A and Step B are performed.

This procedure builds a description of the edges that appear in the scene. Then either Step CL or Step CR is executed. The following is a general discussion of each step.

Step A - Edge segment coordinate testing - The boundaries of the object are composed of edge segments which connect different types of nodes. In this step the edge segments are found by examining the boundaries of the surfaces. The surfaces in the scene are numbered sequentially starting with 1. When new edge segments are encountered as a boundary is being traced, they are numbered sequentially. The following information about an edge segment is placed in the edge segment library under the number of the edge segment: the number of the surface being traced, the number of points in the edge segment, and the space which is reserved for another surface number. The x-y coordinates of the points of the edge segment are placed on the x-y coordinate list under the number of the edge segment. The edge segment is marked as being known. If a known-edge segment is encountered while a boundary is being traced, the number of the surface being traced is placed in the reserved location in the edge segment library under the number of the edge segment. As the boundary of each surface is traced, the numbers of the edge segments which form the boundary are placed in an edge segment list under the number of the surface.

After all the boundaries have been traced, edges are found by examining the edge segment lists of the surfaces. As the new edges are found, they are numbered sequentially starting with 1. An edge is represented by a list of all the sequential edge segments on a boundary which have the same two surface numbers. This list of edge segments is placed in an edge list under the number of the edge. At the completion of Step A the edges in the scene are described by ordered lists of edge segments. Each edge segment is described by its length, a list of the x-y coordinates of the chain-connected points of the edge segment, and the number of the surface on each side of the edge segment. The lists formed are passed to Step B. A detailed discussion of this step and its data structure is given in Appendix A.

Step B - Edge description - For each edge from Step A, a list of the x-y coordinates of the points on the edge is created from the x-y coordinates of the points on the edge segments which form the edge. This list of coordinates is converted into a type of Freeman code, Freeman [6], and straight lines are fitted to the graph of the points of the code vs. distance. The following procedure is done for each edge on the edge list. The x-y coordinates of the first edge segment on the list for the edge are placed on the list of coordinates of the edge. If there is another edge segment on the list, points approximating a straight line between the end of the first edge segment and the start of the second edge segment are placed on the coordinate list of the edge. Then the x-y coordinates of the second edge are placed on the coordinate list of the edge. This process is repeated if there are still more edge segments on the list for the edge. The result is a list of the x-y coordinates of the points of the edge. This list of coordinates is converted into a Freeman code for the edge. The code is transformed in a series of steps into a smoothed-compensated-expanded-Freeman code, SCEFC as described in Appendix B. A series of straight lines is then fitted to the SCEFC data points vs. distance graph. These straight lines form the description of the edge. All the edges are described in this manner and their descriptions are passed to the next step. A detailed discussion of this step and its data structure is given in Appendix B.

Step CL - learning - The learning algorithm reads in from a permanent file a library of known objects and a library of known edges. The name for the object to be learned is also read in. The following steps are performed for each edge from Step B. The description of the new edge is compared to the description of each edge in the library of known edges. If the description of the new edge and the description of an edge in the library are "similar", the name of the edge in the library is given to the new edge. If none of the known edge descriptions is "similar" to the description of the new edge, the new edge is given a name and its description is placed in the library of known edges. In either case, the new edge has a name when this portion of the algorithm is complete. After all the edges from Step B have been named, a description of the object is formed. The name of the object, along with the name of each of its edges and some information about each edge, is placed in the library of known objects. The library of known objects and the library of known edges replace the old libraries on the permanent file and the program ends. A detailed discussion of this step and its data structure is given in Appendix C under the name CL.

Step CR - recognizing - The recognition algorithm reads in a library of known objects and a library of known edges from a permanent file. The following steps are performed for each unknown edge from Step B. The "weight" of the new edge, which is proportional to the length of the edge, is computed and stored under the number of the unknown edge. The description of the new edge is compared to the description of each edge in the library of known edges. A list of the names of the known edges which are "similar" to the new edge is formed. The names on this list are compared to the names of edges in the description of objects in the library of known objects. For each object, the name is placed on a list of edges common to both the partial view and the object. After all the edges from Step B have been processed, each object will have a list of all the edges which are common between the object and the partial view. The edges on this list are used to compute a scale factor between the object and the partial view. The scale factor for an object is a "weighted" average of the ratios of the length of an edge in the partial view to the length of the corresponding edge in the object's description, for all the edges that are common between the object and the partial view. When only a portion of an edge is present in the partial view, the total length of the edge is estimated. This estimate is used in the ratio computation. After the scale factor has been computed, a weight for the object is formed. Initially the weight of each object is zero. For each edge that is common between the object's description and the partial view, a ratio of the lengths is computed, as above. If this ratio is within a fixed percentage of the scale factor computed for the object, the "weight" of the corresponding unknown edge is added to the weight of the object to form the new weight of the object. At the completion of the above procedure, the object's weight is proportional to the number of edges in the partial view that maintained their relative lengths when matched to edges in the object's description. In this fashion a weight is computed for each known object. The names, normalized weights, and information about how the edges matched for the best three objects are outputted and the program ends. A detailed discussion of this step and its data structure is given in Appendix C under the name CR.

CHAPTER 4

EXAMPLES

In this section, examples illustrating the performance of the two algorithms will be given. There are nine objects in the library of known objects: a block with a corner cut off, an oil can, a coffee mug, a tape dispenser, a pair of scissors, a pair of pliers, a can, a glue bottle, and a hammer. First we will describe the procedure by which the objects were learned. Each object was placed before the image dissector camera, and a 128 x 128 point gray scale image of the scene was placed on tape. Before any objects were learned, an empty known-object library and an empty known-edge library were created on a library tape. Then for each object, its gray scale information was processed by the boundary-marking algorithm and then by the learning algorithm. The libraries from the library tape were read in, and a name for the object was read in. After the algorithm learned the edges of the object and learned the object, the new libraries were written over the old libraries. This process builds on what it had learned previously until all nine of the objects have been learned. After the nine objects have been learned, the library of known edges consisted of a set of distinct edges. For purposes of display, each edge was given a distinct character to be used in printouts, starting with 1 and going through 9, then A through Z. To get a visual feel for how the objects have been described, the gray scale images on tape, which were used to learn the object, were used as the input to the recognition algorithm. Besides the possible name and weight of an object, a two-dimensional printout of matched edges is given. The printout is in the form of the input edge array of the scene, but instead of points, the symbols for the names of the edge are used. The normalized weight of the object is expressed in octal with a maximum of 1750 octal (1000 decimal). For example, Fig. 2 is the input edge array for the block and Fig. 3 is a printout in which the edges are represented by the symbols for the names. As discussed in Appendix C, an edge may have one or two names. If the edge has a second name, it is represented by one symbol for the name typed on the printout with an arrow to the corresponding edge. In the recognition program, if an edge is considered too short to be useful, no attempt is made to recognize the edge. Unrecognized edges are marked with zeros (0). Figure 2, Fig. 4, Fig. 6, etc. through Fig. 18, are the input edge arrays generated from the gray scale information used to learn the objects. Figure 3, Fig. 5, etc., through Fig. 19, are the corresponding output from the recognition algorithm. For each of the nine known objects, Table 1 gives the name of the known object and the names and normalized weights of the three objects which best matched the scene. Table 1 also gives the total CPU time from camera input to the output of the new library for the learning program, and the total CPU time from tape input to printer output for the recognition program.

In the recognition portion of the testing of the algorithm, various partial views of the known objects were processed. The camera was aimed so that a partial view of the object was seen. The boundary-marking algorithm and the recognition algorithm were then run. The output consisted of the names, weights, and printouts of the best three matches to the partial view. Forty-three different views, other than the library view, of the known objects were processed. In 40 of these scenes, the algorithm computed the correct names for the objects. There were two ties, in which one or two objects had the same weight as the correct object (Fig. 23 and Fig. 24; Fig. 52, Fig. 53, and Fig. 54). There was one scene in which the algorithm made an error by computing a weight for an incorrect object that was greater than the weight for the correct object (Fig. 23 and Fig. 24). Due to the lack of space, it is not possible to present all the printouts; therefore, only the more interesting outputs are shown. As discussed in Appendix C, if an unknown edge has only one real node and is relatively straight, i.e., it does not curve by at least 45°, no attempt is made to match the edge to known edges. Similarly, if an edge has two perimeter nodes and does not curve by at least 90°, no attempt is made to use the edge. These types of edges are not used in the recognition program because usually with so little curvature they would match many known edges, and there is not enough information in the edge to get a good estimate of its total length. In the printouts, these edges, along with unrecognizable edges, are represented by zeros (0). Table 2 gives a summary of the tests run. For each object, the figure numbers of the partial views of the object are given. For each view, the names of the three best matches and their weight are given. And the total run time from camera input to printer output is given. The types and numbers of the edges in the scene are also given, so that the reader may compare the number and the type of the edges described and named to the total CPU time and accuracy of the comparisons. An edge is classified into one of three types, depending on whether it has two real nodes, one real node and one perimeter node, or two perimeter nodes. Figures 20 through 55 are given to enable the reader to evaluate the performance of the algorithm on partial views.

TABLE 1

LIBRARY OBJECTS

Object Viewed	Figure Numbers		Total CPU Time *		Best Name and Weight	Second Name and Weight	Third Name and Weight
	Edge Array	Library Object	To Learn Object in Seconds	To Recognize Object in Seconds			
Block	2	3	12	19	Block/1750	Pair of Pliers/350	Coffee Mug/330
Oil Can	4	5	10	14	Oil Can/1750	Can/707	Scissor/707
Pair of Pliers	6	7	10	22	Pair of Pliers/1750	Scissors/1270	Can/526
Coffee Mug	8	9	13	21	Coffee Mug/1750	Block/176	Hammer/172
Tape Dispenser	10	11	12	15	Tape Dispenser/1750	Coffee Mug/475	Hammer/41
Glue Bottle	12	13	10	12	Glue Bottle/1750	-	-
Scissors	14	15	32	32	Scissors/1750	Pair of Pliers/513	Tape Dispenser/311
Hammer	16	17	14	14	Hammer/1750	Can/1217	Scissor/1217
Can	18	19	9	13	Can/1750	Pair of Pliers/1267	Tape Dispenser/460

*On an XDS-930 Computer.

TABLE 2

PARTIAL VIEWS OF KNOWN OBJECTS

Object Viewed	Figure No.(s)	Type & Number of Edges: 2 Real Nodes	Type & Number of Edges: 1 Real Node 1 Perimeter Node	Type & Number of Edges: 2 Perimeter Nodes	Total CPU Time in Seconds*	Best Name and Weight	Second Name and Weight	Third Name and Weight
Block	20	4	2	0	42	Block/1301	Can/206	Pair of Pliers/206
	-	8	2	0	26	Block/1055	Coffee Mug/351	Can/130
	-	9	0	0	26	Block/1632	Pair of Pliers/341	Can/125
Oil Can	21	4	2	0	17	Oil Can/1305	Hammer/126	Scissors/126
	22	4	2	0	36	Oil Can/1403	Can/1037	Scissors/1037
	23,24	4	2	0	17	Pair of Pliers/1361	Oil Can/1353	Can/1255
	-	5	0	0	19	Oil Can/1601	Pair of Pliers/764	Can/717
Pair of Pliers	25	3	2	0	41	Pair of Pliers/1661	Hammer/751	Scissors/751
	26	1	2	0	38	Pair of Pliers/1750	Hammer/1424	Can/1345
	27	3	2	0	35	Pair of Pliers/1564	Scissors/1037	Oil Can/1037
	28	1	4	0	47	Pair of Pliers/1350	Oil Can/205	-
	29,30	3	2	0	22	Pair of Pliers/1565	Scissors/1565	Hammer/700
	-	4	0	0	23	Pair of Pliers/1750	Scissors/1303	Hammer/542
Coffee Mug	31	2	1	1	54	Coffee Mug/1716	Hammer/777	Block/717
	32	1	2	0	47	Coffee Mug/1667	Scissors/1126	Tape Dispenser/1126
	33	1	3	0	47	Coffee Mug/1343	Scissors/715	Tape Dispenser/715
	34	3	2	0	73	Coffee Mug/1674	Oil Can/252	Hammer/226
	35	2	2	1	124	Coffee Mug/1664	Pair of Pliers/601	Scissors/367
	36	3	2	0	27	Coffee Mug/1353	Oil Can/305	Block/305
	-	3	2	0	31	Coffee Mug/1576	Oil Can/275	Block/275
	-	4	0	0	26	Coffee Mug/1750	Block/427	Oil Can/217

TABLE 2 - continued

Object Viewed	Figure No.(s)	Type & Number of Edges: 2 Real Nodes	Type & Number of Edges: 1 Real Node 1 Perimeter Node	Type & Number of Edges: 2 Perimeter Nodes	Total CPU Time in Seconds*	Best Name and Weight	Second Name and Weight	Third Name and Weight
Tape Dispenser	37	3	2	0	51	Tape Dispenser/1750	Scissors/670	Coffee Mug/670
	38	3	2	0	58	Tape Dispenser/1664	Oil Can/740	Scissors/735
	39	2	2	0	50	Tape Dispenser/1335	Hammer/1163	Glue Bottle/370
	40	3	2	0	28	Tape Dispenser/1750	Scissors/766	Coffee Mug/766
	-	4	0	0	25	Tape Dispenser/1750	Scissors/476	Coffee Mug/476
Glue Bottle	41	2	1	0	25	Glue Bottle/1320	Pair of Pliers/125	Can/124
	42	1	2	0	47	Glue Bottle/743	Pair of Pliers/652	Block/374
	-	3	0	0	18	Glue Bottle/1713	Scissors/346	Pair of Pliers/346
Scissors	43	1	0	2	605	Scissors/1750	Tape Dispenser/1331	Coffee Mug/1331
	44	5	4	2	264	Scissors/1425	Pair of Pliers/571	Can/566
	45	3	2	0	52	Scissors/1726	Pair of Pliers/1076	Tape Dispenser/611
	46	5	4	0	42	Scissors/1471	Tape Dispenser/375	Coffee Mug/375
	-	8	0	0	48	Scissors/1740	Pair of Pliers/474	Tape Dispenser/323
Hammer	47	2	2	0	57	Hammer/1714	Oil Can/1512	Pair of Pliers/1462
	48	1	4	0	36	Hammer/710	Coffee Mug/220	Scissors/54
	49	1	4	0	57	Hammer/1267	Oil Can/564	Can/526
	50	1	4	0	40	Hammer/1301	Scissors/107	Glue Bottle/107
	51	2	2	0	19	Hammer/1277	Scissors/46	Glue Bottle/46
	-	3	0	0	20	Hammer/1750	Oil Can/1221	Scissors/1175
Can	52,53, 54	0	3	0	24	Can/1450	Hammer/1450	Scissors/1450
	55	2	0	0	17	Can/1750	Pair of Pliers/1457	Glue Bottle/270
	-	3	0	0	17	Can/1750	Oil Can/1523	Pair of Pliers/1272

*On an XDS-930 Computer

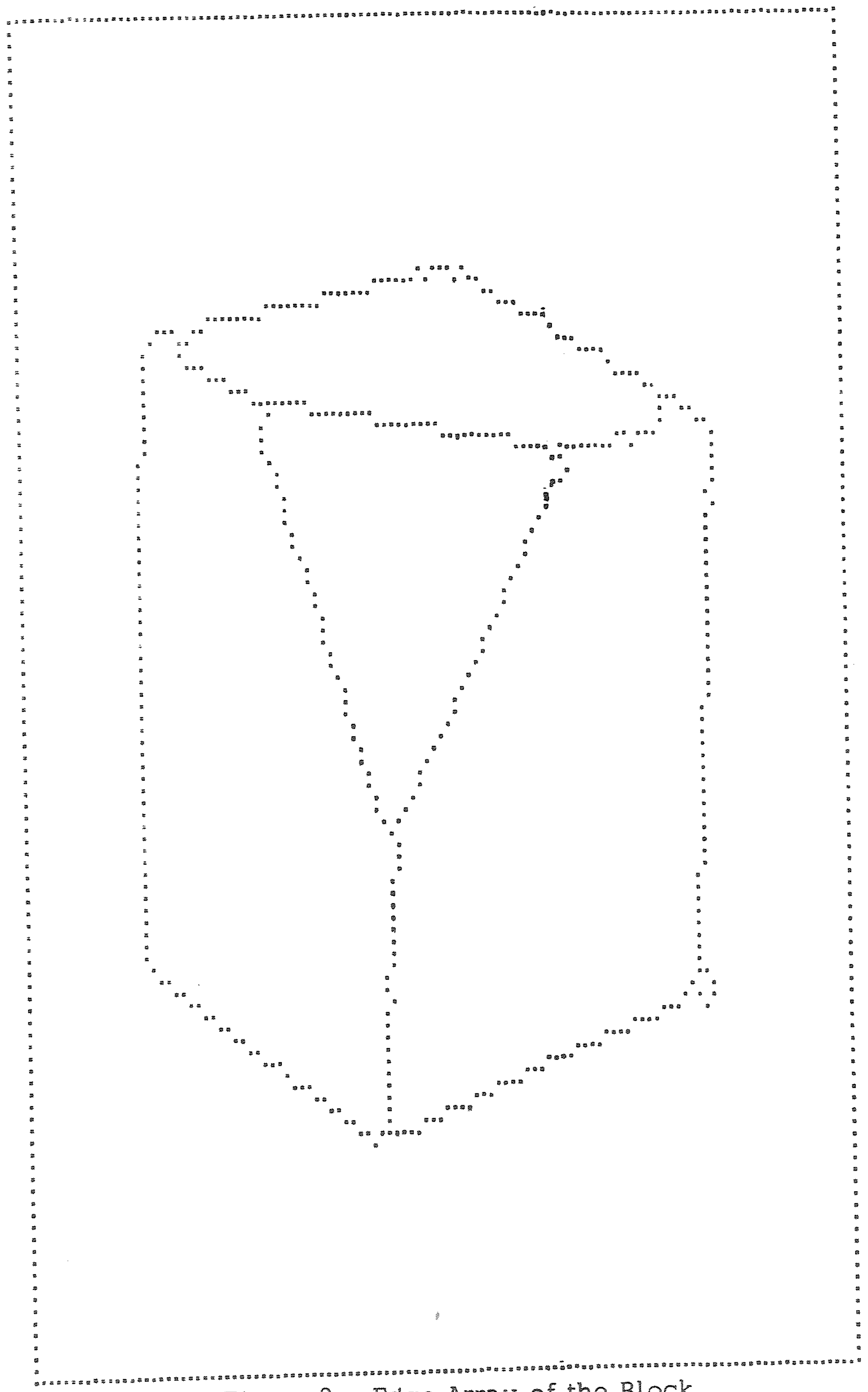


Figure 2. Edge Array of the Block

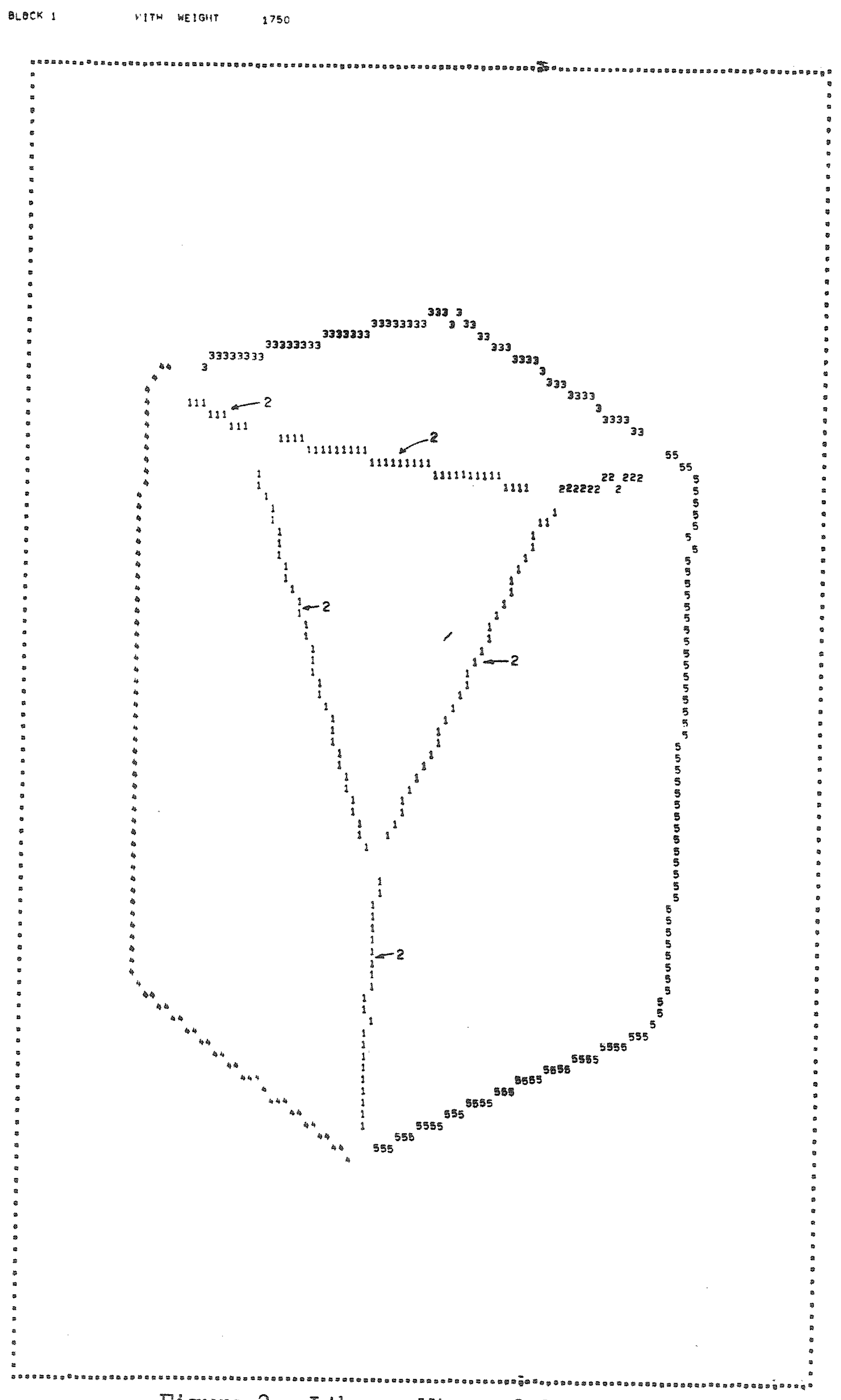


Figure 3. Library View of the Block

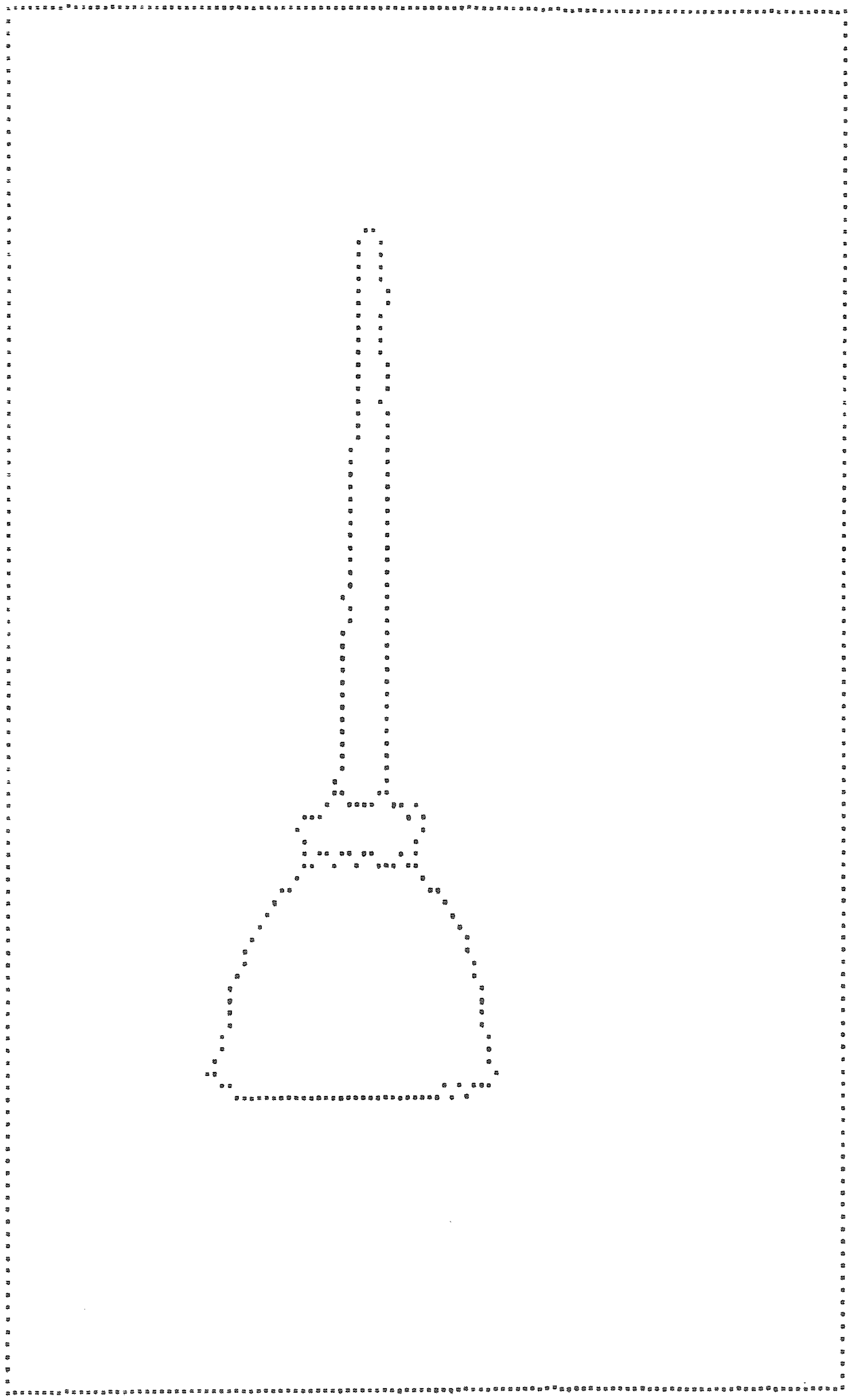


Figure 4. Edge Array of the Oil Can

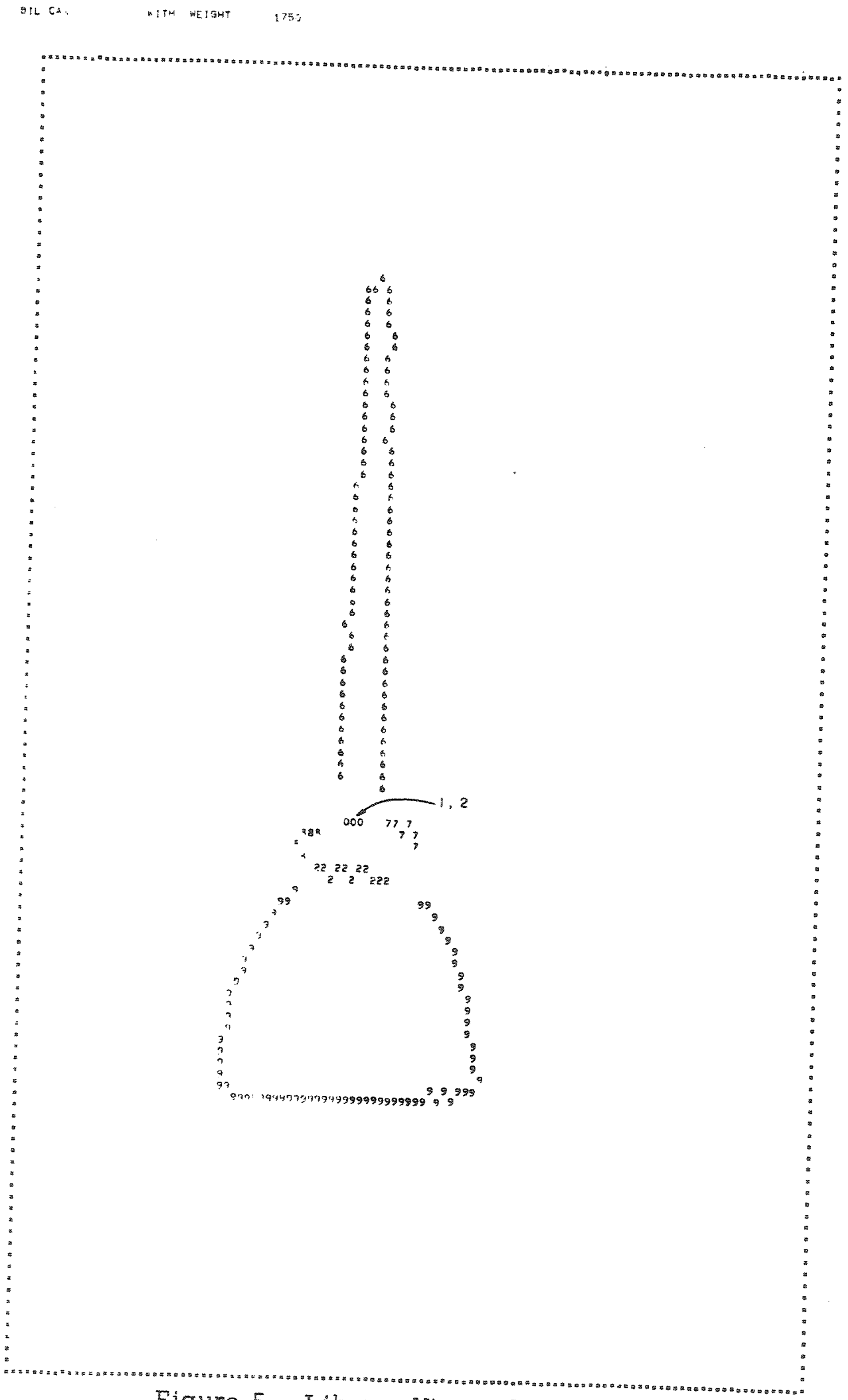


Figure 5. Library View of the Oil Can

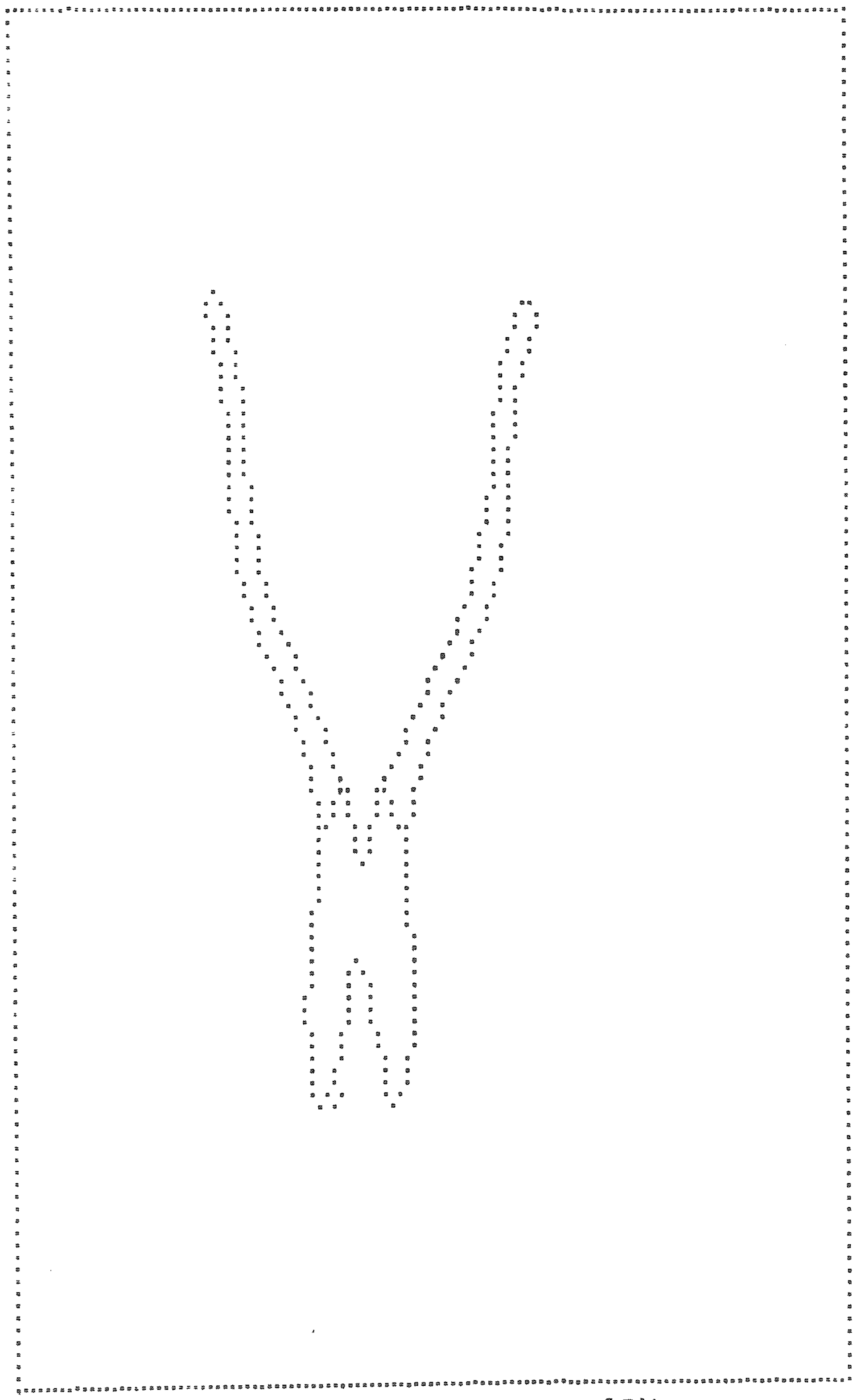


Figure 6. Edge Array of the Pair of Pliers

PAIR OF PLIERS WITH WEIGHT 1750

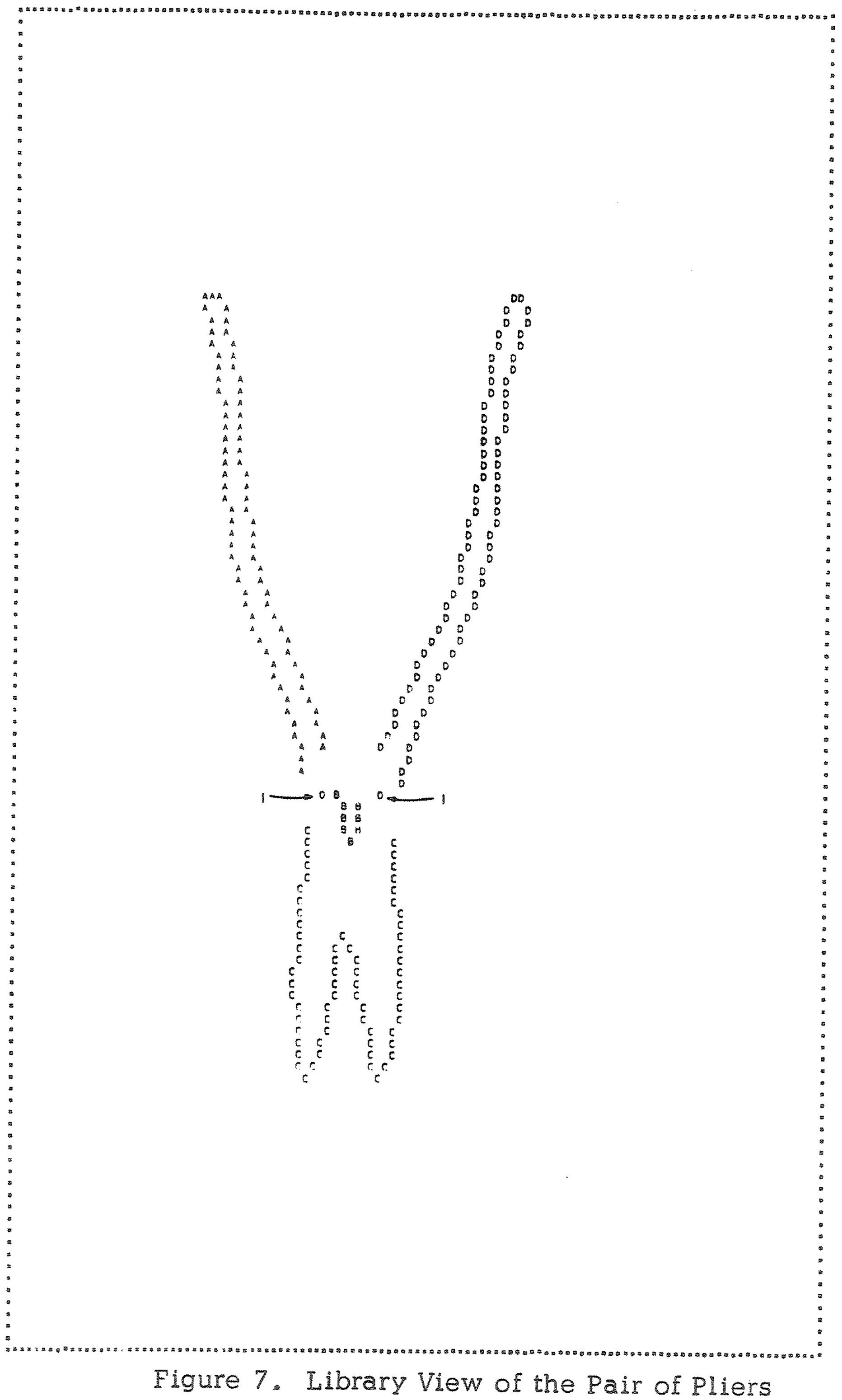


Figure 7. Library View of the Pair of Pliers

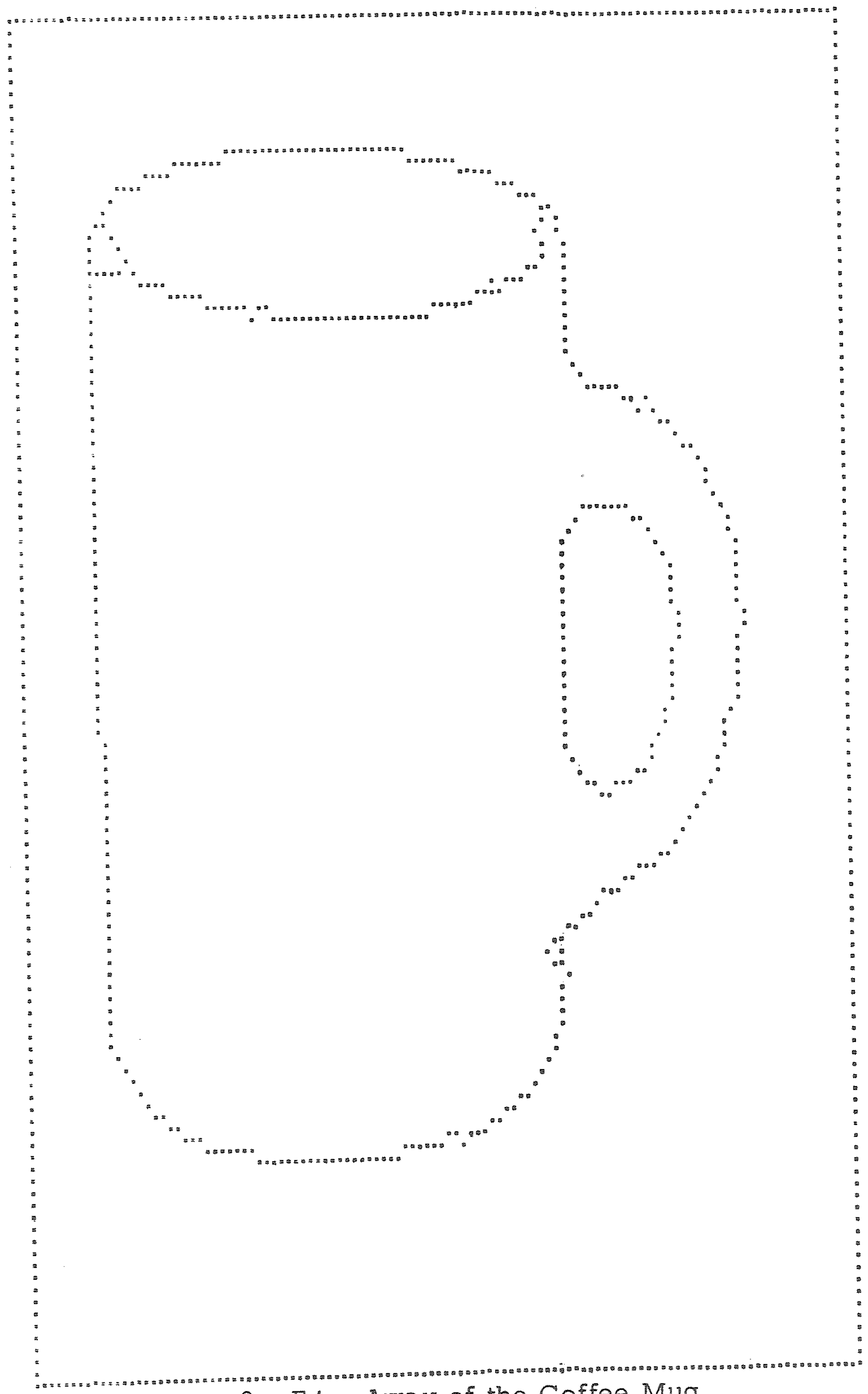


Figure 8. Edge Array of the Coffee Mug

```
COFFEE MUG        WITH  WEIGHT       1750
```

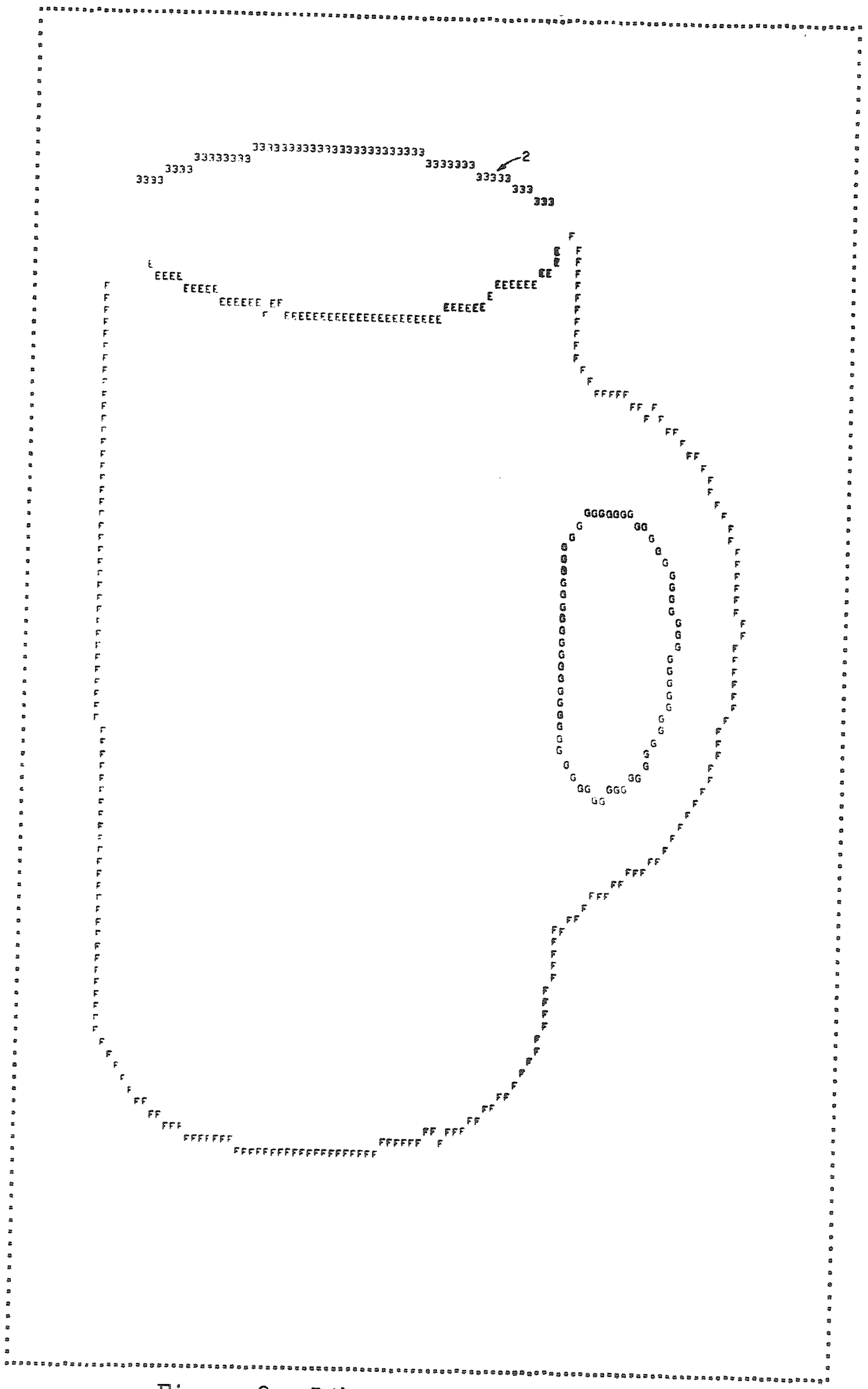


Figure 9. Library View of the Coffee Mug

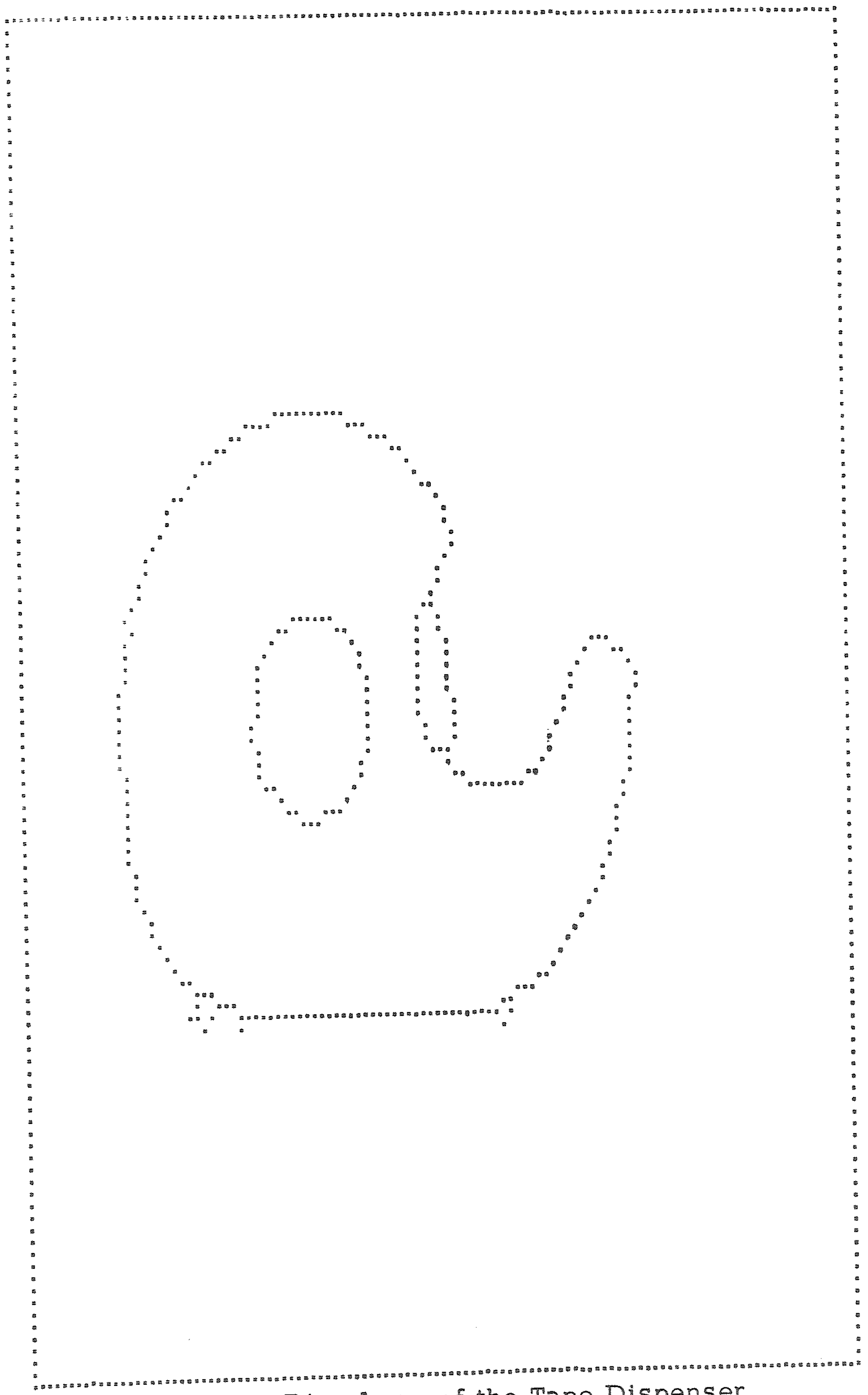


Figure 10. Edge Array of the Tape Dispenser

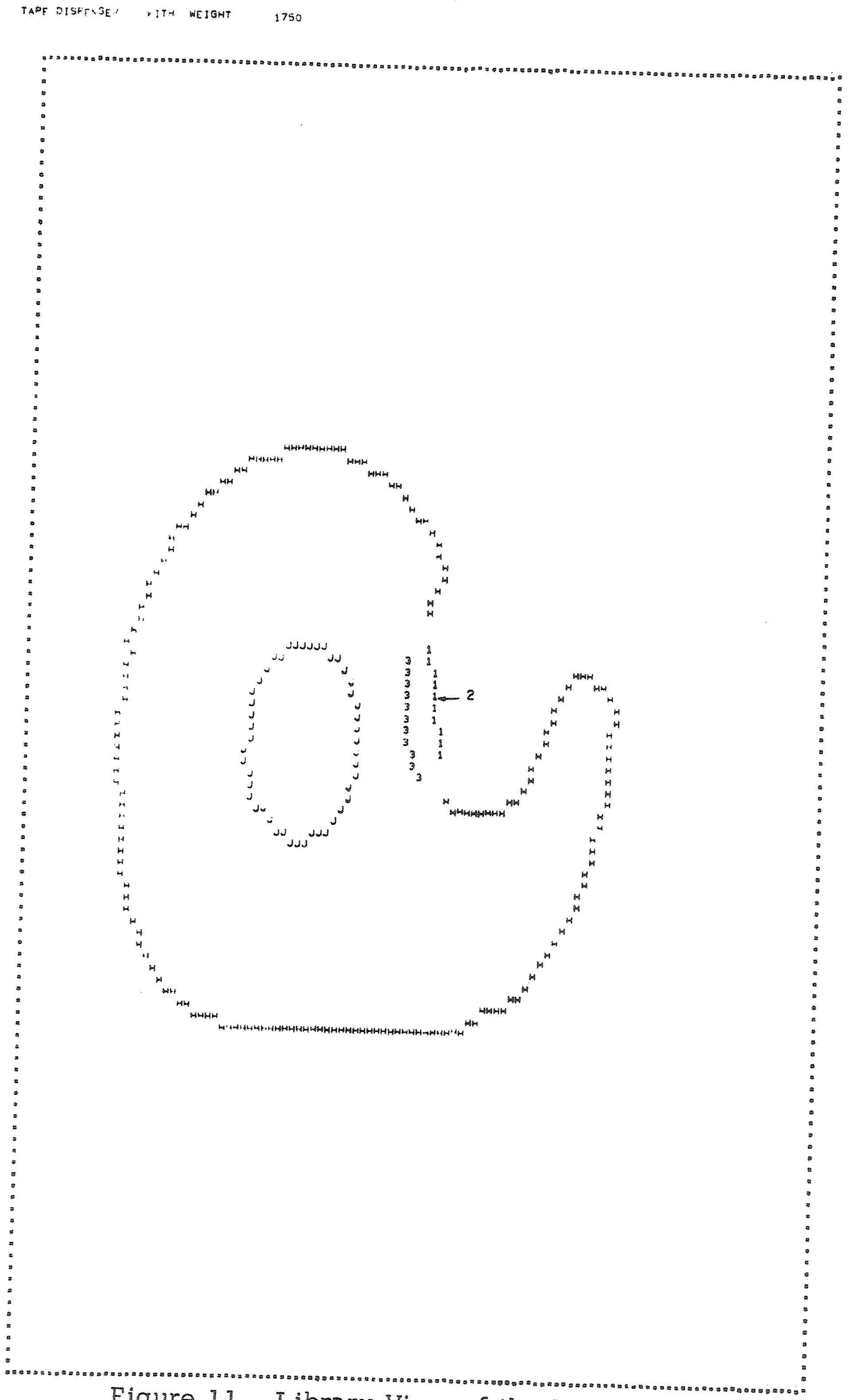


Figure 11. Library View of the Tape Dispenser

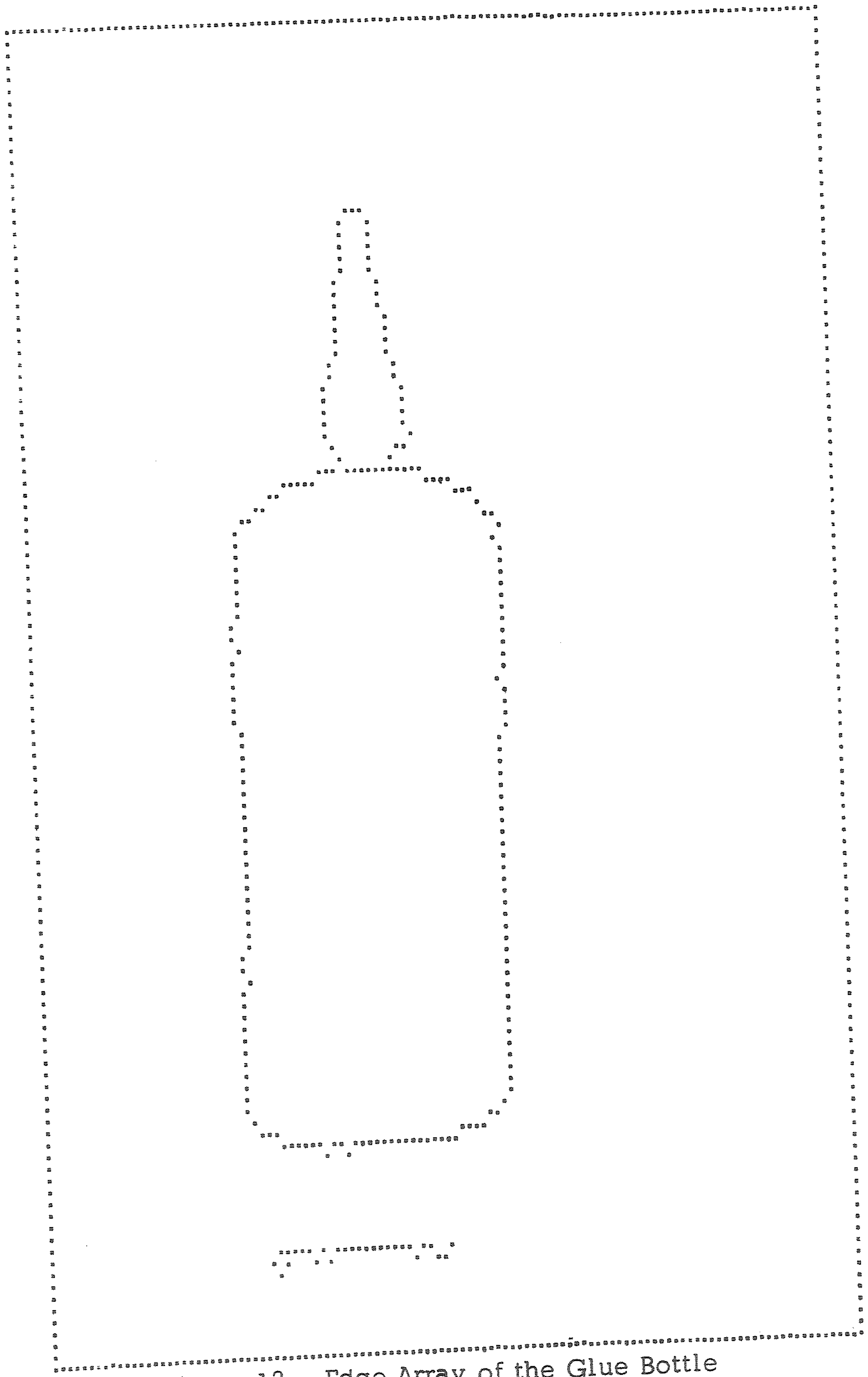


Figure 12. Edge Array of the Glue Bottle

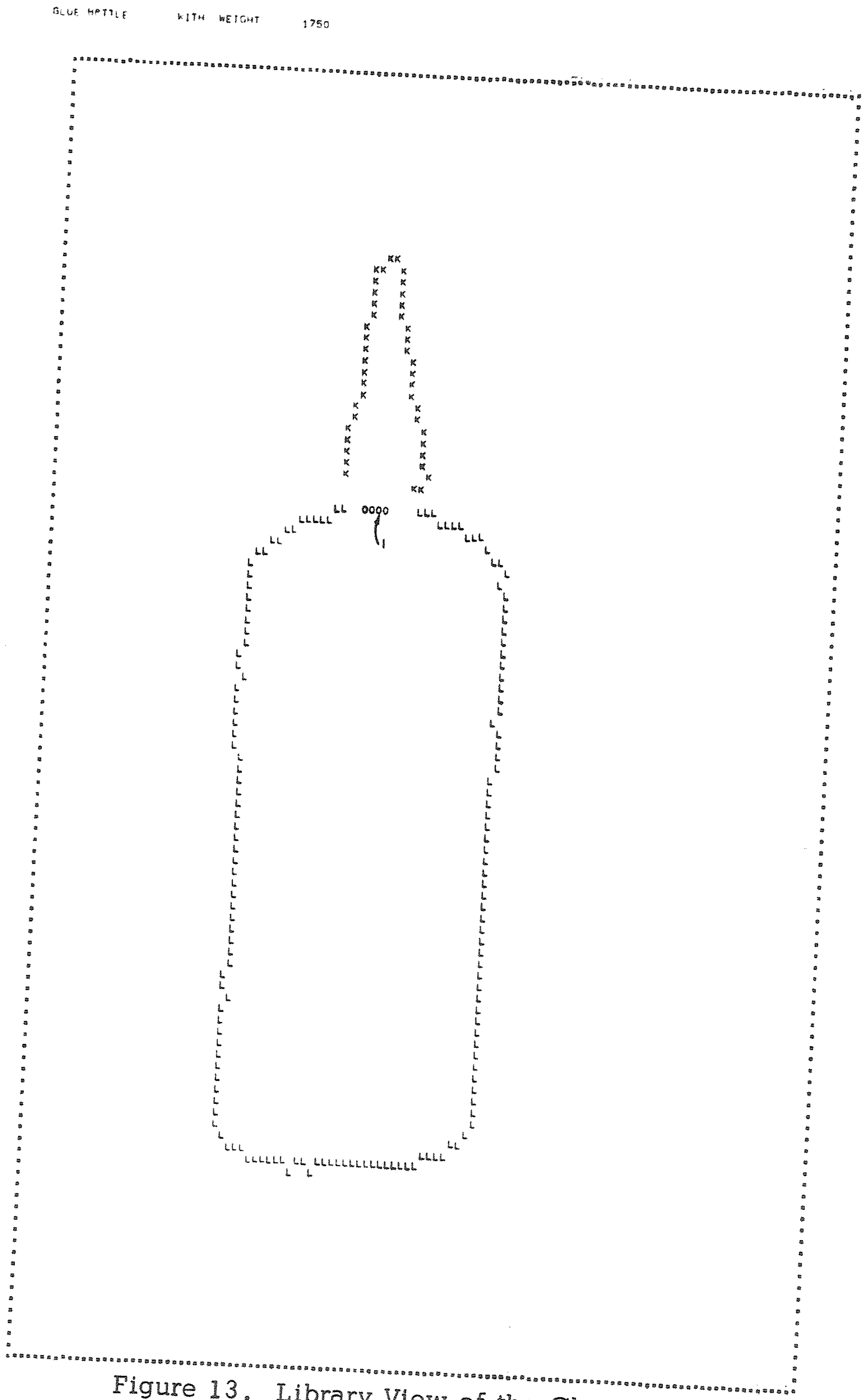


Figure 13. Library View of the Glue Bottle

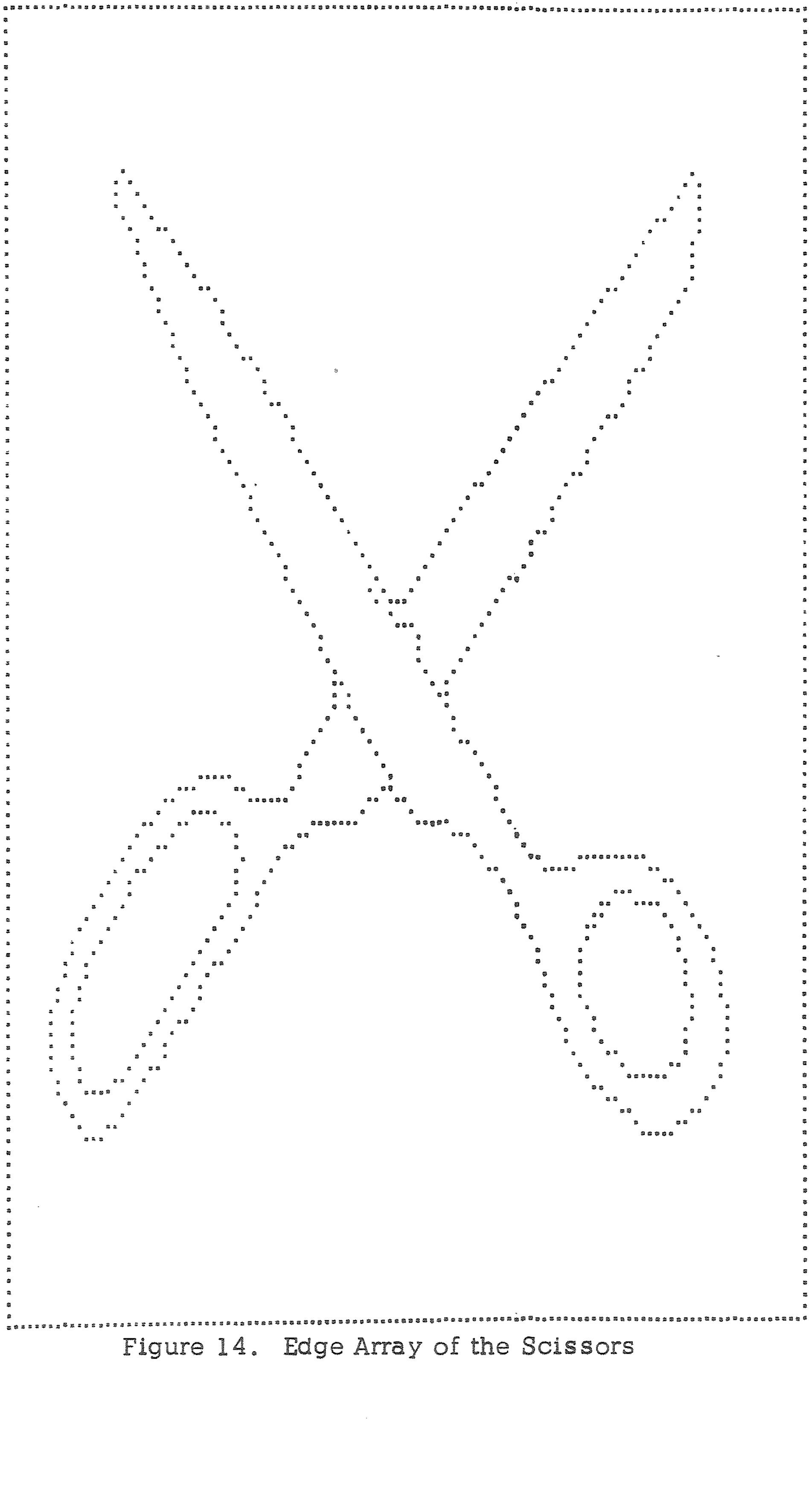


Figure 14. Edge Array of the Scissors

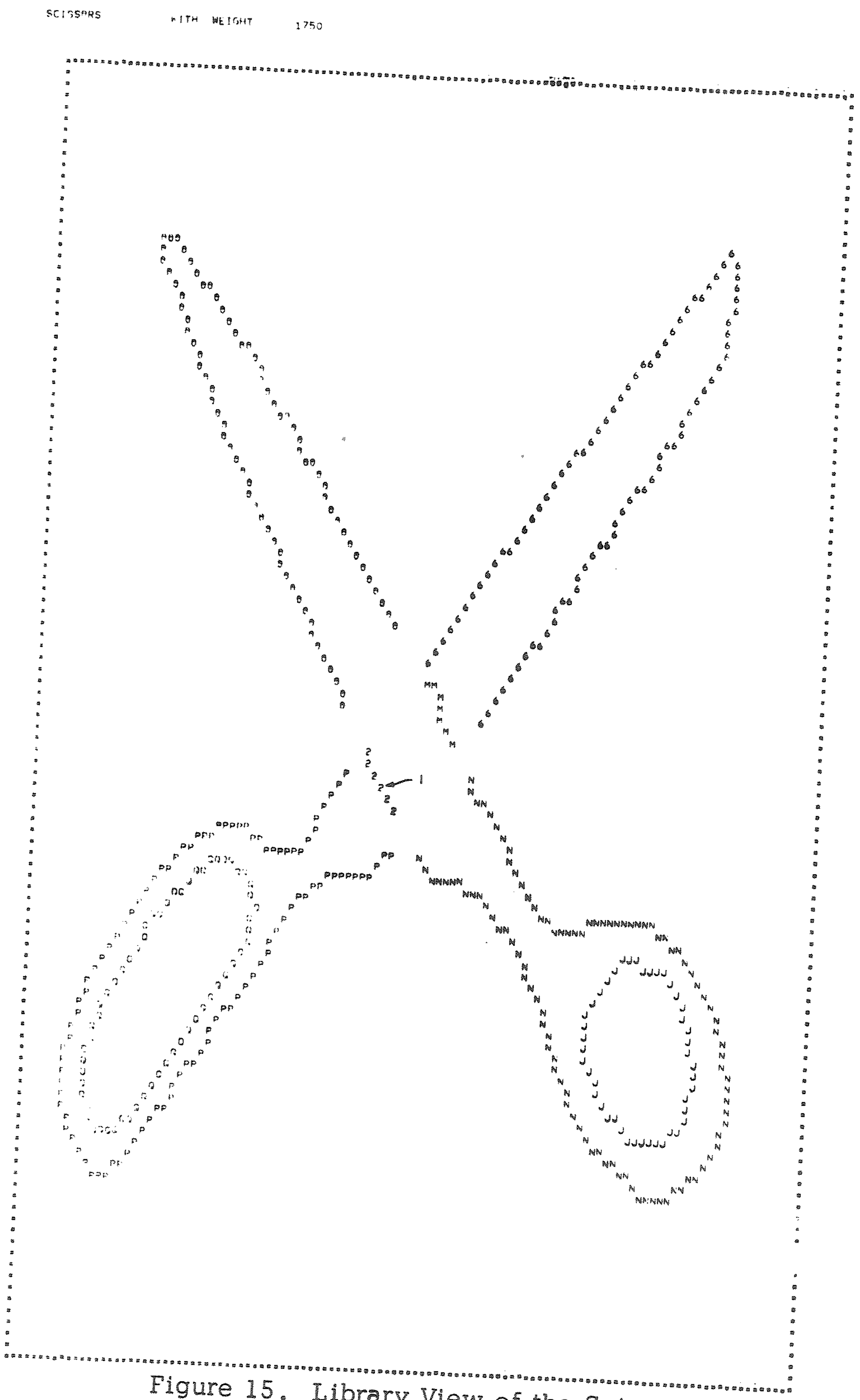


Figure 15. Library View of the Scissors

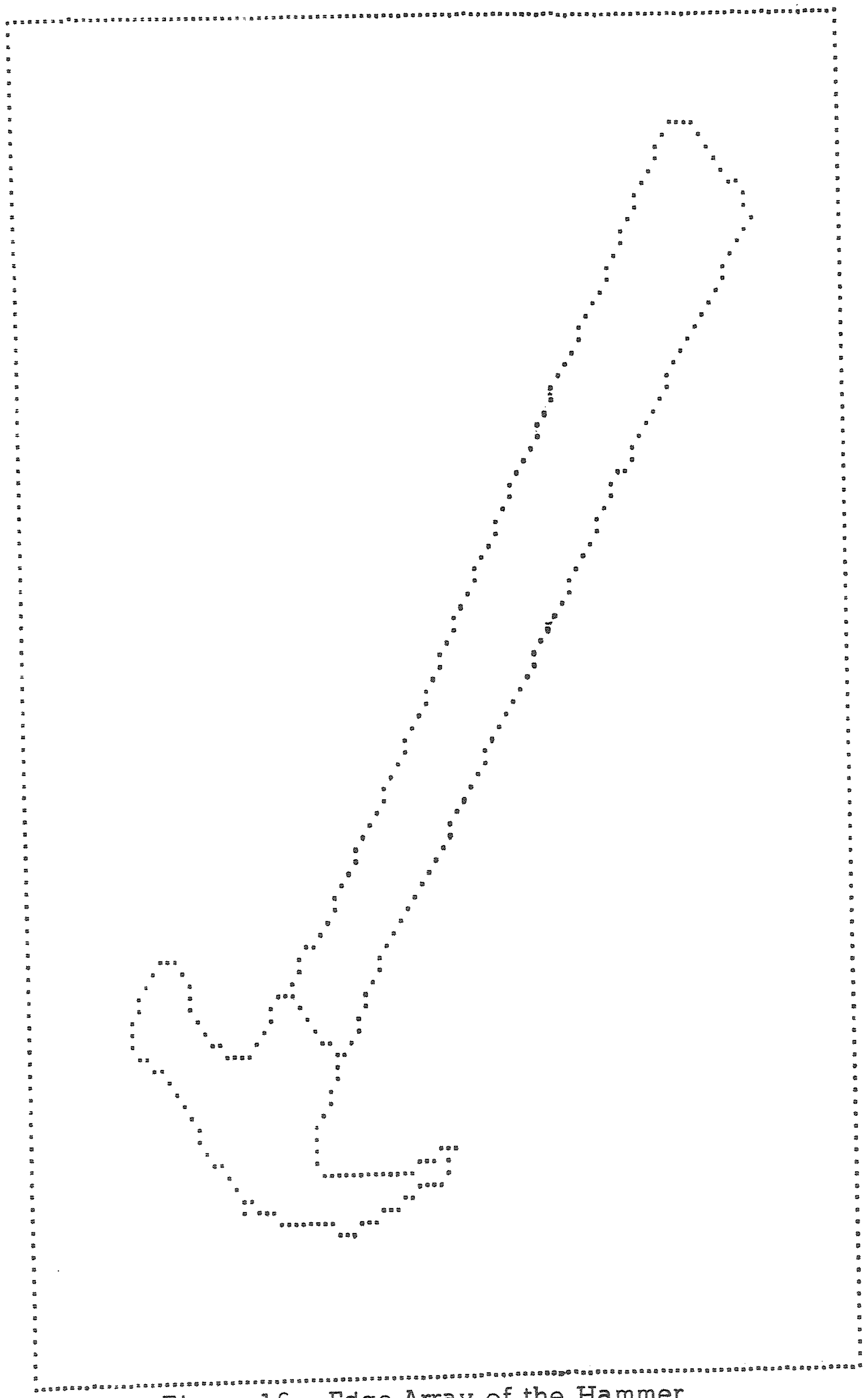


Figure 16. Edge Array of the Hammer

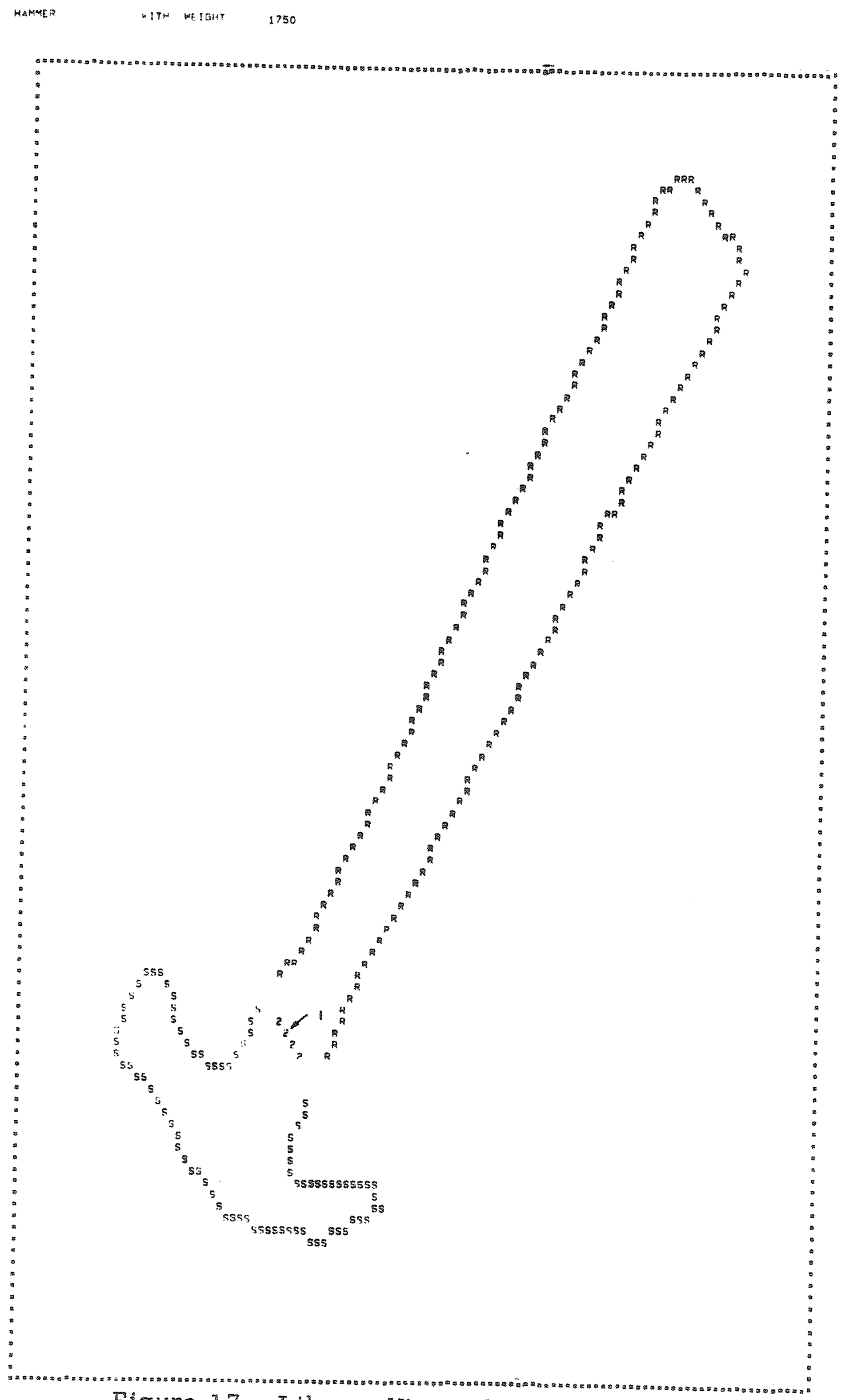


Figure 17. Library View of the Hammer

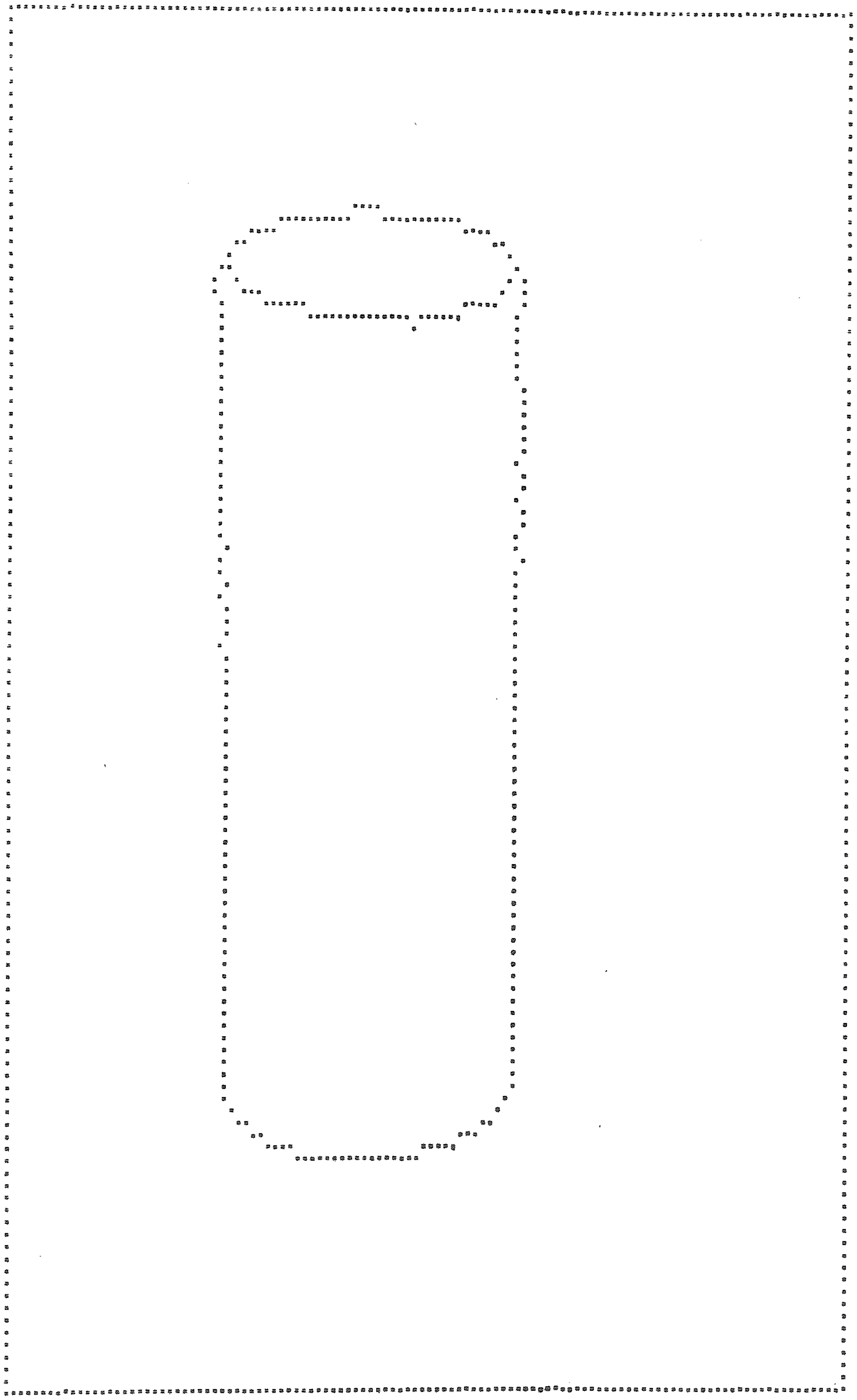


Figure 18. Edge Array of the Can

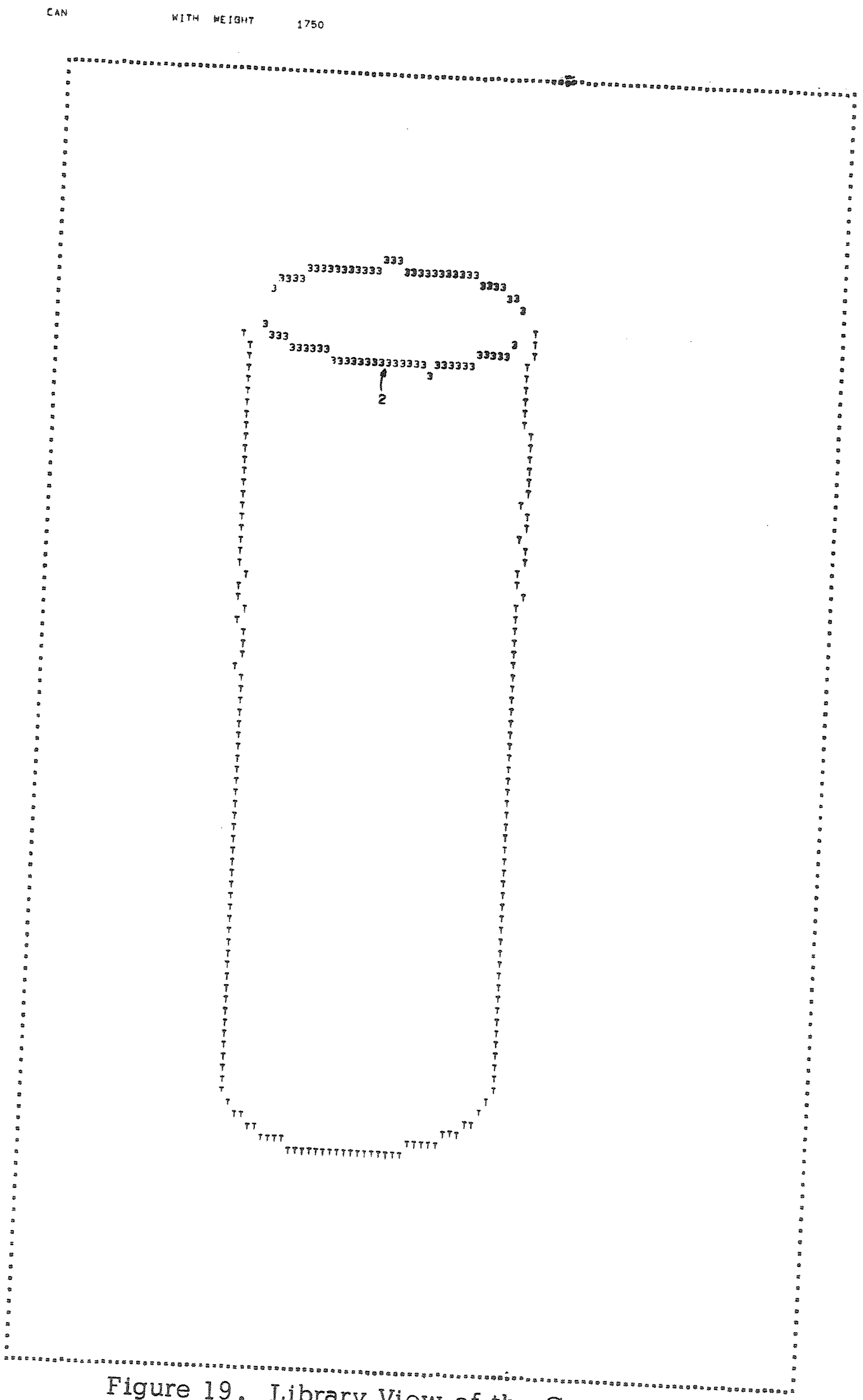


Figure 19. Library View of the Can

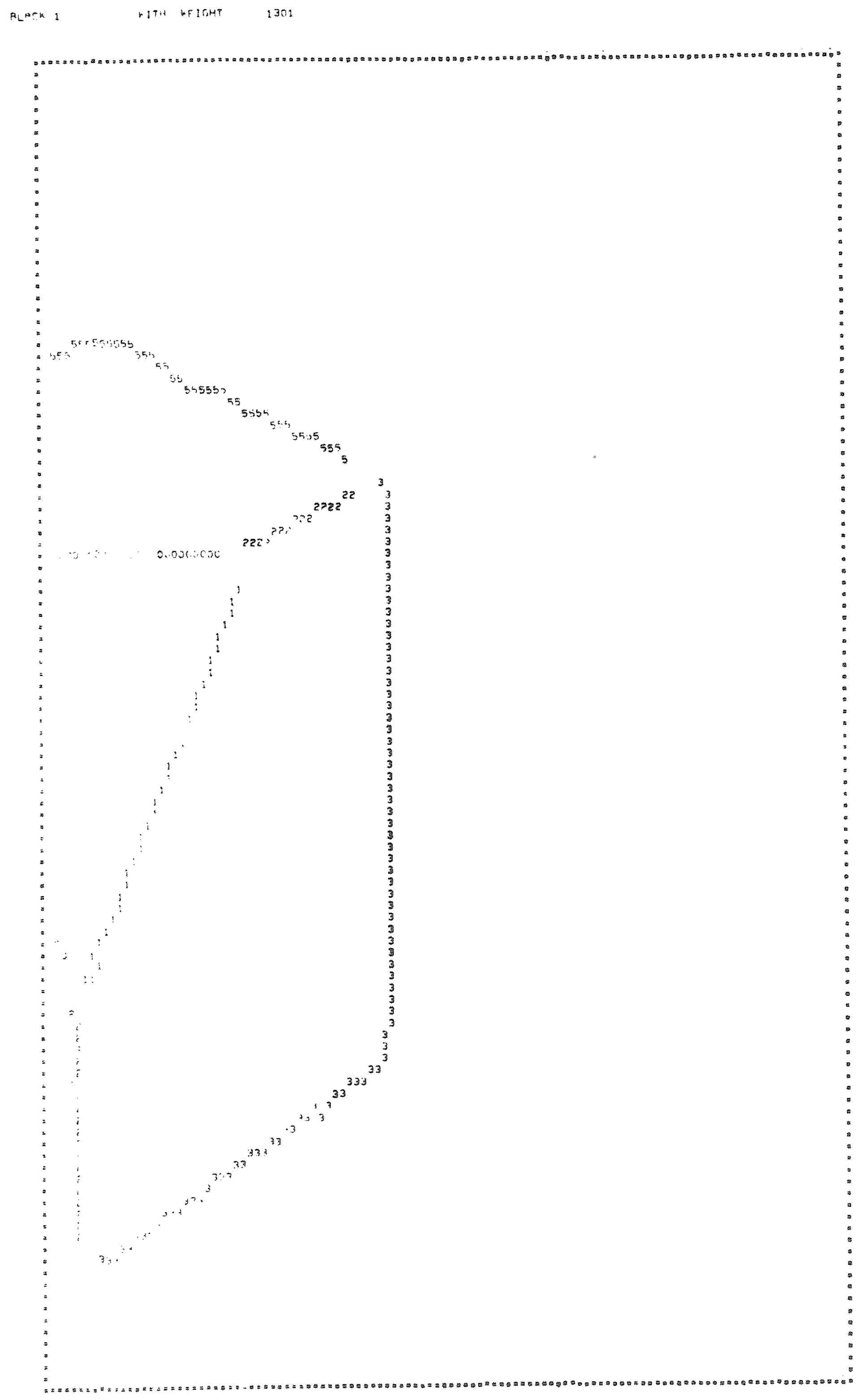


Figure 20. Partial View of the Block

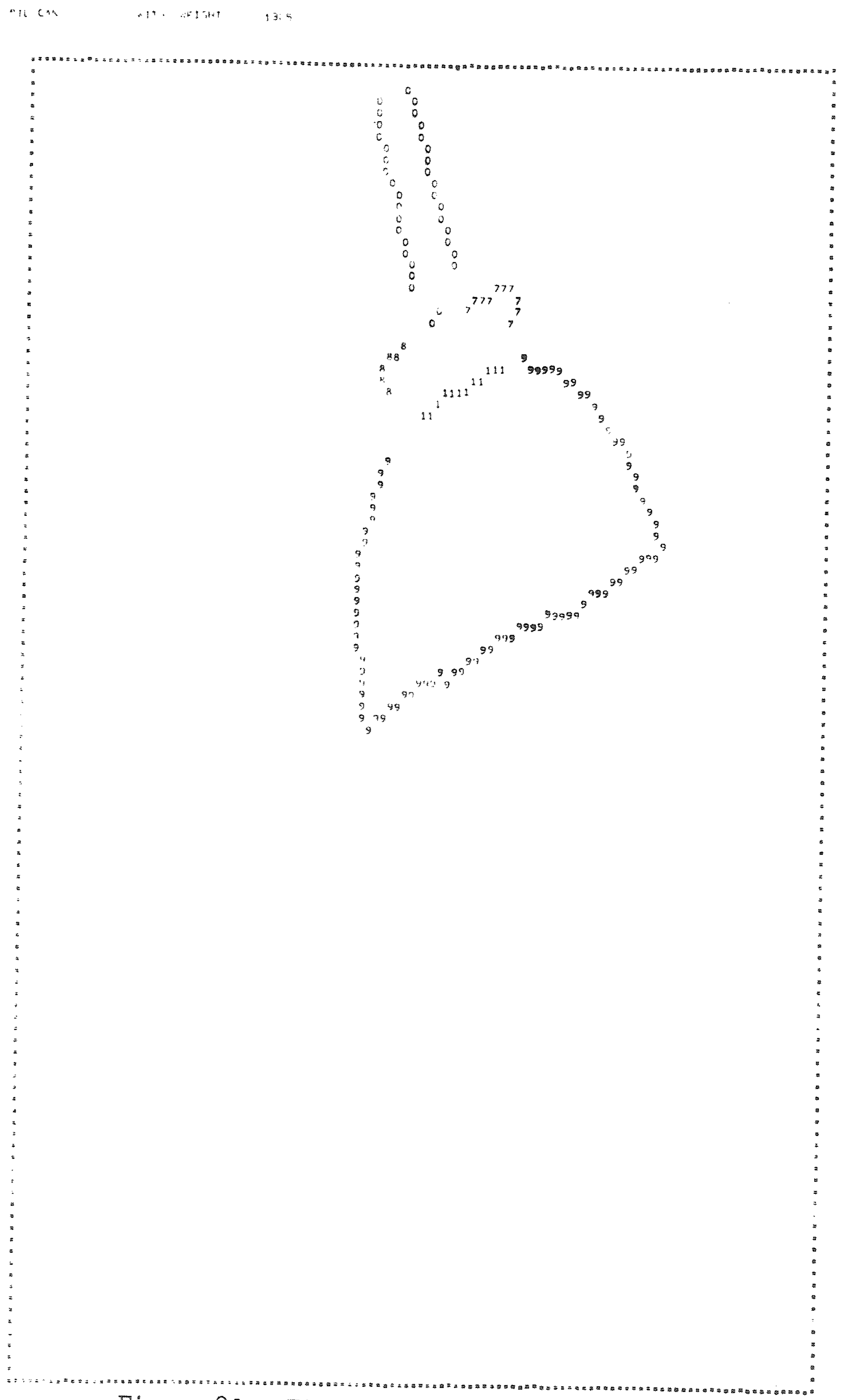


Figure 21. First Partial View of the Oil Can

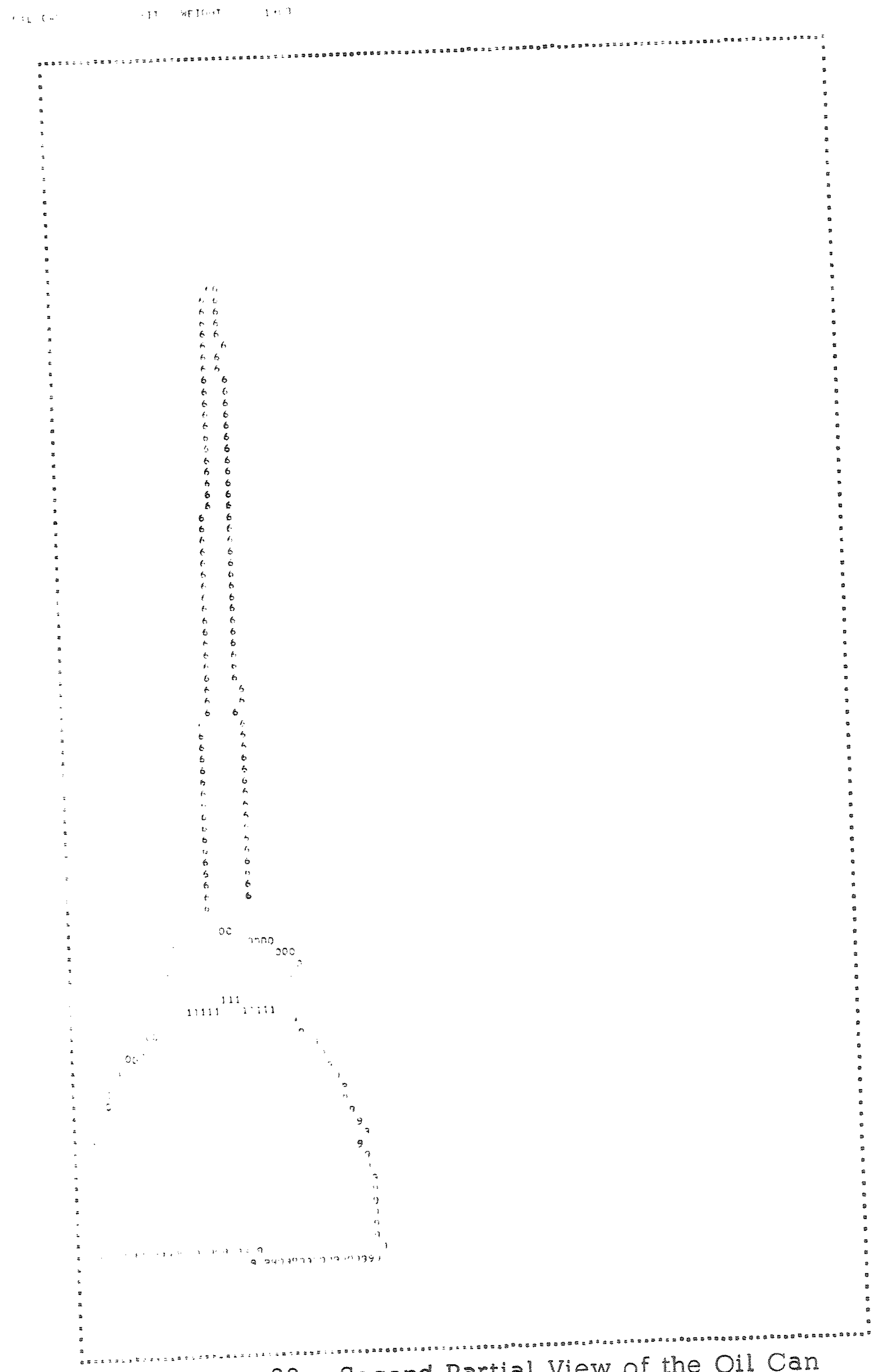


Figure 22. Second Partial View of the Oil Can

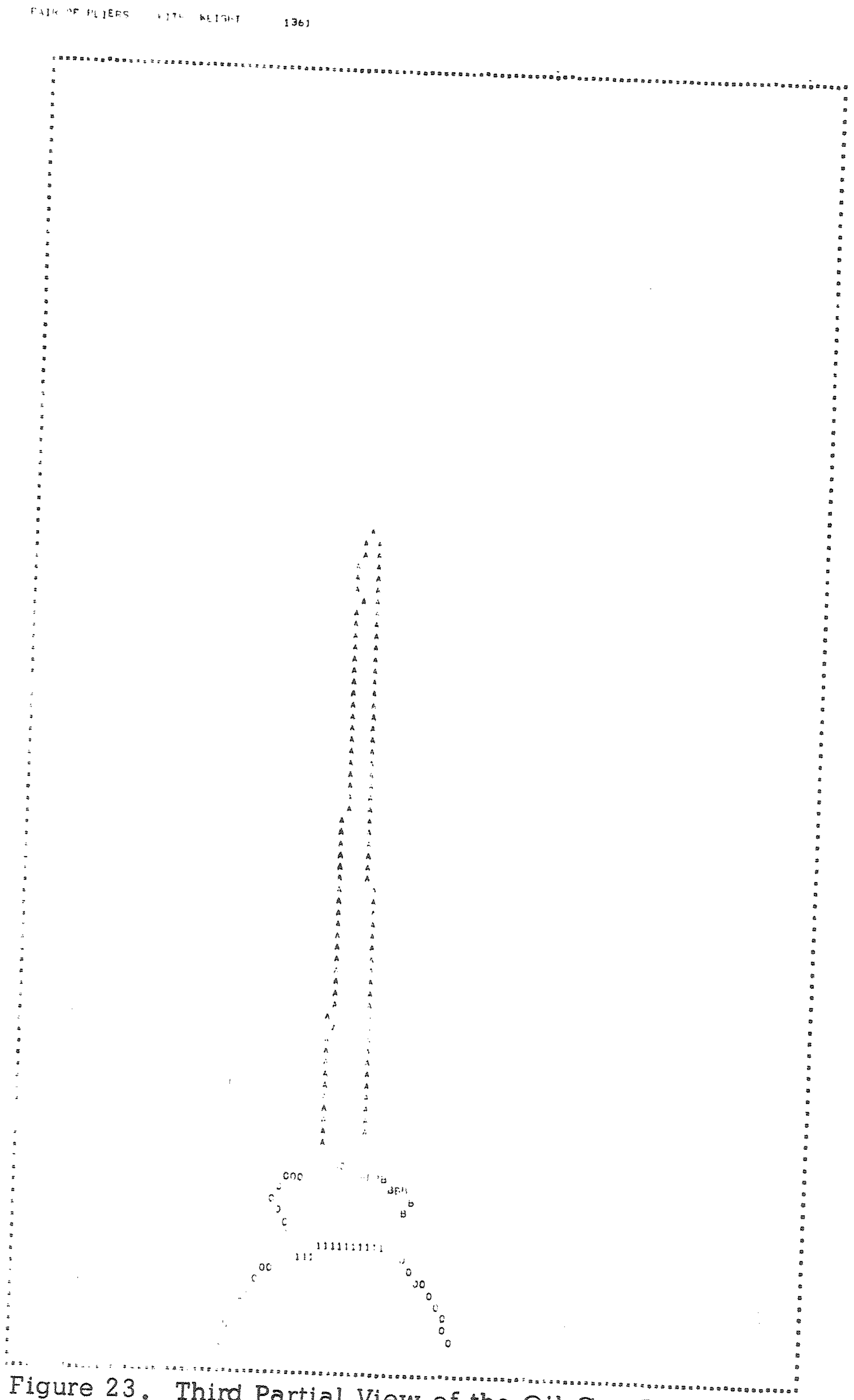


Figure 23. Third Partial View of the Oil Can Identified as a Pair of Pliers

```
OIL CAN          WITH WEIGHT      1353
```

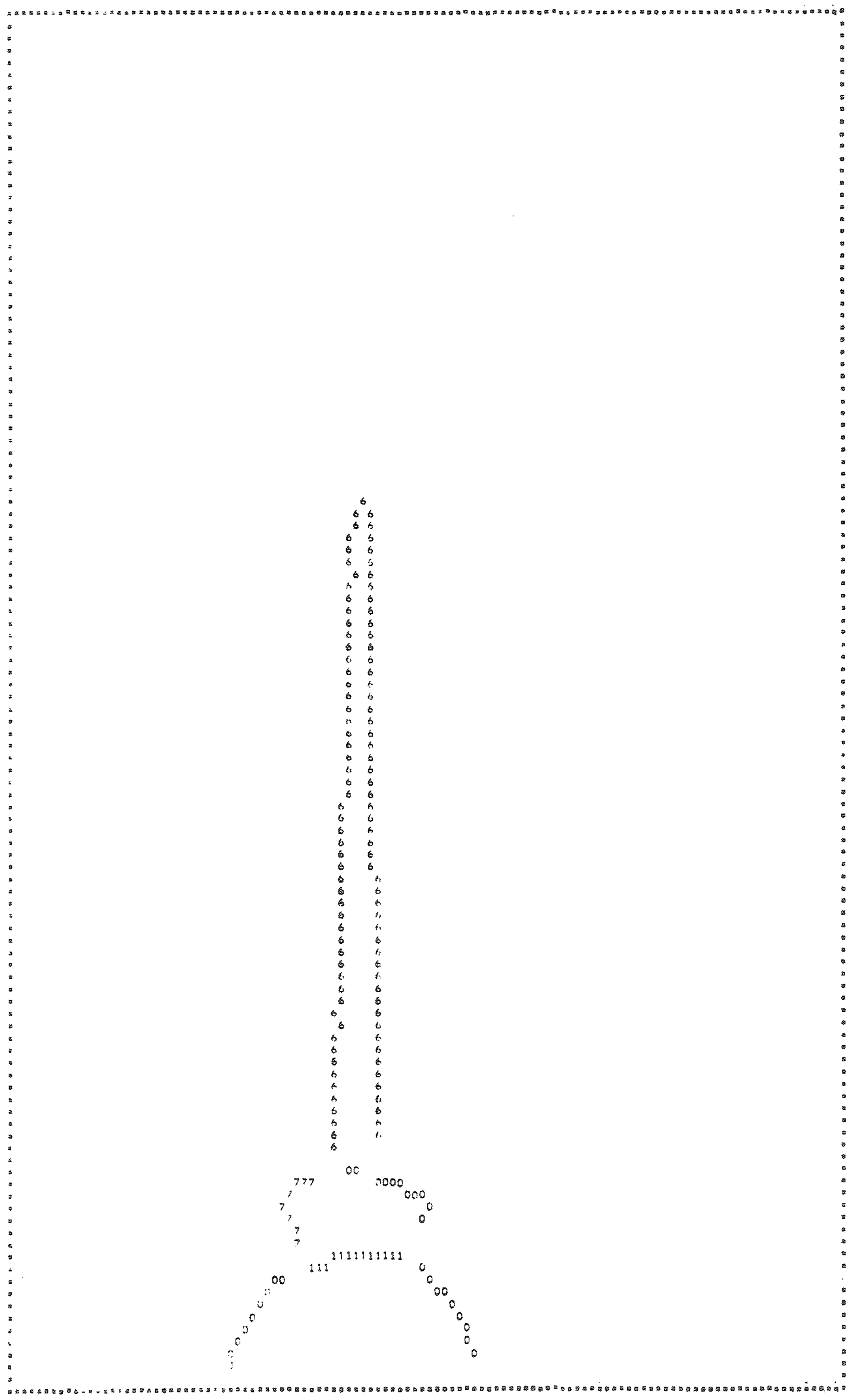


Figure 24. Third Partial View of the Oil Can

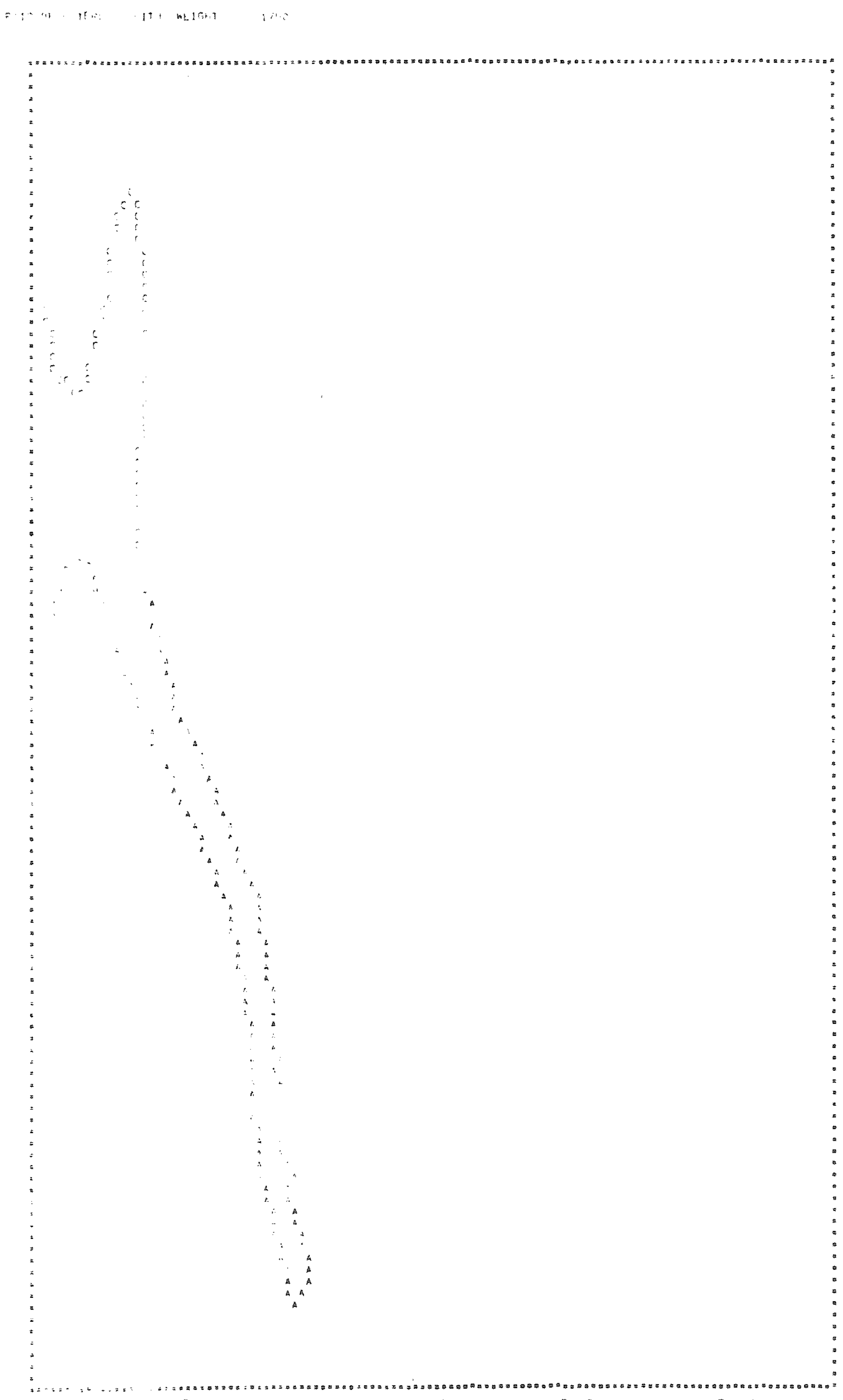


Figure 26. Second Partial View of the Pair of Pliers

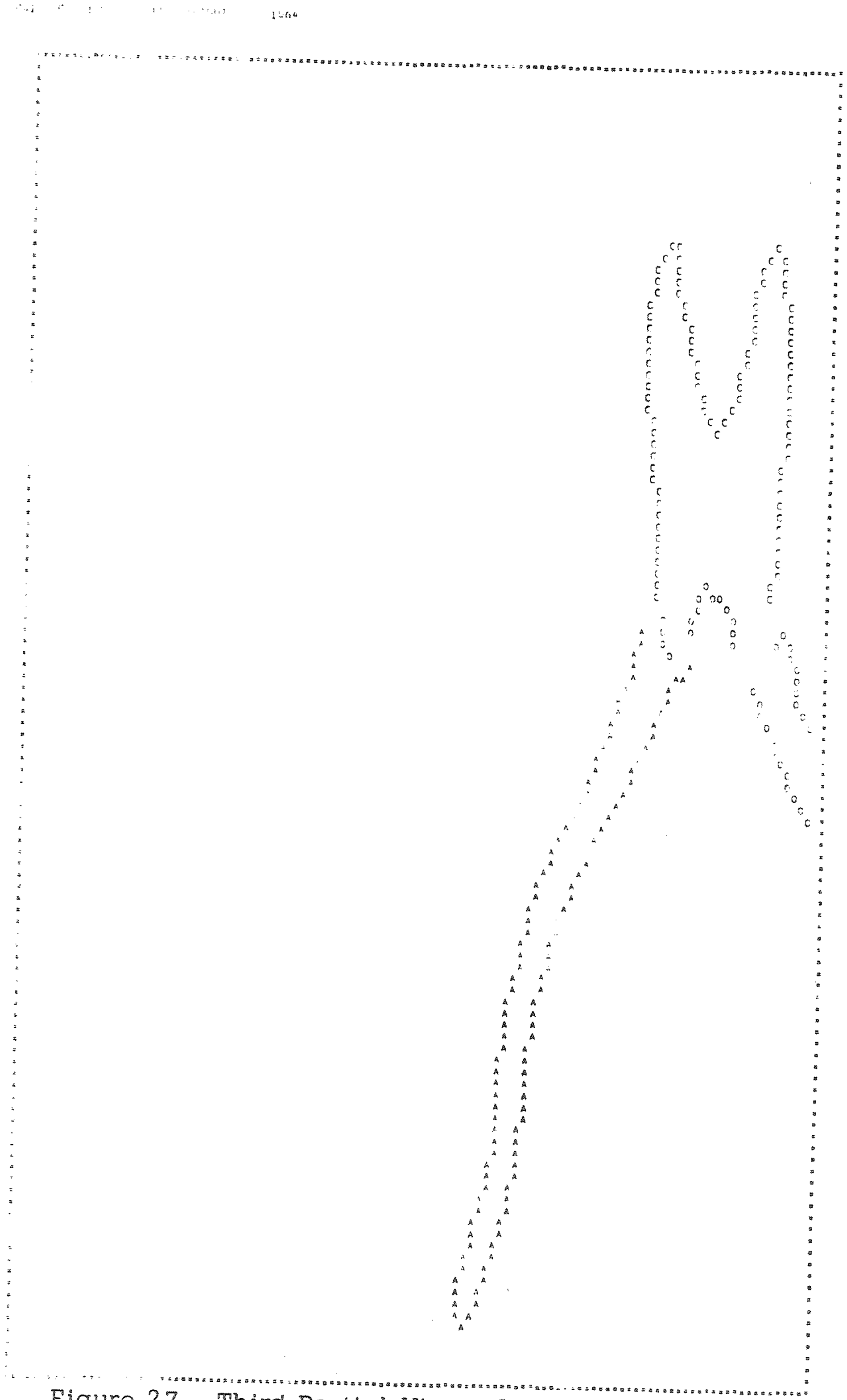


Figure 27. Third Partial View of the Pair of Pliers

PAIR OF PLIERS   1TH WEIGHT   1661

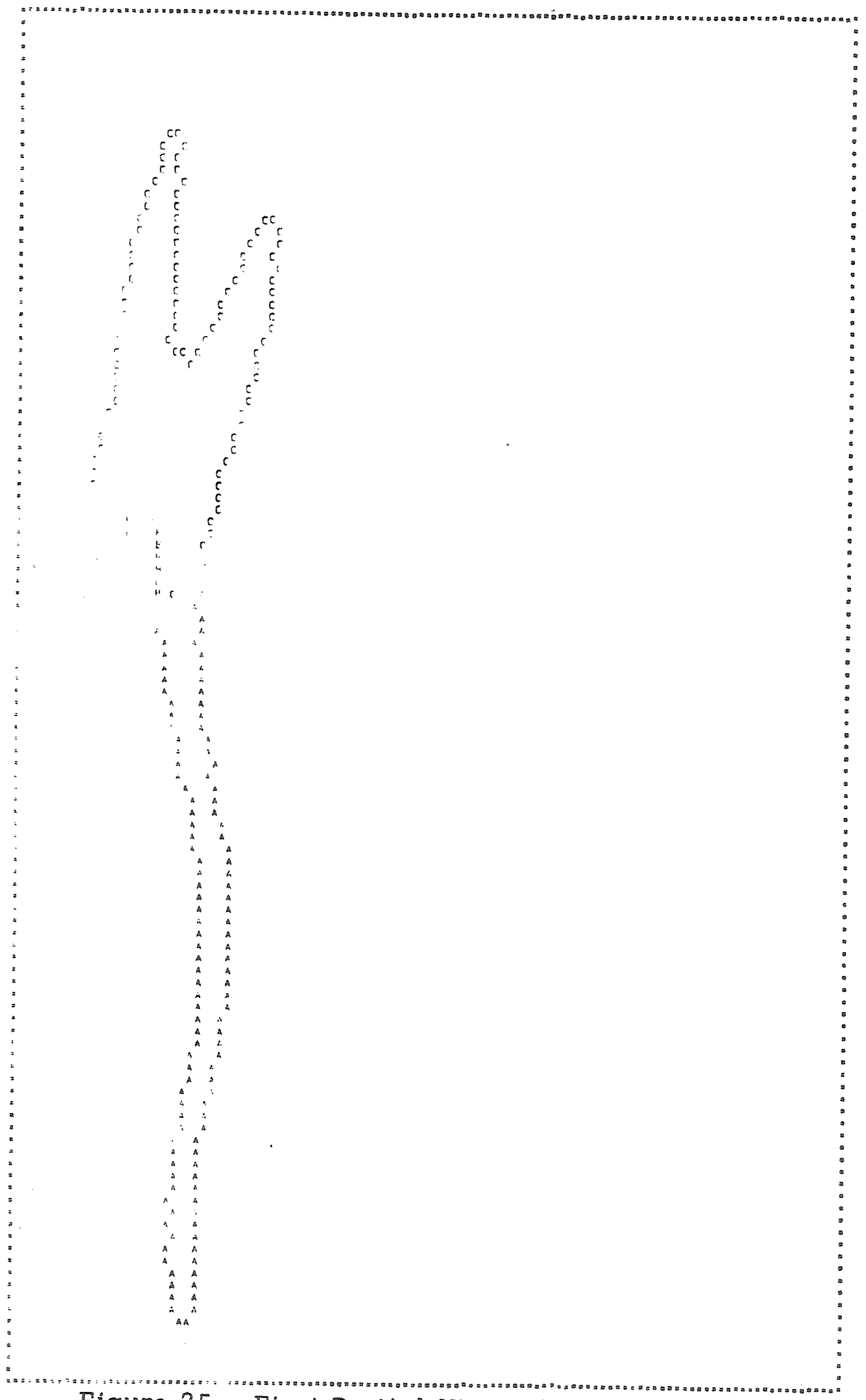


Figure 25. First Partial View of the Pair of Pliers

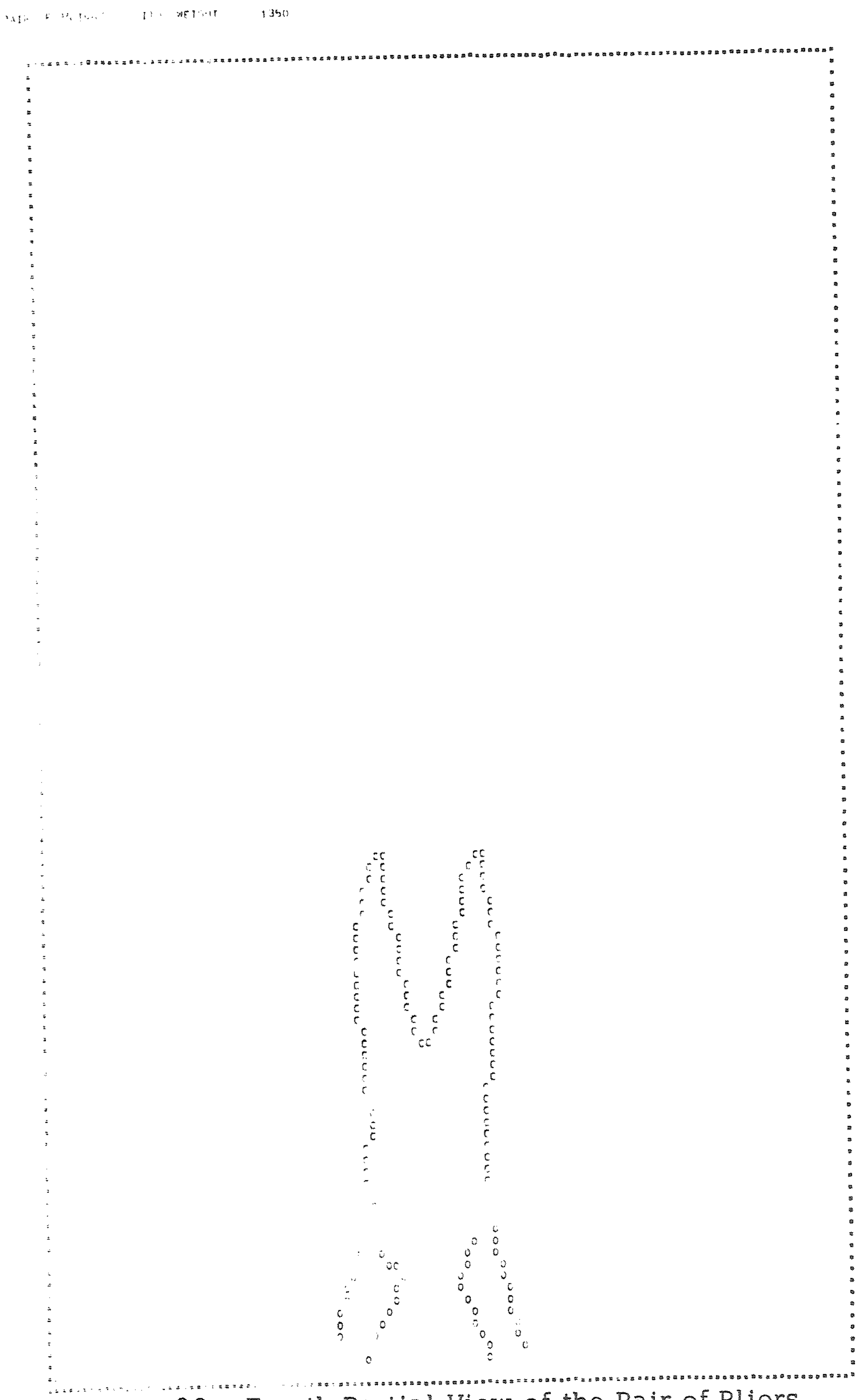


Figure 28. Fourth Partial View of the Pair of Pliers

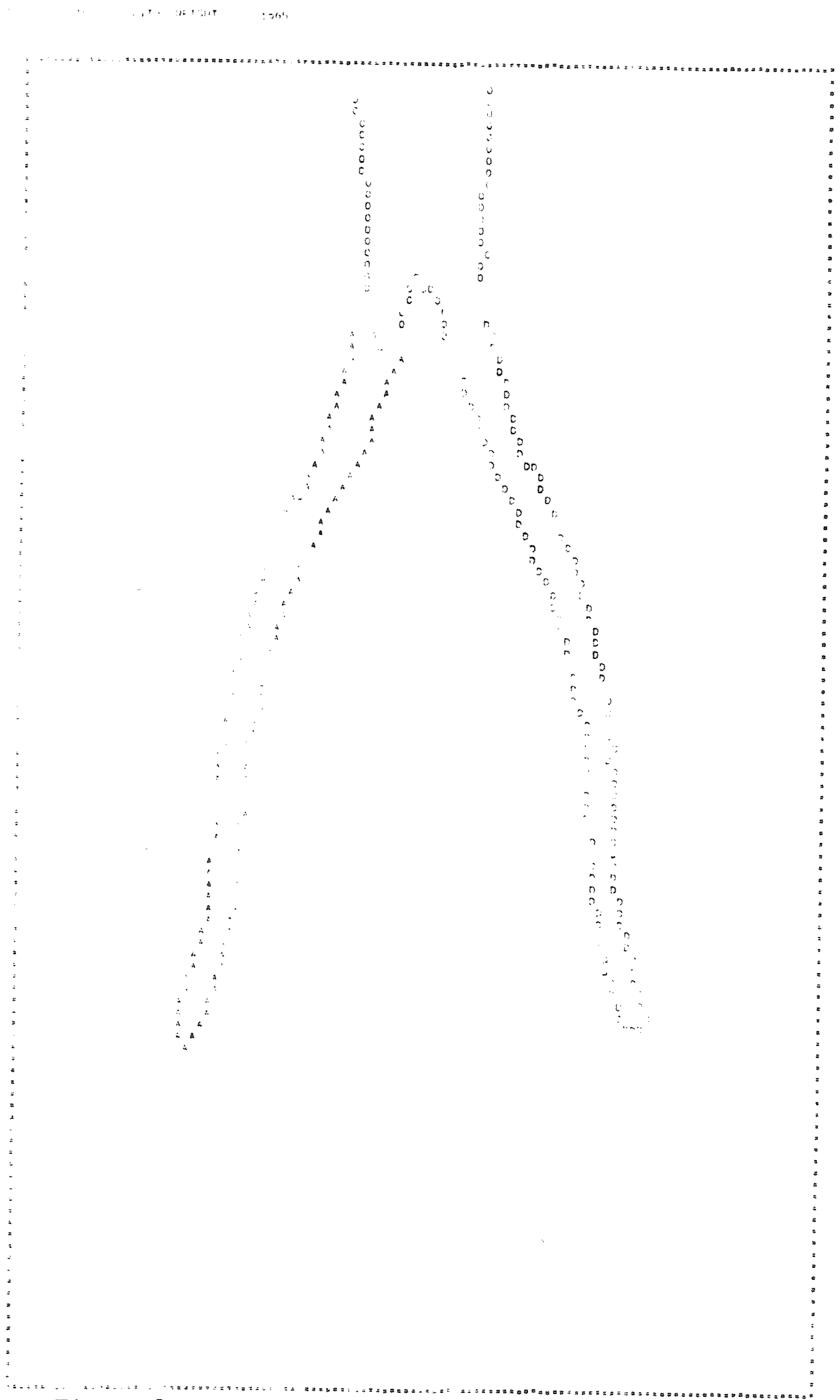


Figure 29. Fifth Partial View of the Pair of Pliers

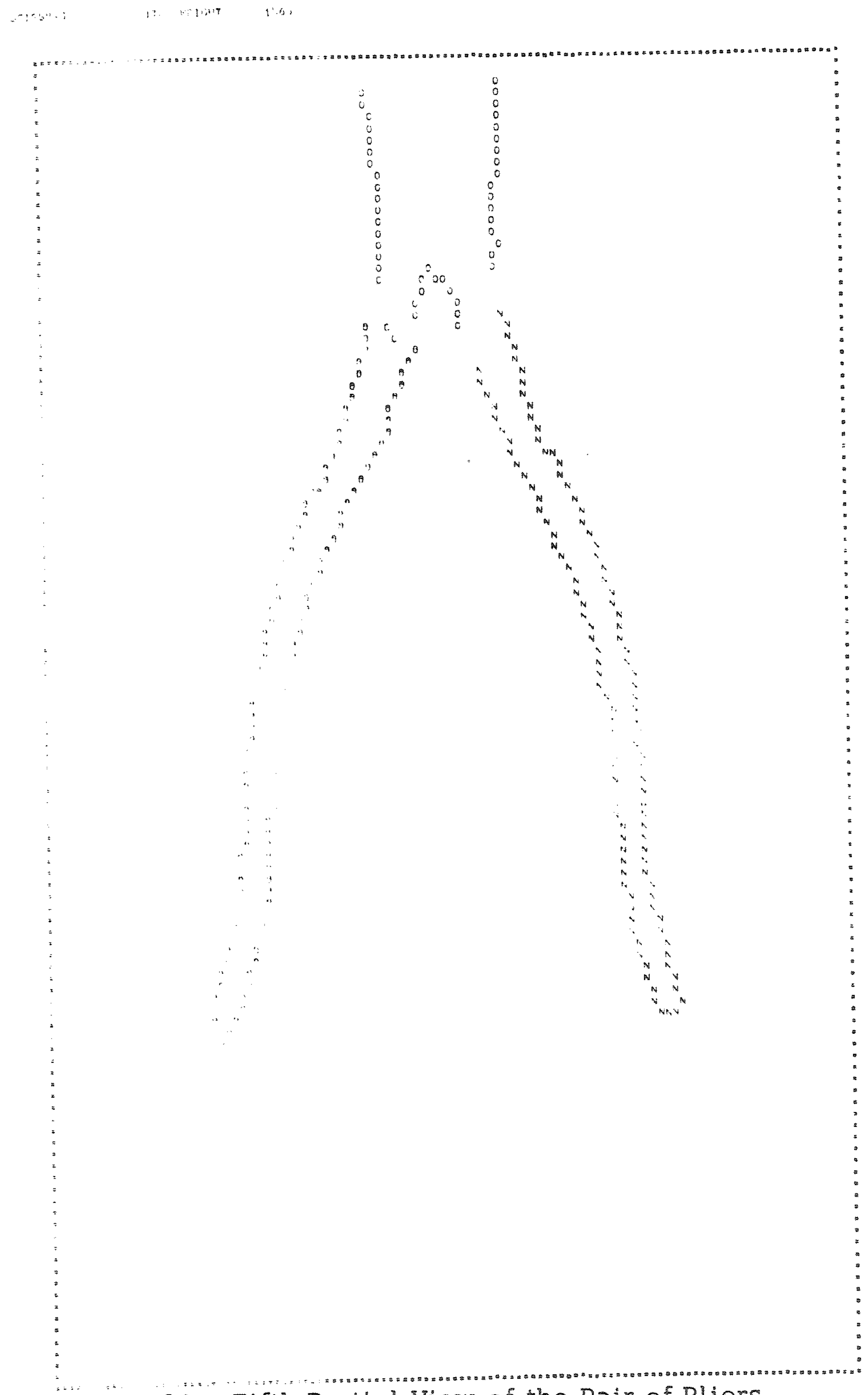


Figure 30. Fifth Partial View of the Pair of Pliers Identified as Scissors

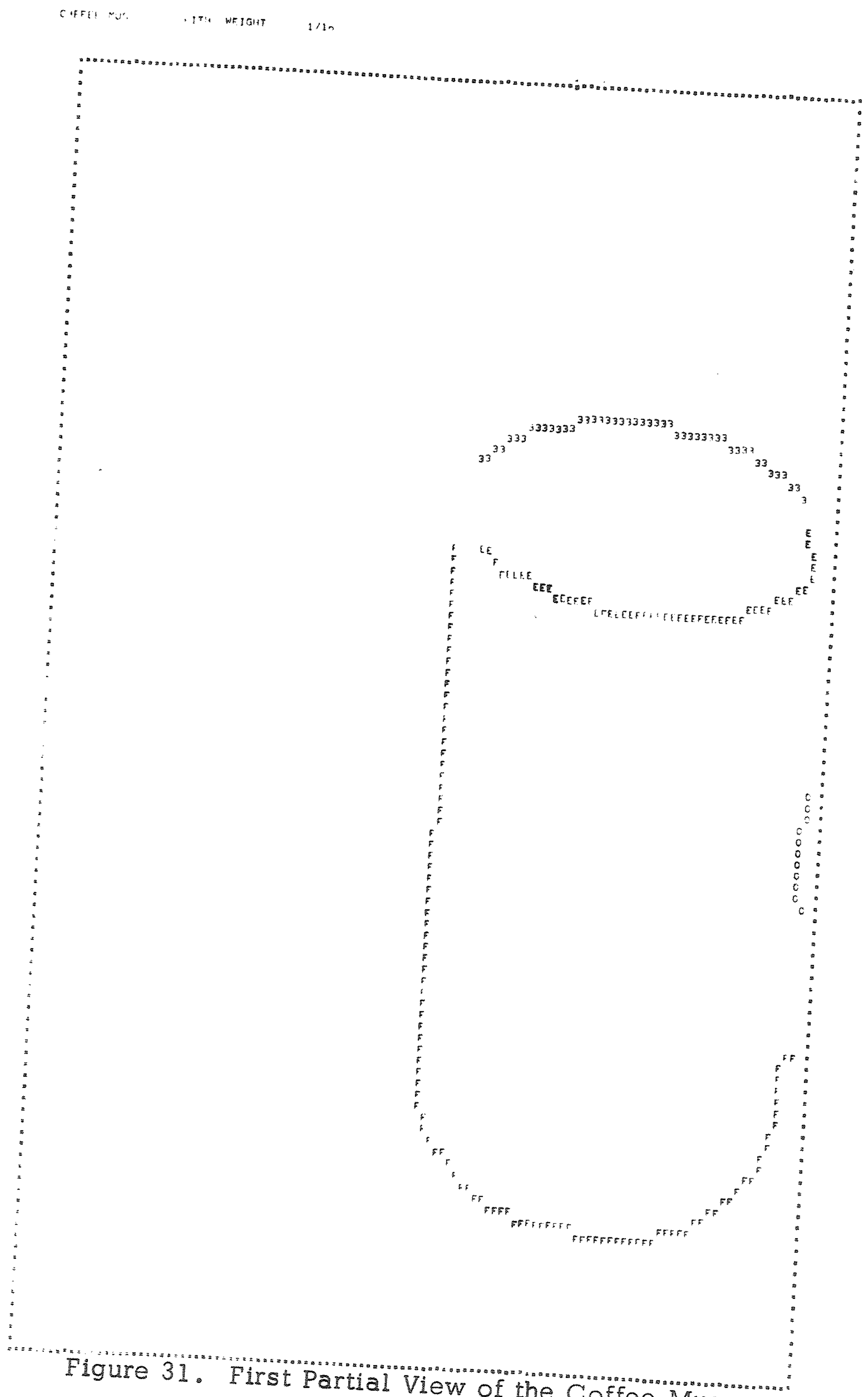


Figure 31. First Partial View of the Coffee Mug

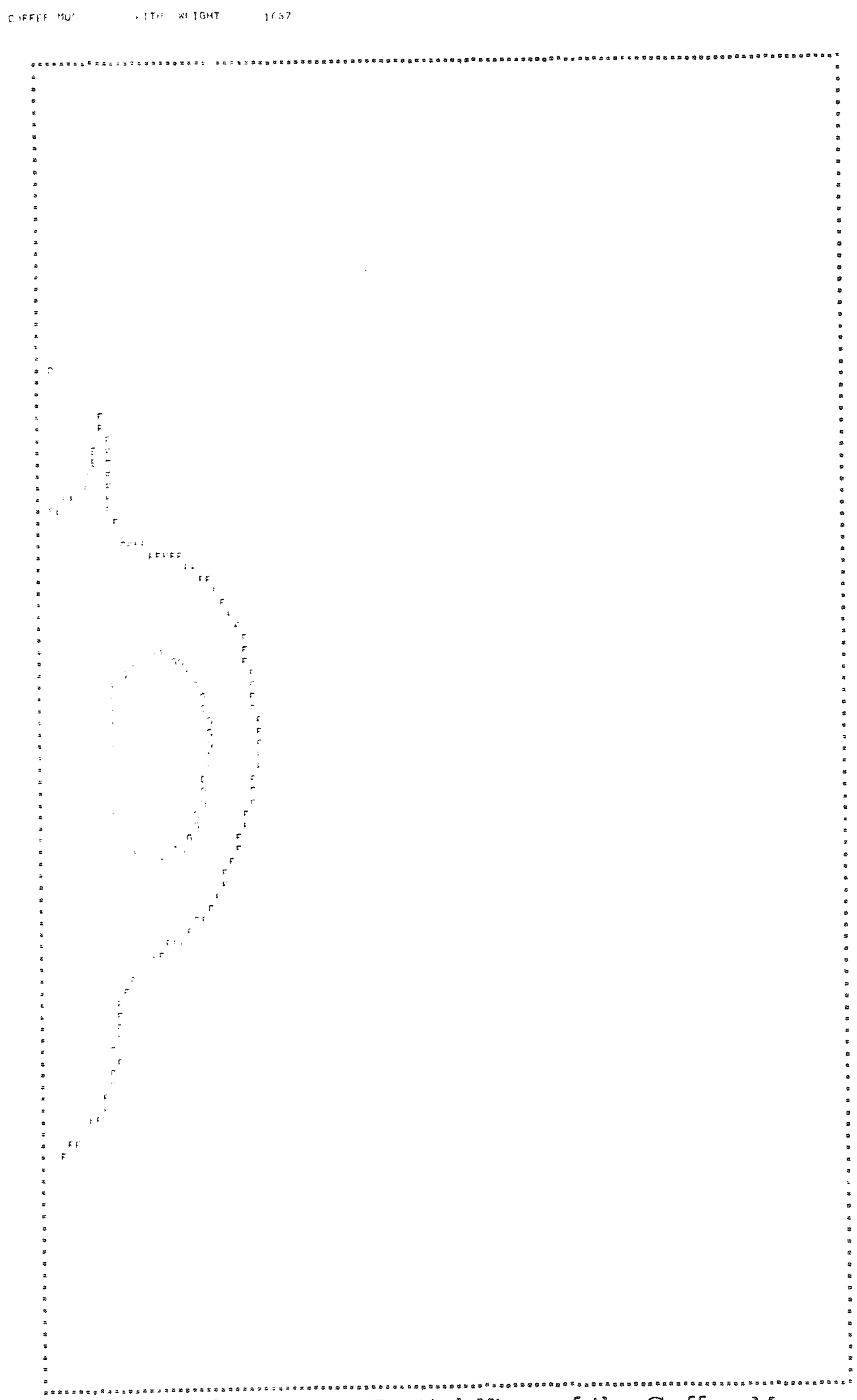


Figure 32. Second Partial View of the Coffee Mug

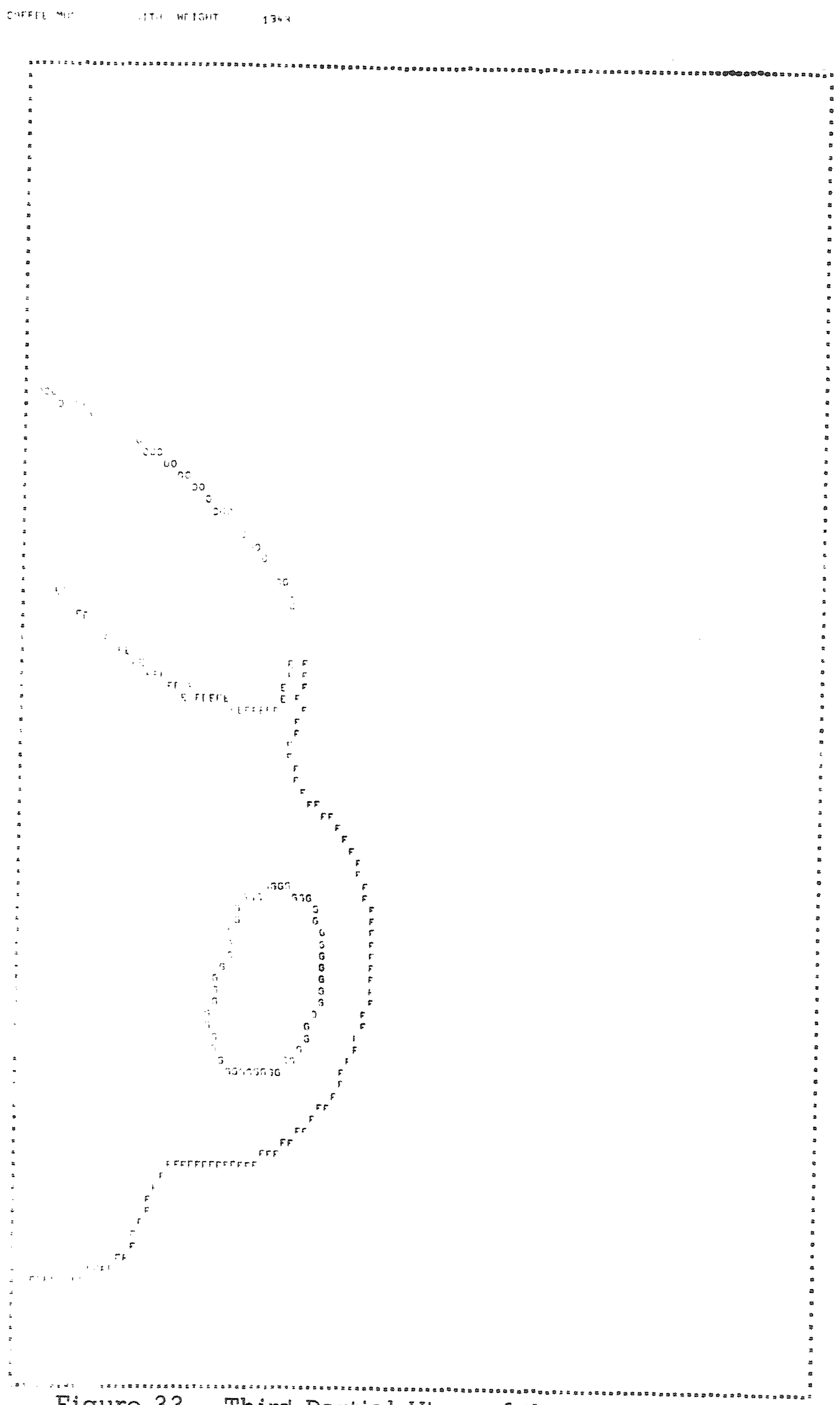


Figure 33. Third Partial View of the Coffee Mug

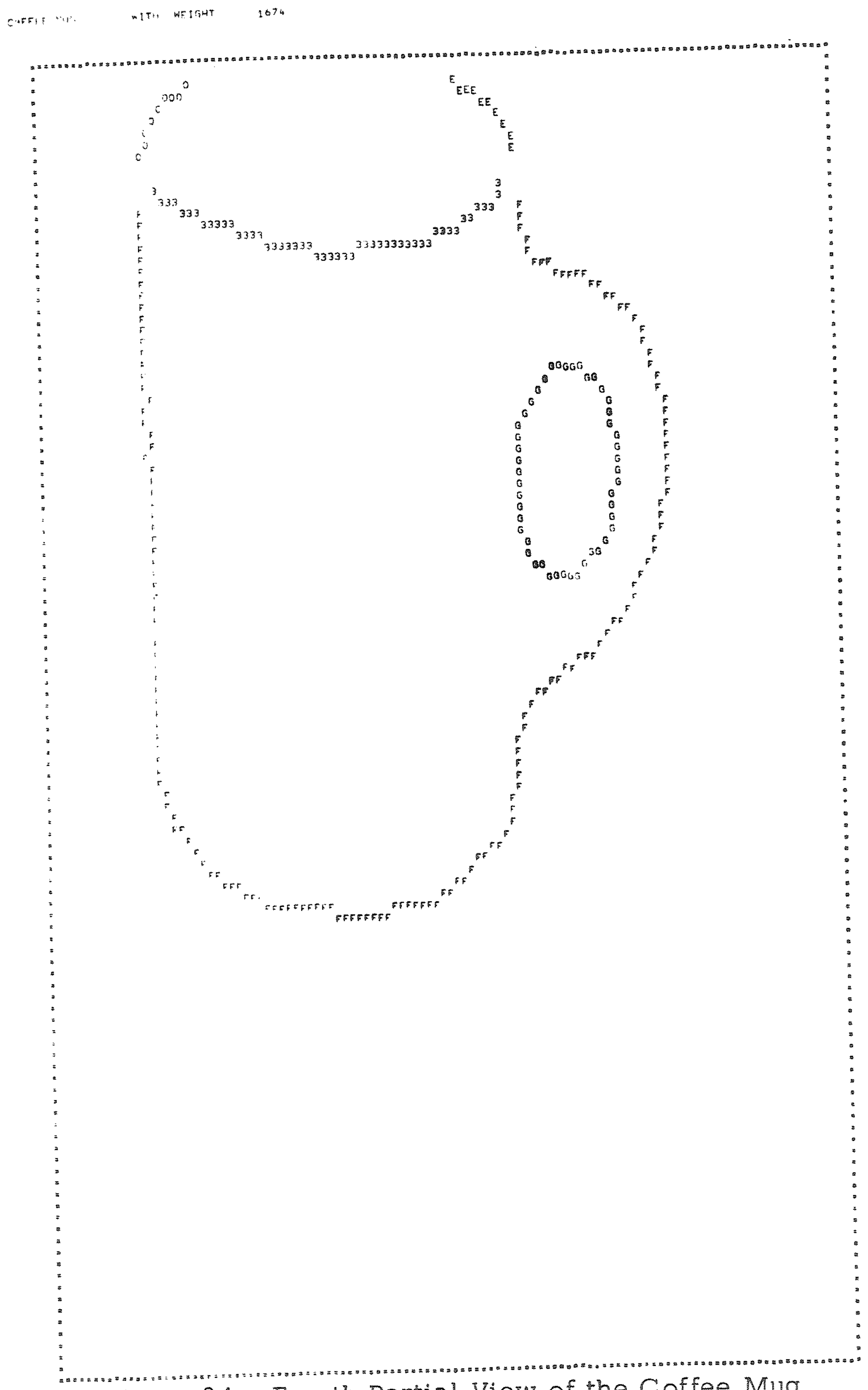


Figure 34. Fourth Partial View of the Coffee Mug

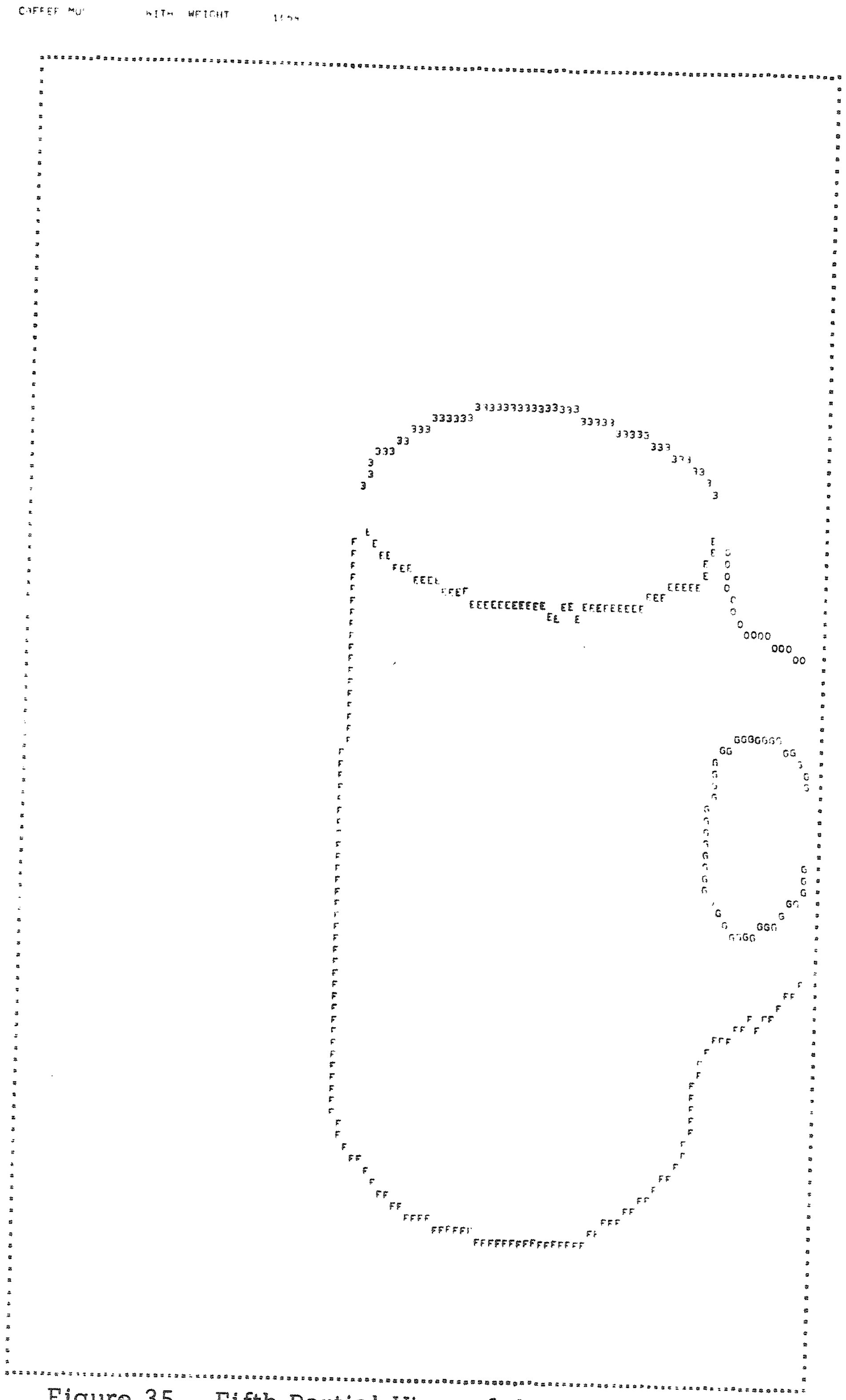


Figure 35. Fifth Partial View of the Coffee Mug

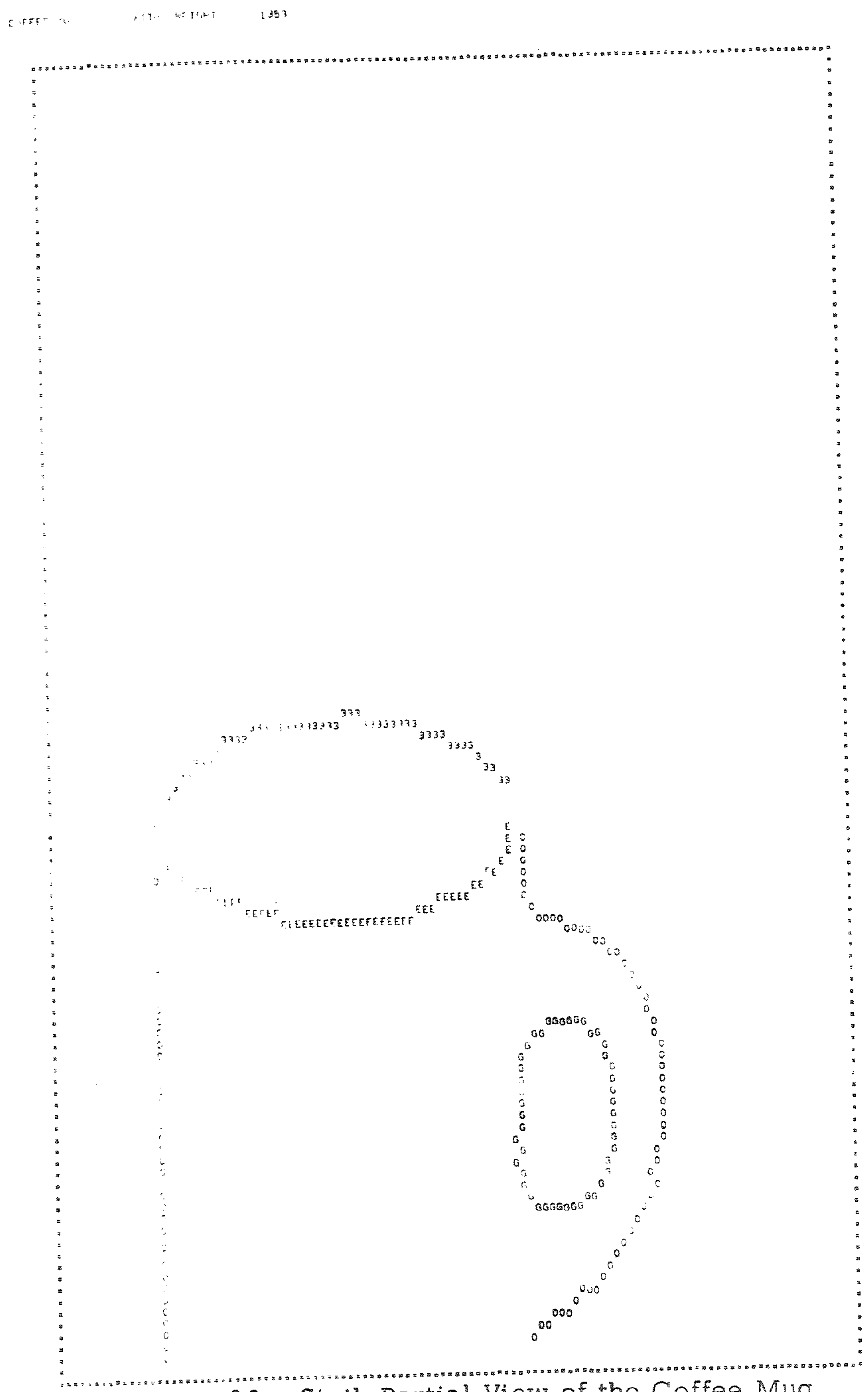


Figure 36. Sixth Partial View of the Coffee Mug

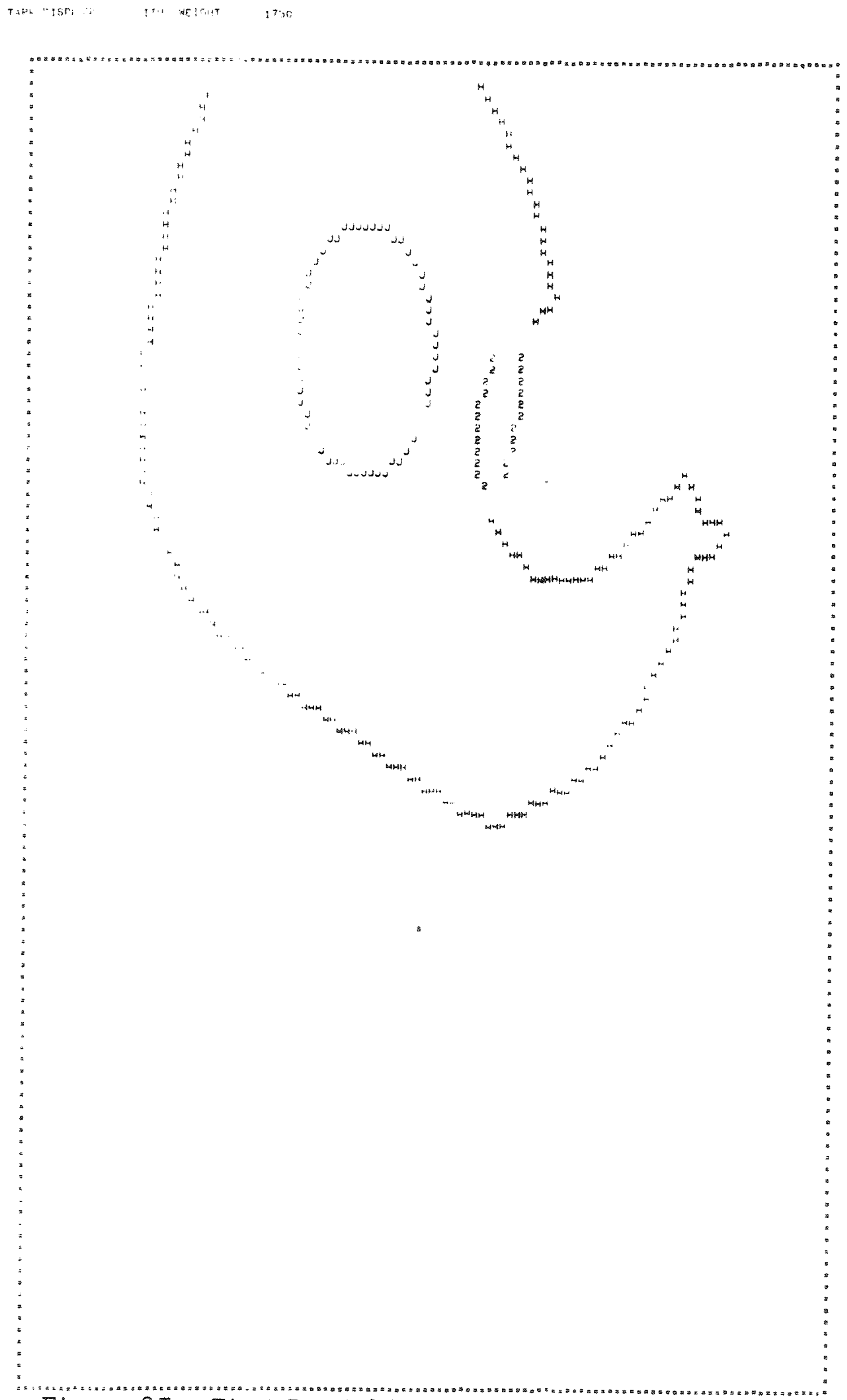


Figure 37. First Partial View of the Tape Dispenser

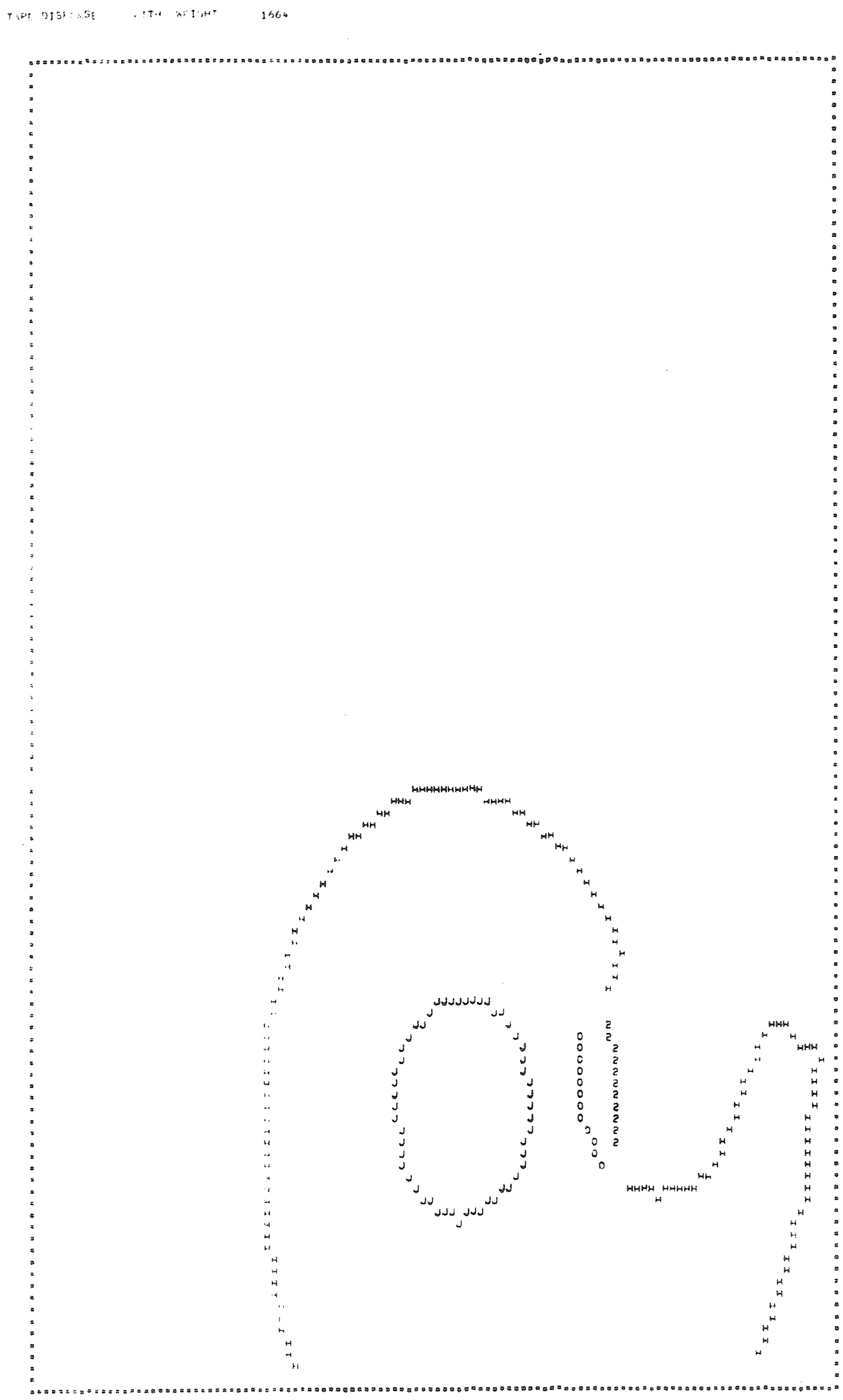


Figure 38. Second Partial View of the Tape Dispenser

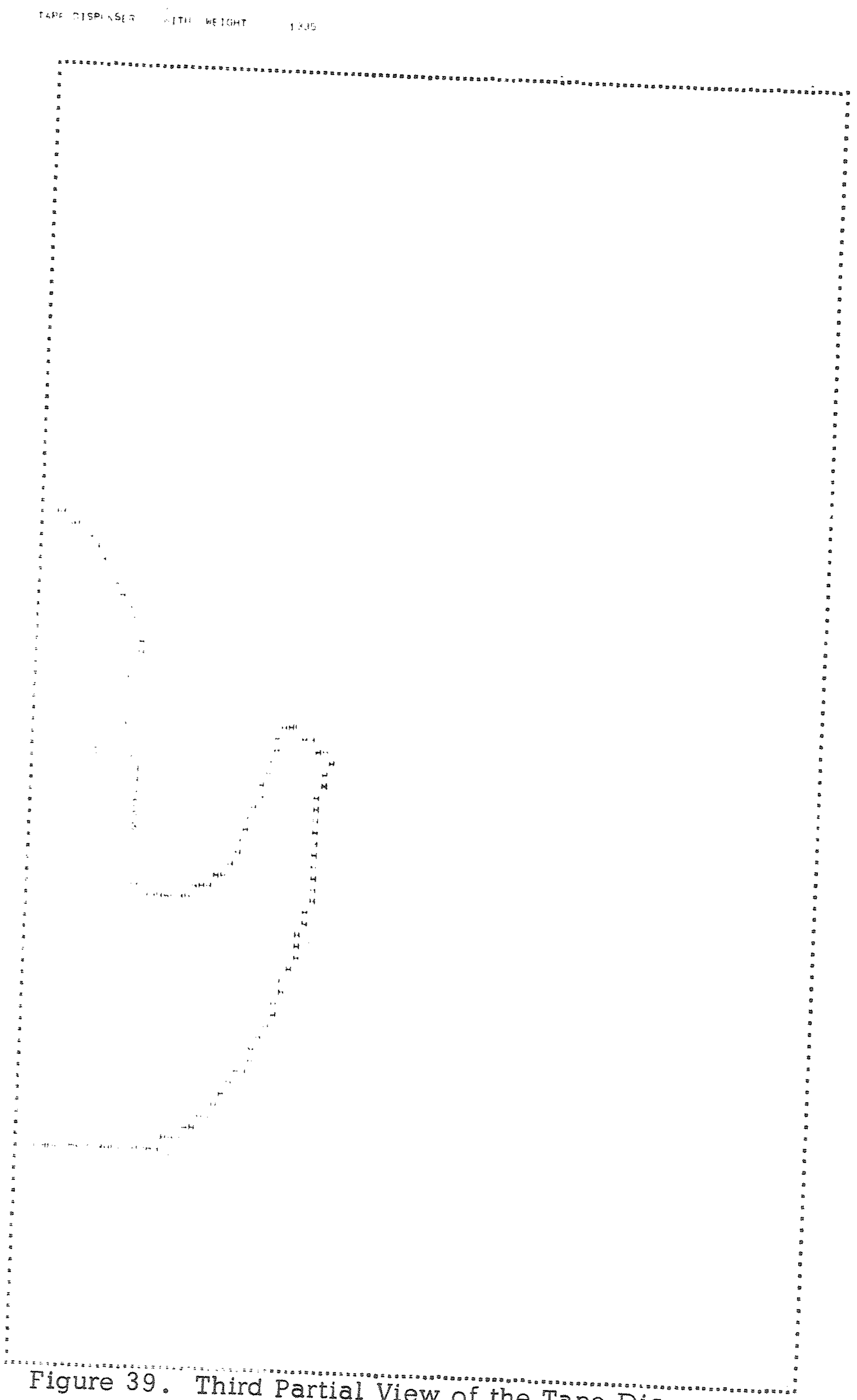


Figure 39. Third Partial View of the Tape Dispenser

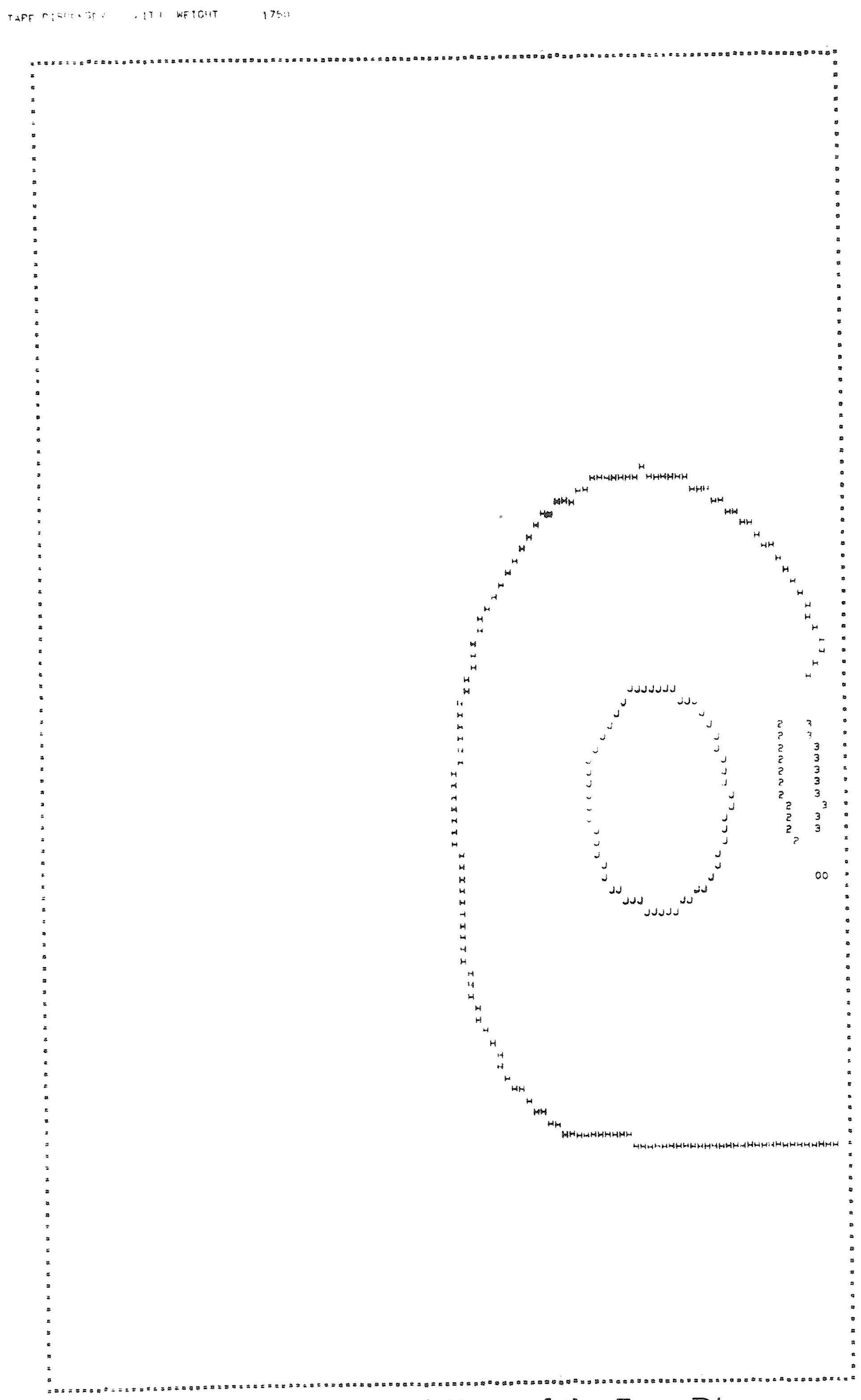


Figure 40. Fourth Partial View of the Tape Dispenser

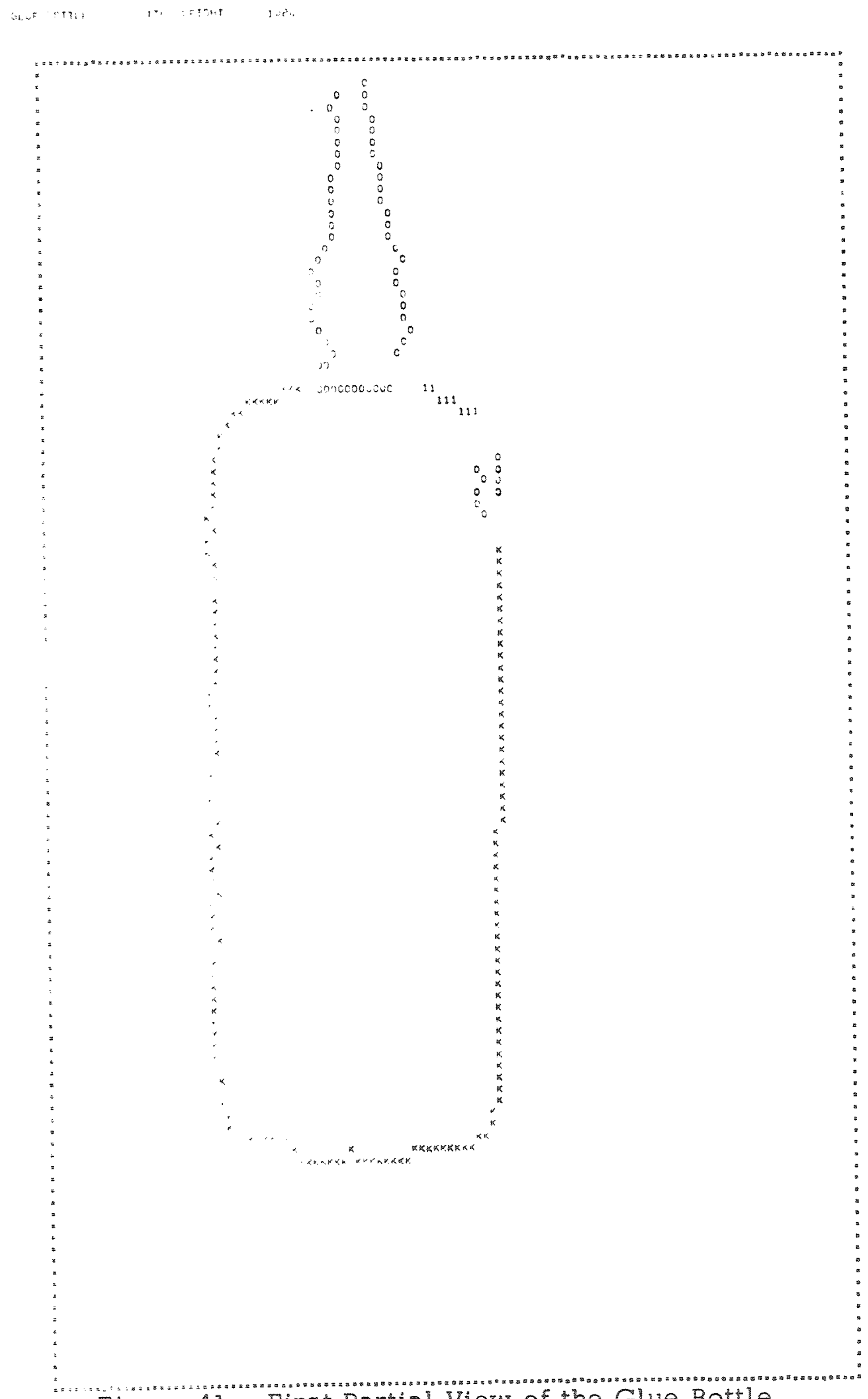


Figure 41. First Partial View of the Glue Bottle

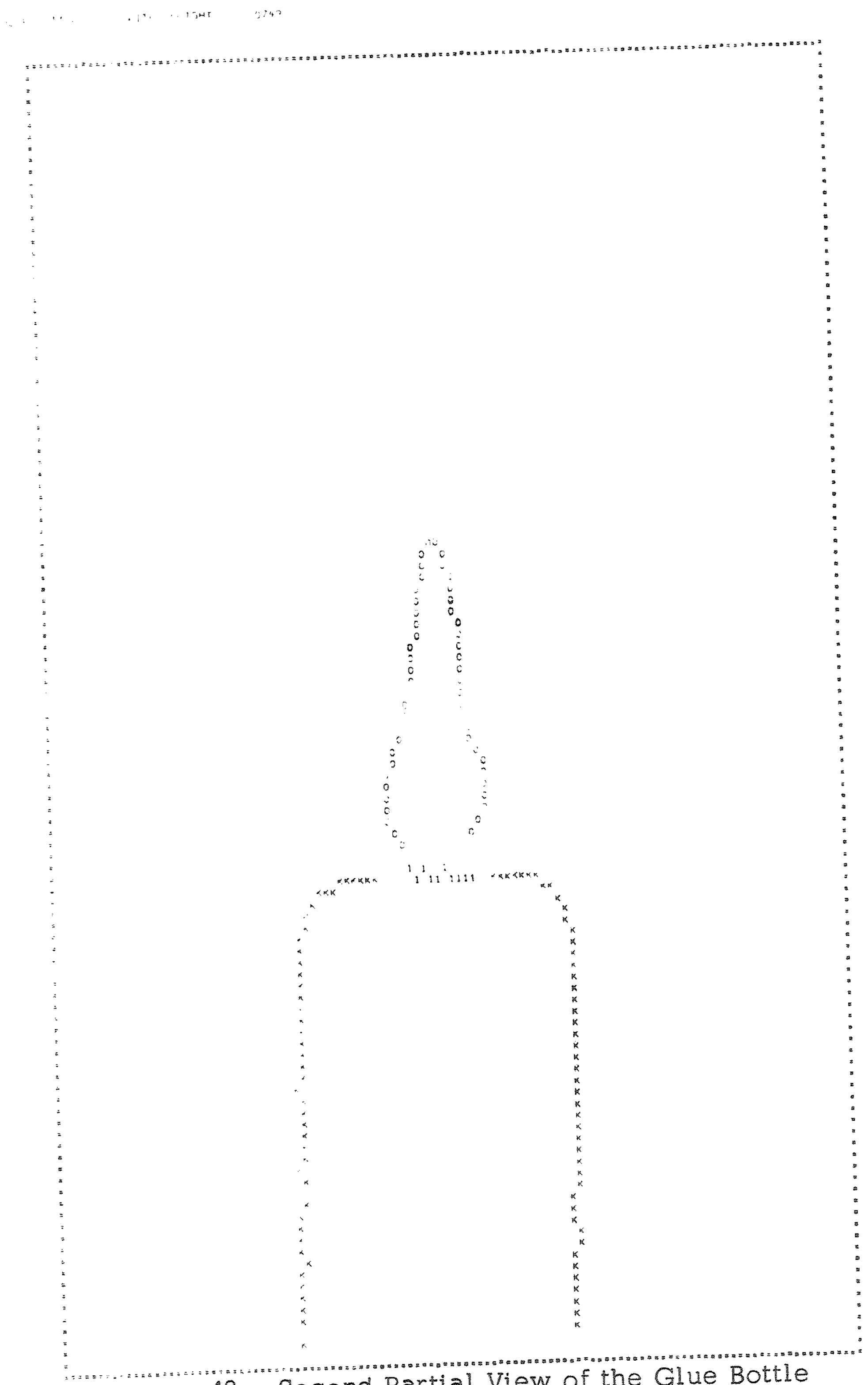


Figure 42. Second Partial View of the Glue Bottle

SCISSORS WITH WEIGHT 1/50

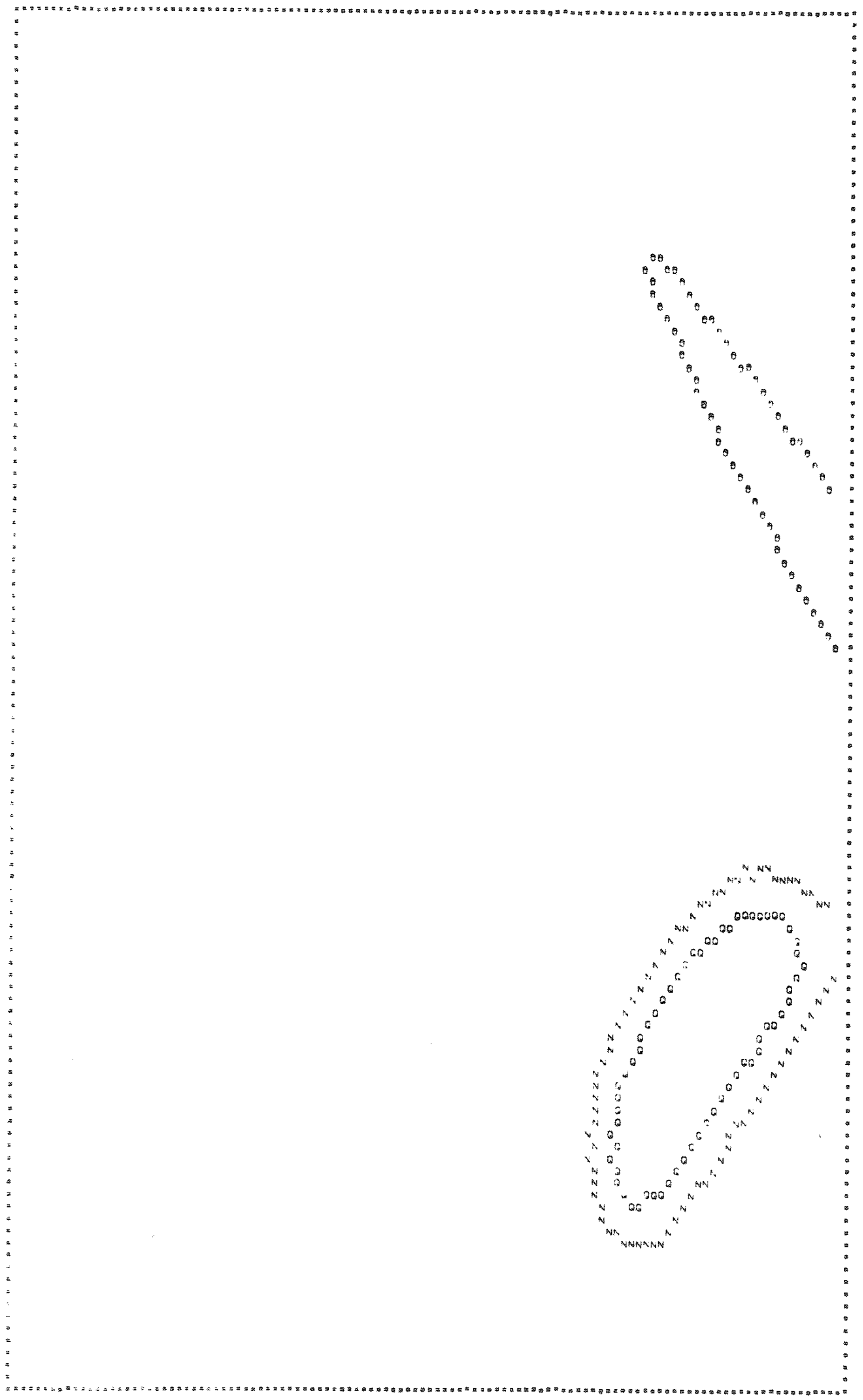


Figure 43. First Partial View of the Scissors

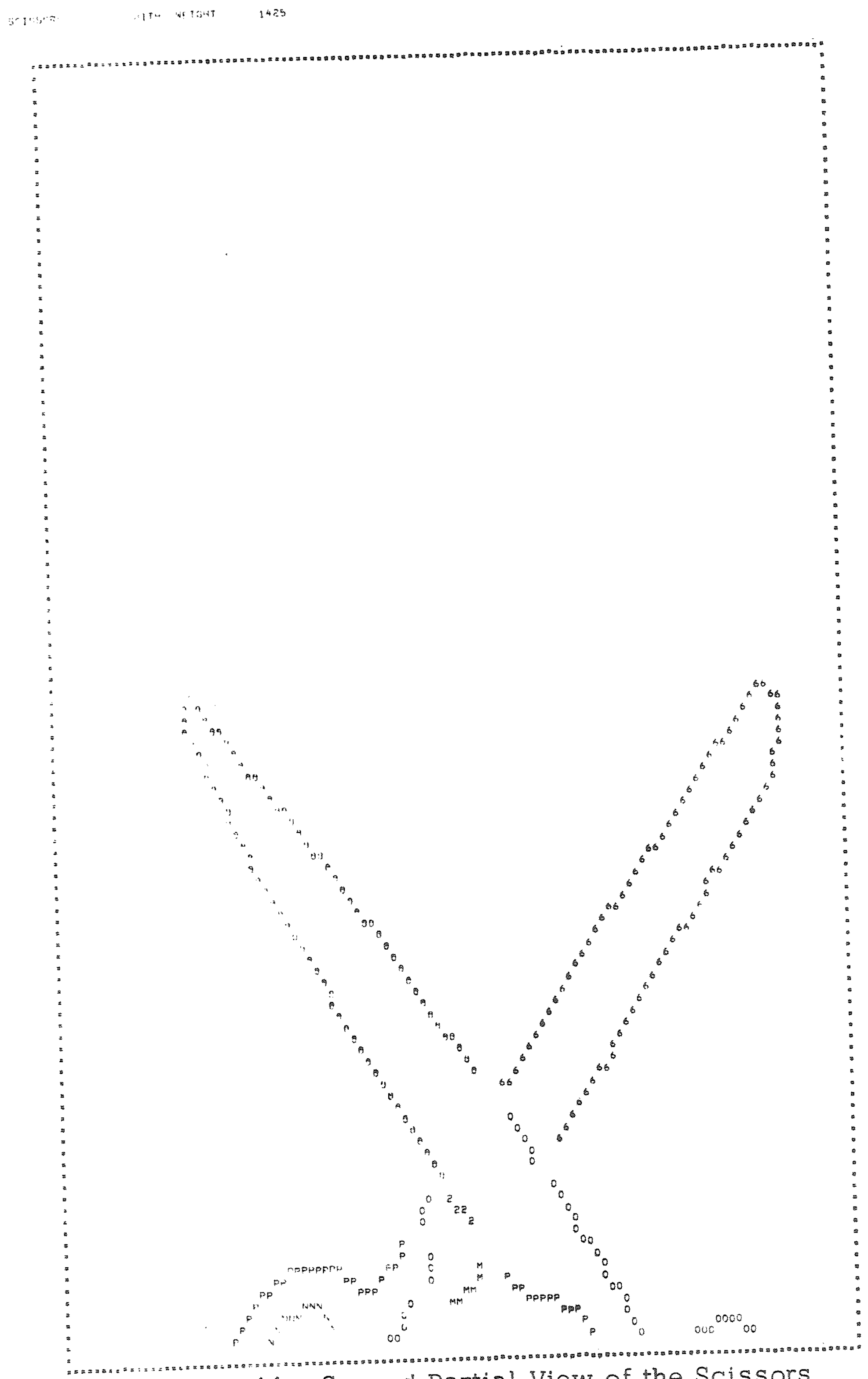


Figure 44. Second Partial View of the Scissors

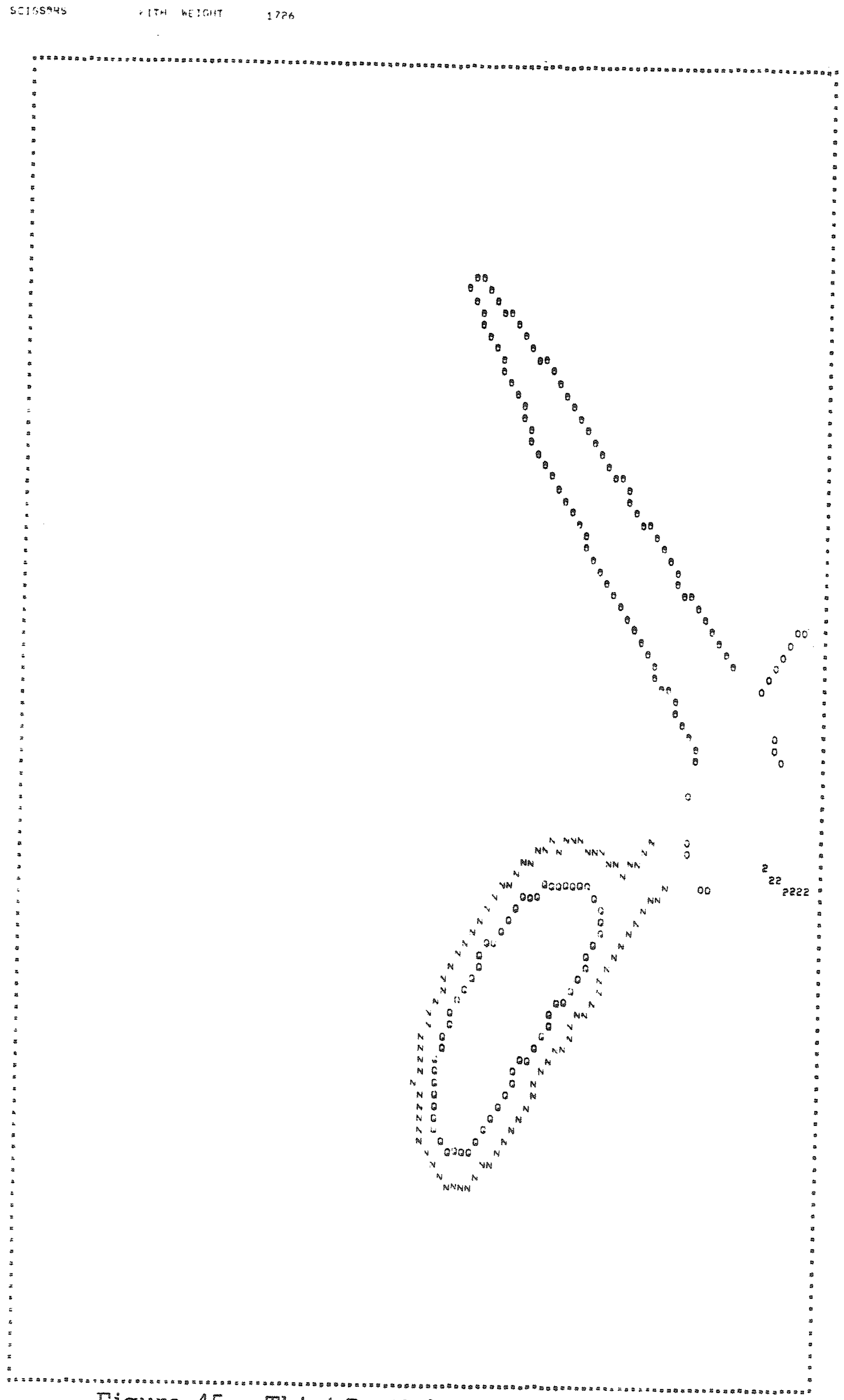


Figure 45. Third Partial View of the Scissors

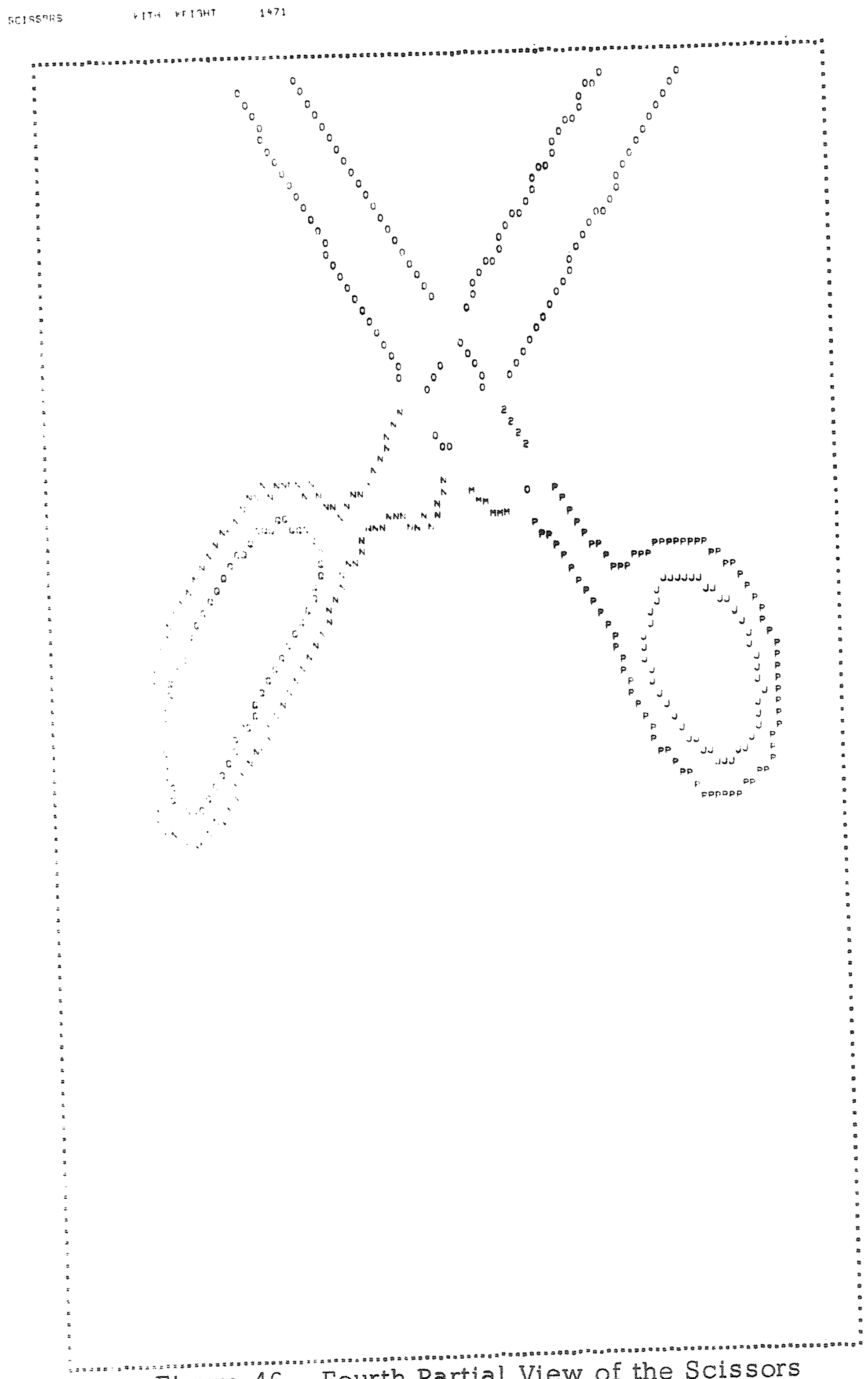


Figure 46. Fourth Partial View of the Scissors

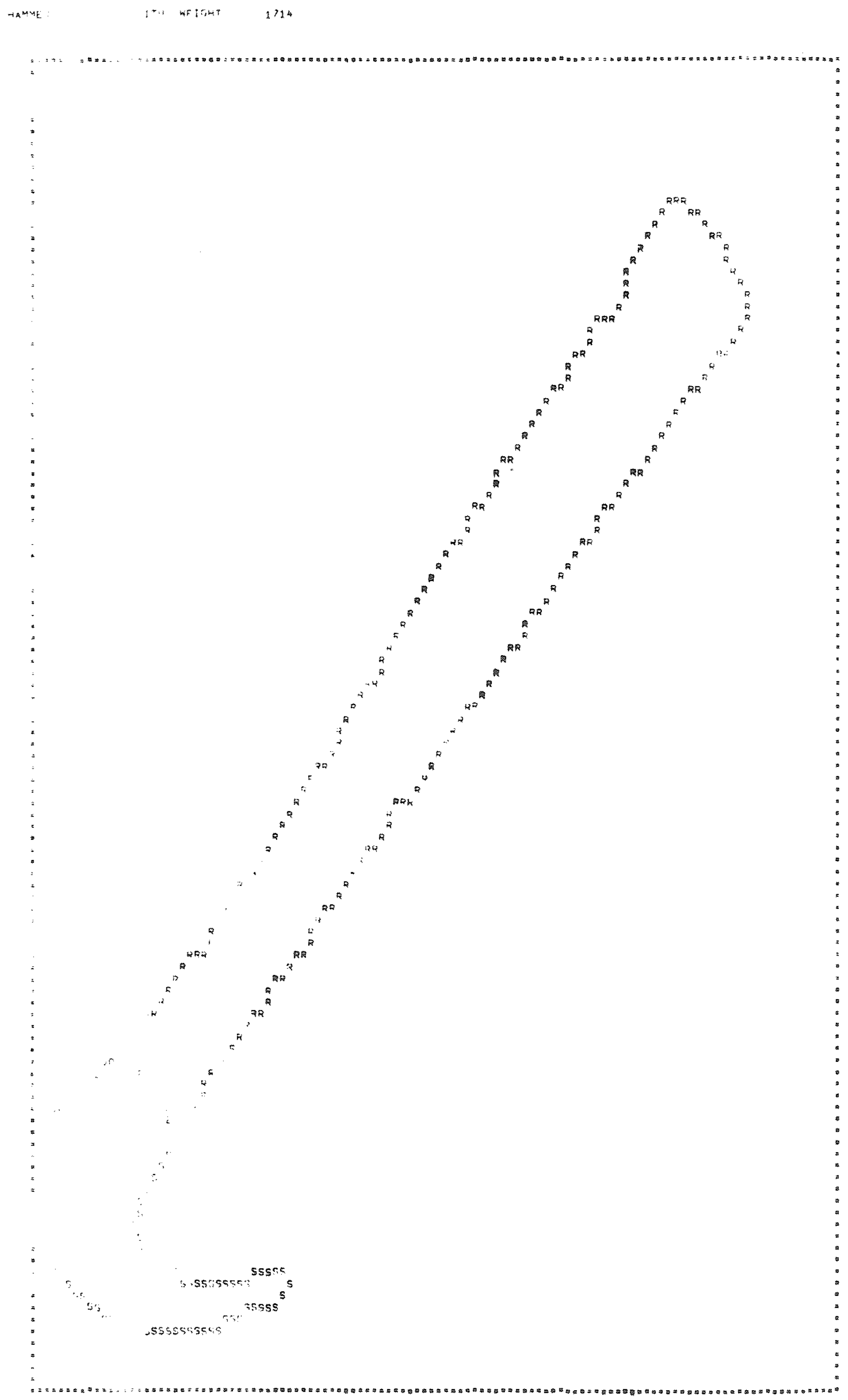


Figure 47. First Partial View of the Hammer

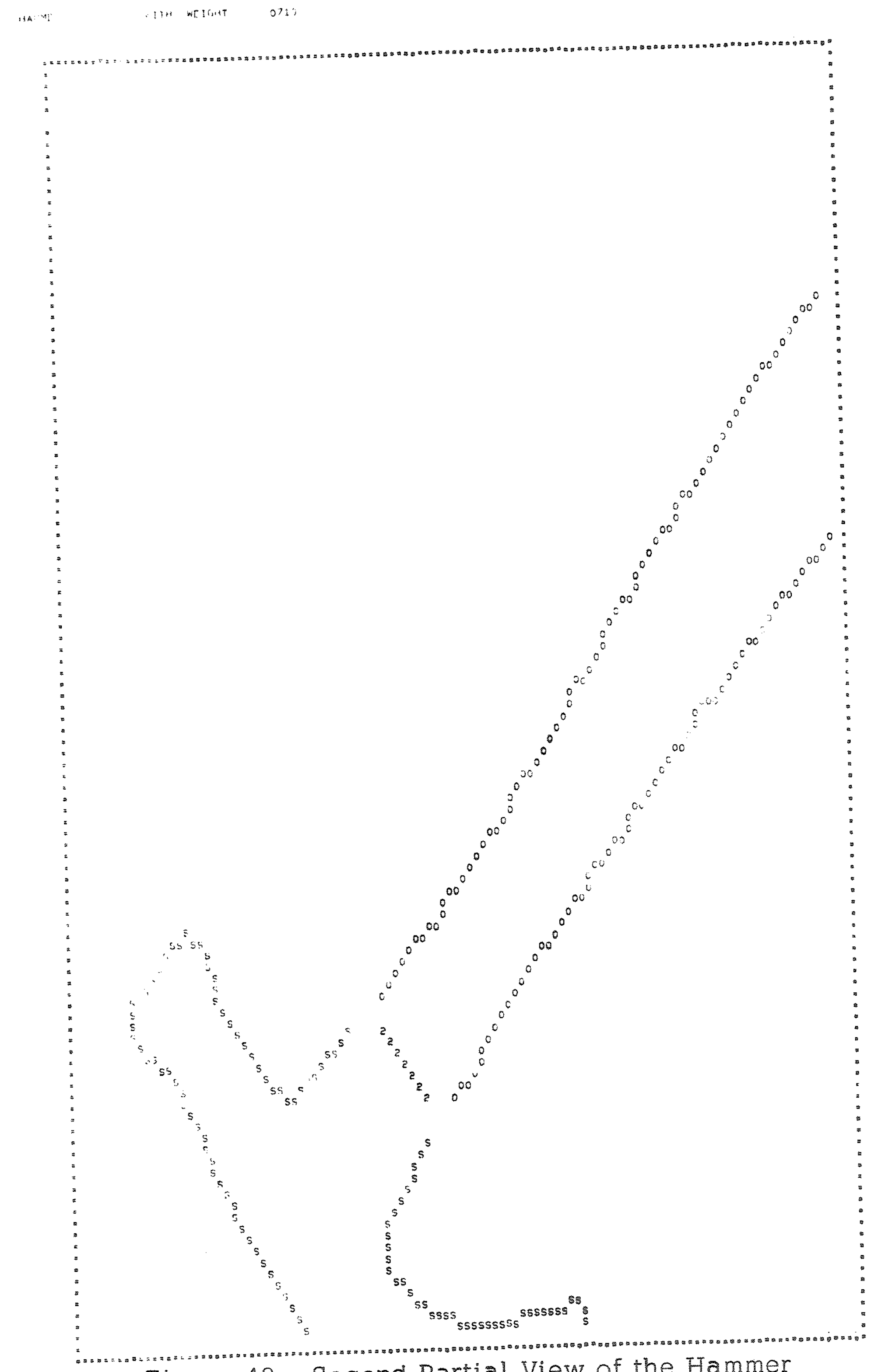


Figure 48. Second Partial View of the Hammer

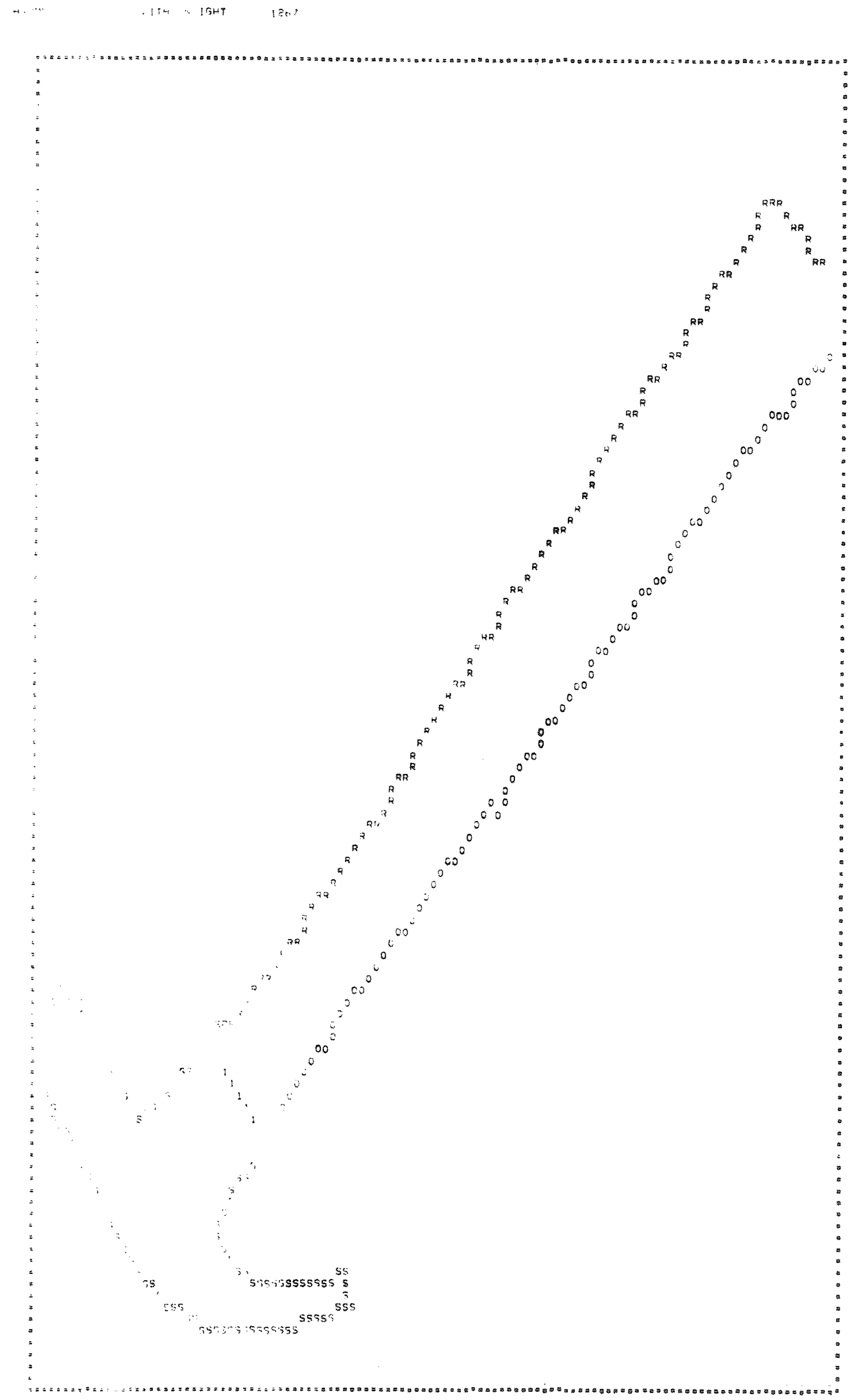


Figure 49. Third Partial View of the Hammer

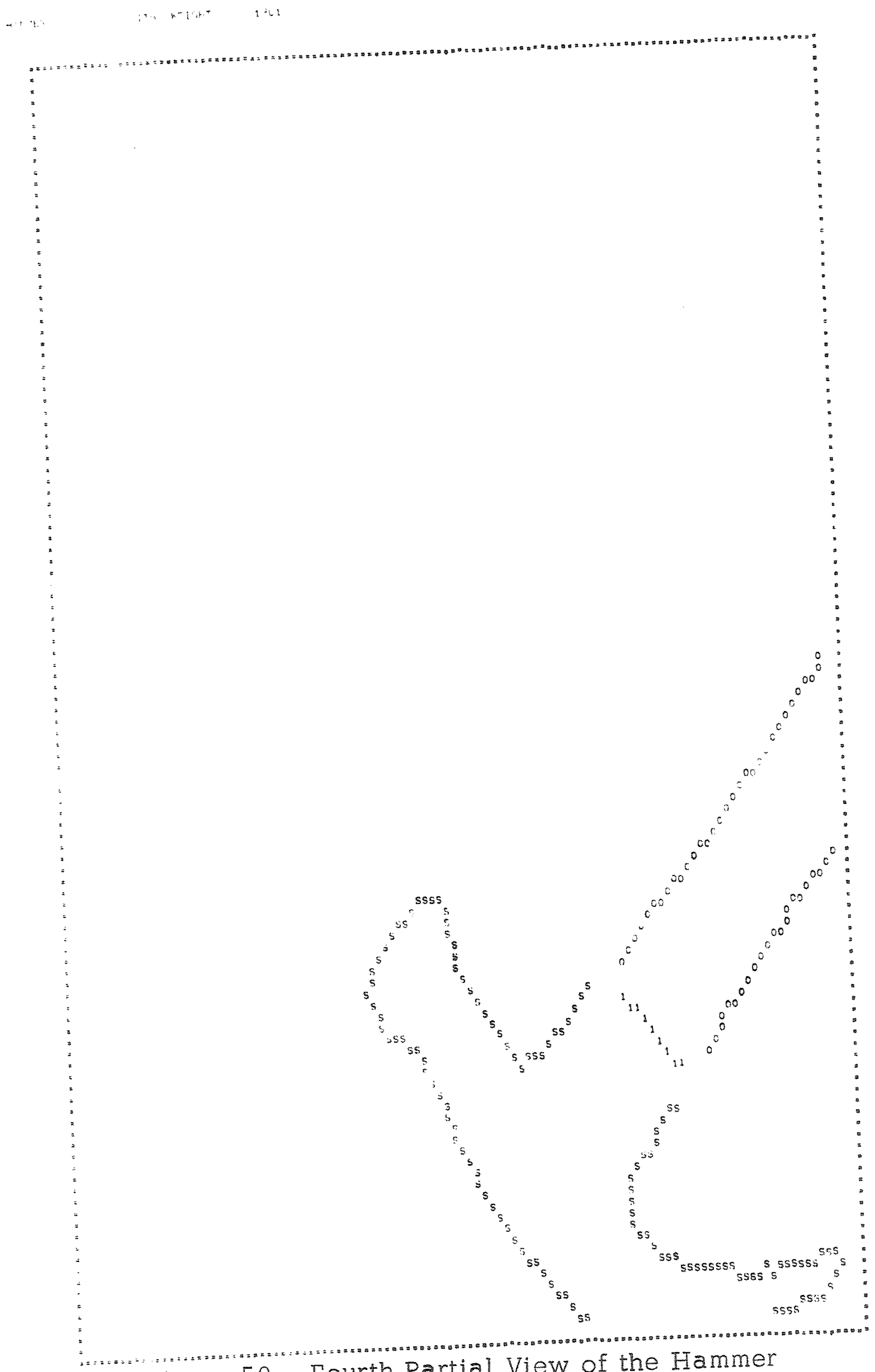


Figure 50. Fourth Partial View of the Hammer

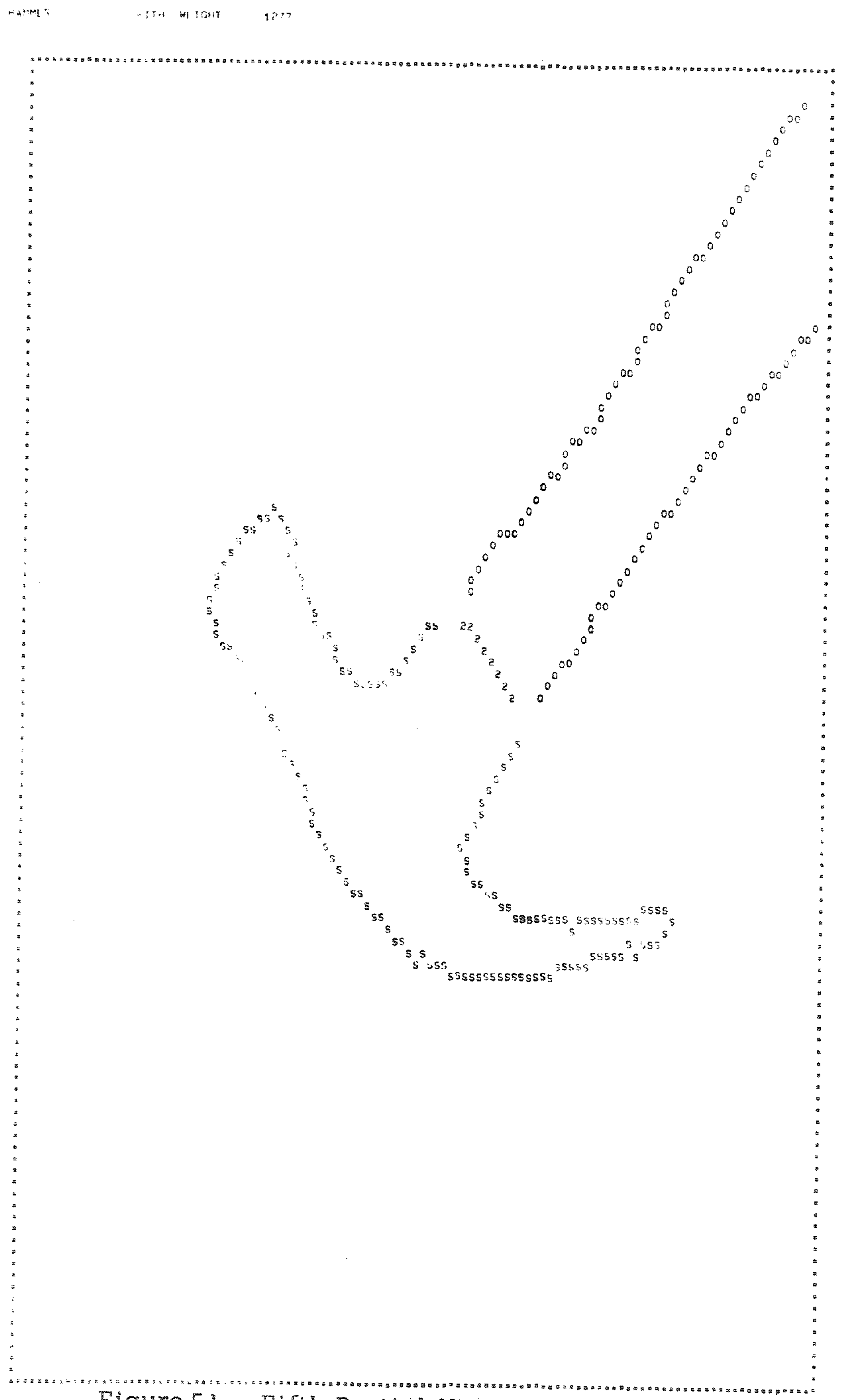


Figure 51. Fifth Partial View of the Hammer

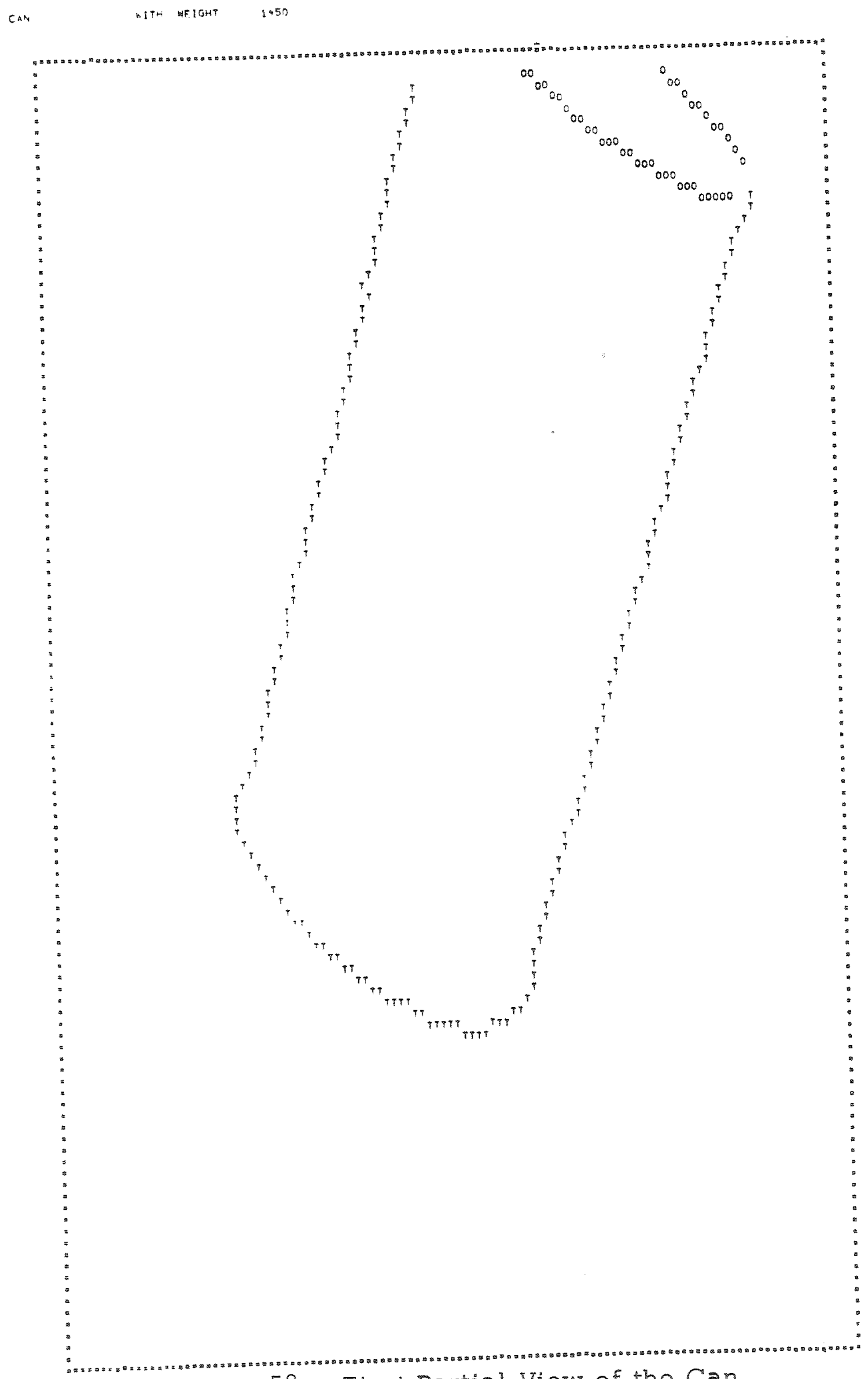


Figure 52 . First Partial View of the Can

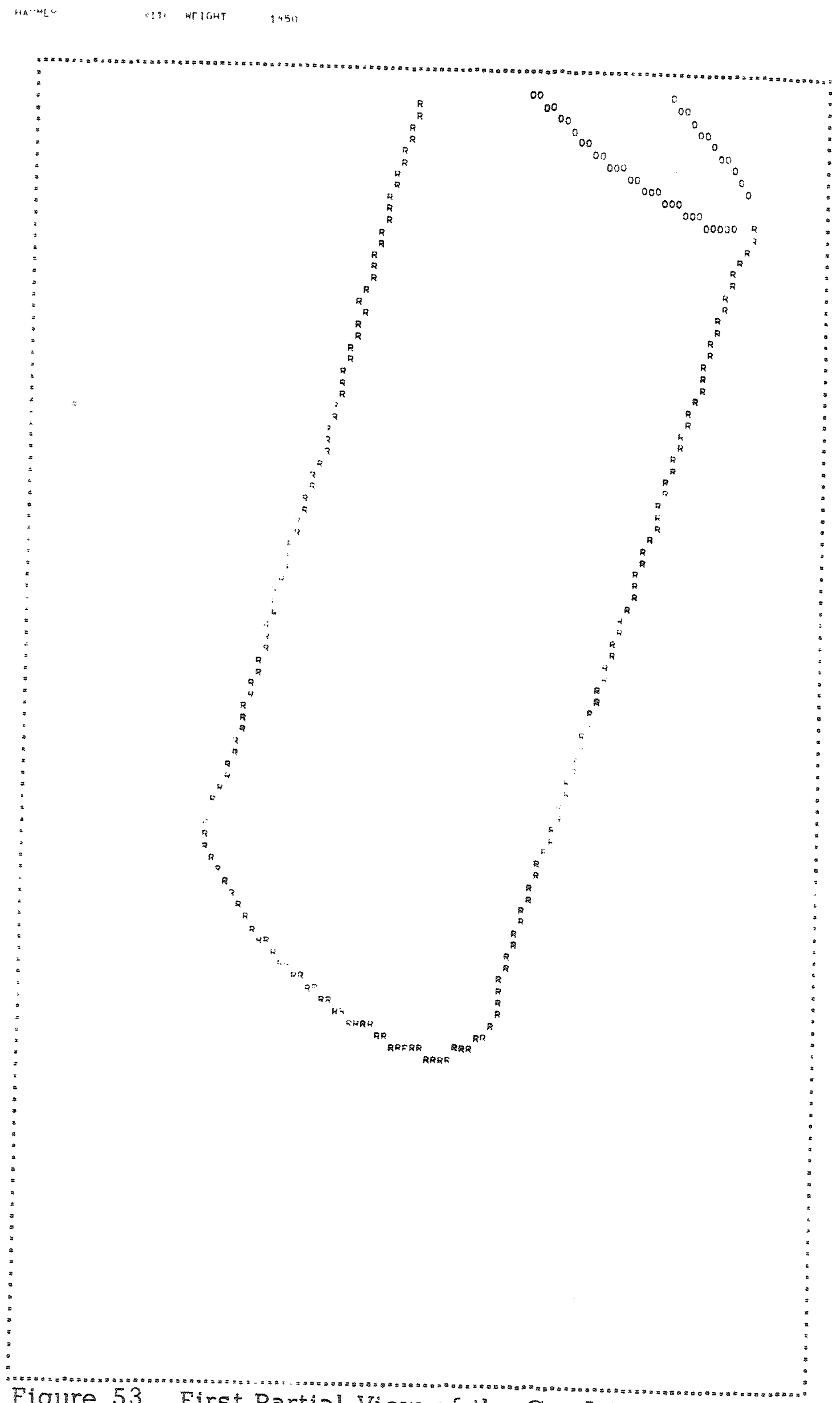


Figure 53. First Partial View of the Can Identified as the Hammer

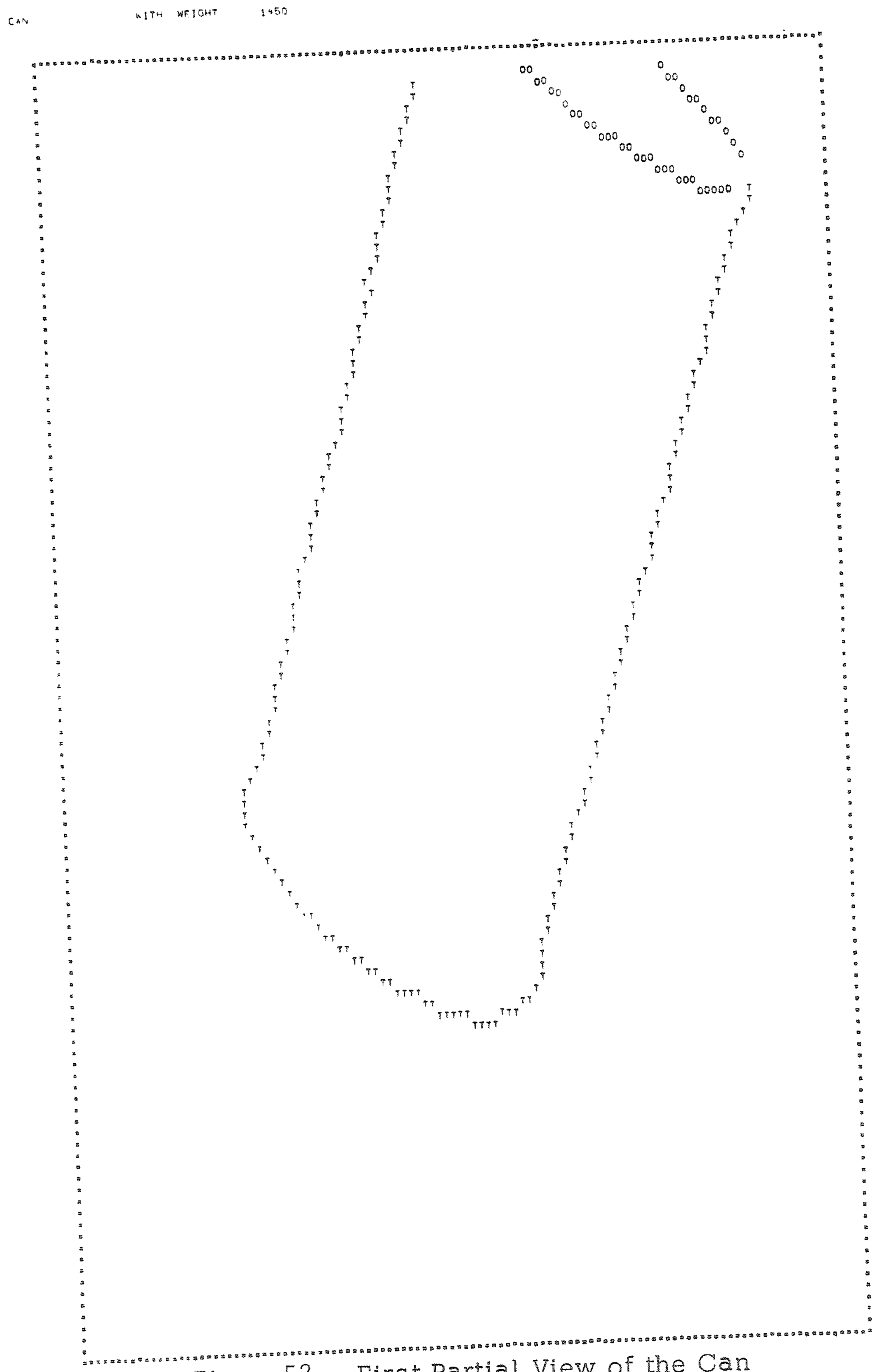


Figure 52 . First Partial View of the Can

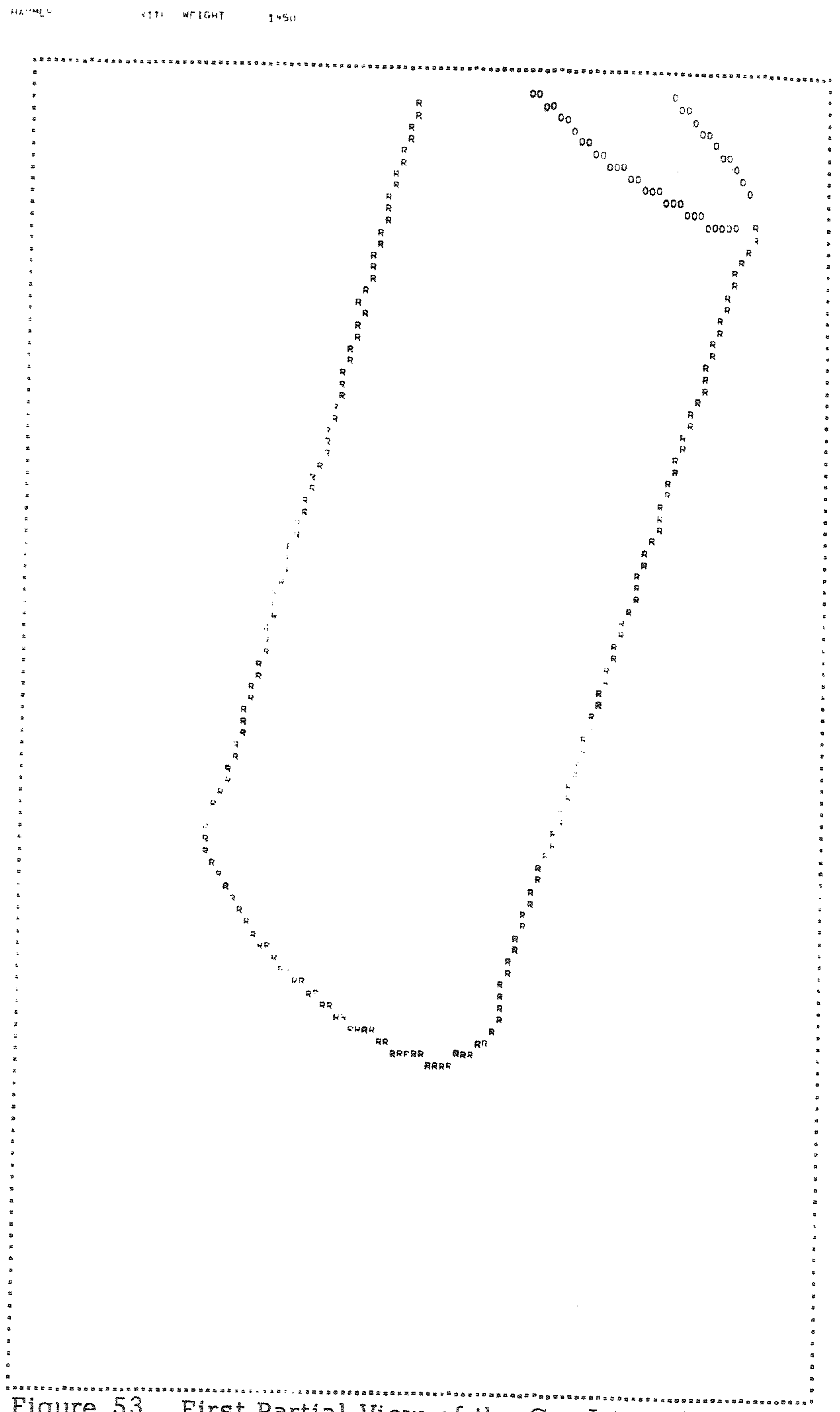


Figure 53. First Partial View of the Can Identified as the Hammer

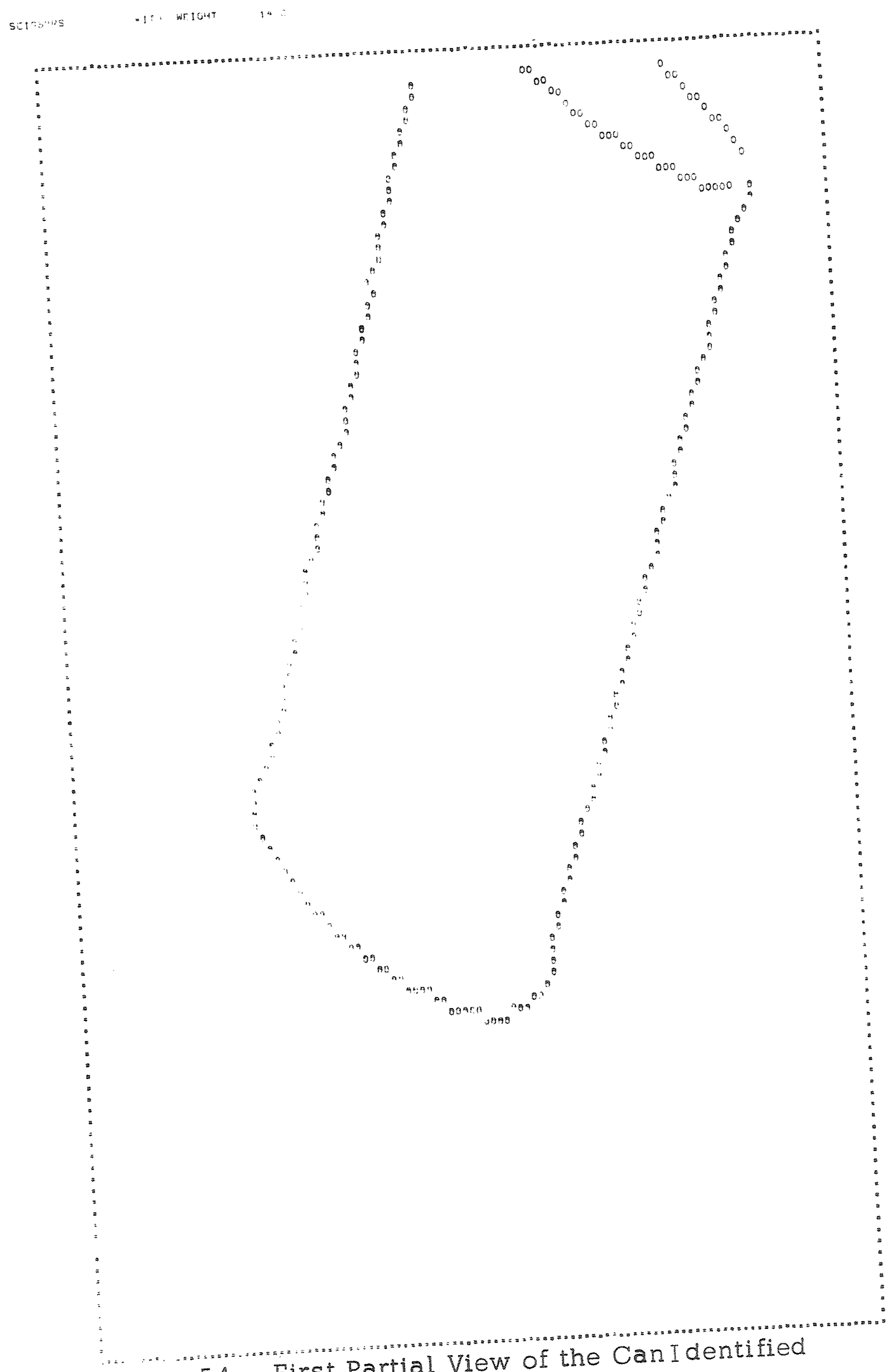


Figure 54. First Partial View of the Can Identified as Scissors

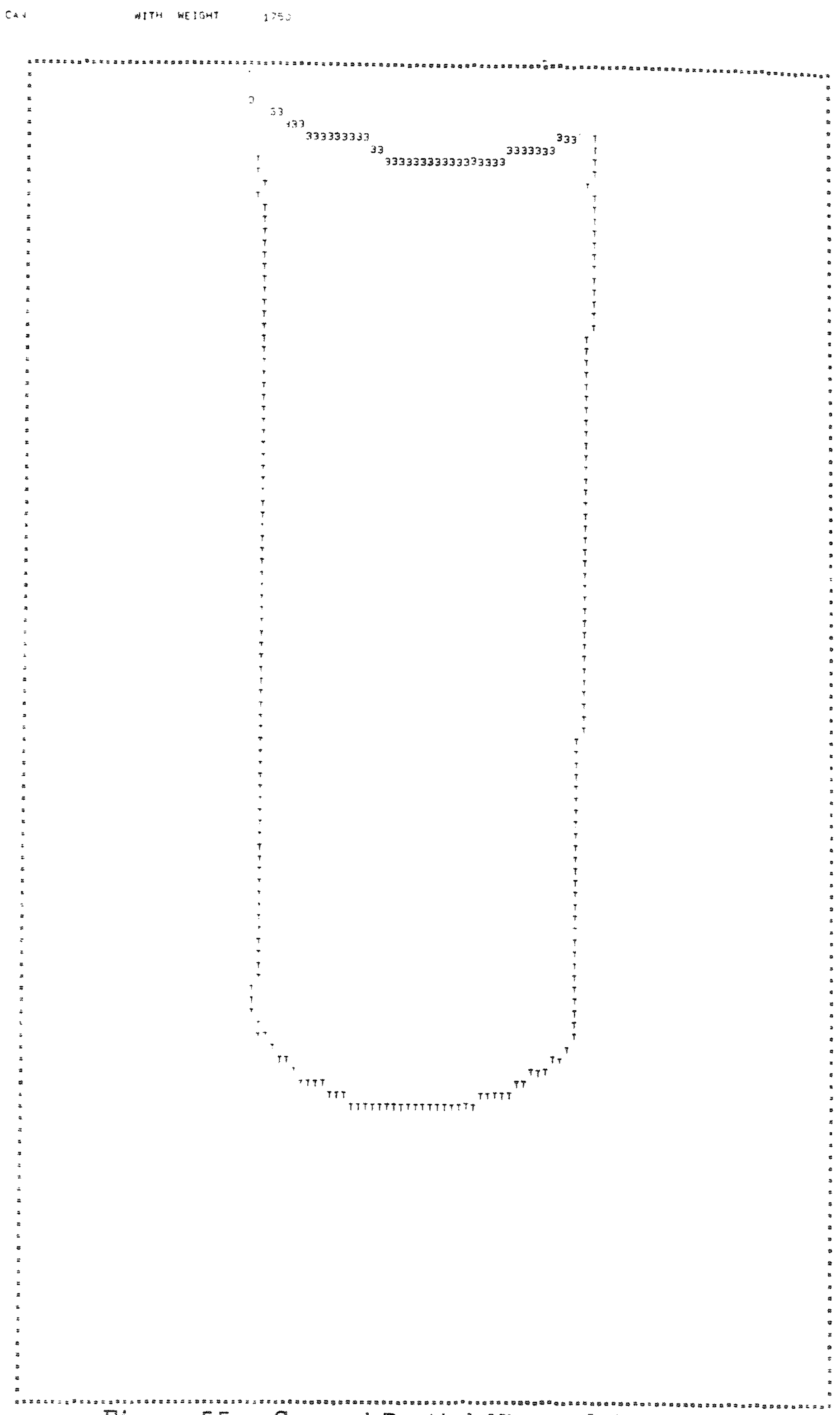


Figure 55. Second Partial View of the Can

CHAPTER 5

CONCLUDING REMARKS

An object is placed before the image dissector camera and an edge array is formed. For each edge in the scene, a list of the x-y coordinates of the points of the edge is formed. This list is transformed into a smoothed-expanded-compensated-Freeman code, and straight lines are fitted to the points of the SECFC data. After all the edges in the scene have been described in this manner, the object in the scene is either learned or recognized. If the object is to be learned, a name is found for each edge of the object. A description of the object is constructed from the names of the edges of the object and placed on the library of known objects. If the object is to be recognized, possible names for each edge of the partial view are found. The possible names of an edge are compared to the names of the edges in the description of each known object. The names, weights, and printouts of the three best matches are outputted.

From the data of Table 2 and from Figs. 20 through 55, it is evident that the learning-recognition algorithm performs well. The performance of the recognition algorithm was a function of how many edges were in the scene, as would be expected. When the whole view of the object was being recognized, the object was correctly identified and usually had the maximum possible weight. It has not been possible to show the edge arrays for all the partial views, but just from the edge arrays of the library views, it is evident that the algorithm was able to use input data which was far from perfect.

There are some interesting observations about this algorithm which suggest some possible extensions of the algorithm. Although a rectangular window was used in this algorithm, there is nothing in the structure of the algorithm to prevent the use of other shapes of windows. In fact, it is not even necessary that the window be connected; that is, there could be two or more disjoint windows of different sizes and shapes. One possible extension would be to have the outlines of other occluding objects form the boundaries of the windows. The present algorithm uses some of the contextual information available in the scene by requiring the preservation of relative lengths of edges when the partial view is matched to an object's description. An obvious next step would be the use of node information. One possibility would be to require the preservation of edge intersections when matching the partial view to an object's description. For example, if two edges, say A and B, intersect in the partial view, then require that the equivalent edges, A' and B', intersect in the object's description. When a partial view was matched to a known object's description, a perfect match was not required. This flexibility allowed the algorithm to name the object correctly without having to recognize all the unknown edges. The algorithm was specifically designed to deal with objects with curved surfaces. This curvature, along with uneven lighting, introduces false edge segments, noise surfaces, and sometimes even false edges. The algorithm can eliminate the false edge segments and noise surfaces. But a false edge causes the algorithm to give erroneous answers usually indicated by the best object having a very low weight. A very desirable extension would be to develop an algorithm that works correctly even with this erroneous edge information. One method to accomplish this would be to use an algorithm which included some type of node information to form an ordered list of possible objects. Then do a type of top-down search, where one would assume the unknown object was the first object on the ordered list. With this assumption, edges in the partial view that were not matched would be examined again. This time they would be compared as partial edges to edges suggested by the node relationships in the object's description. If this process fails to create a high enough weight for the first object on the list, the second object would be examined, etc., until a satisfactory weight is reached or the list of object is exhausted. These are some of the possible extensions of the present work. It is hoped that other extensions will occur to the reader.

APPENDIX A

This appendix describes the data structure and the algorithm of Step A. In Step A, a list of the x-y coordinates of the chain-connected points of the edge segments is formed. An edge segment library is created in which information about each edge is stored. The boundary of each surface is represented by an ordered list of the number of the edge segments that form the boundary. The input data to Step A is in the form of an edge array as shown in Fig. A.1. The points in this array may have three values: 1, 0, or -1. Initially, the points are either 1 or 0. The output of Step A is a set of four lists: the x-y coordinate list of the edge segments, the edge segment library, the edge list, and the surface list. The data structure of these lists is shown in Fig. A.2.

Step A

(AI) The desired closed perimeter is placed in the input edge array. The area contained within this perimeter is the portion of the scene used to recognize the object in the scene. The perimeter consists of chain-connected sets of points--examples of which are shown in Fig. 1 and Fig. A.1. If any points of the scene are outside the perimeter, they are erased from the edge array. The x-y coordinates of all the perimeter points are placed on a perimeter-point list. Throughout this algorithm, the square array of points shown in Fig. A.1 is used as the perimeter.

AII) In this step, the boundaries of the surfaces in a scene are examined to locate the edge segments. This procedure is done by following the boundary of each surface and noting the occurrence of edge segments and nodes. For each surface, the boundary is followed so that the surface being outlined is to the right of the edge segments as they are traced. When a new node is encountered while following the boundary of a surface, it is numbered and traced. The x-y coordinates of the

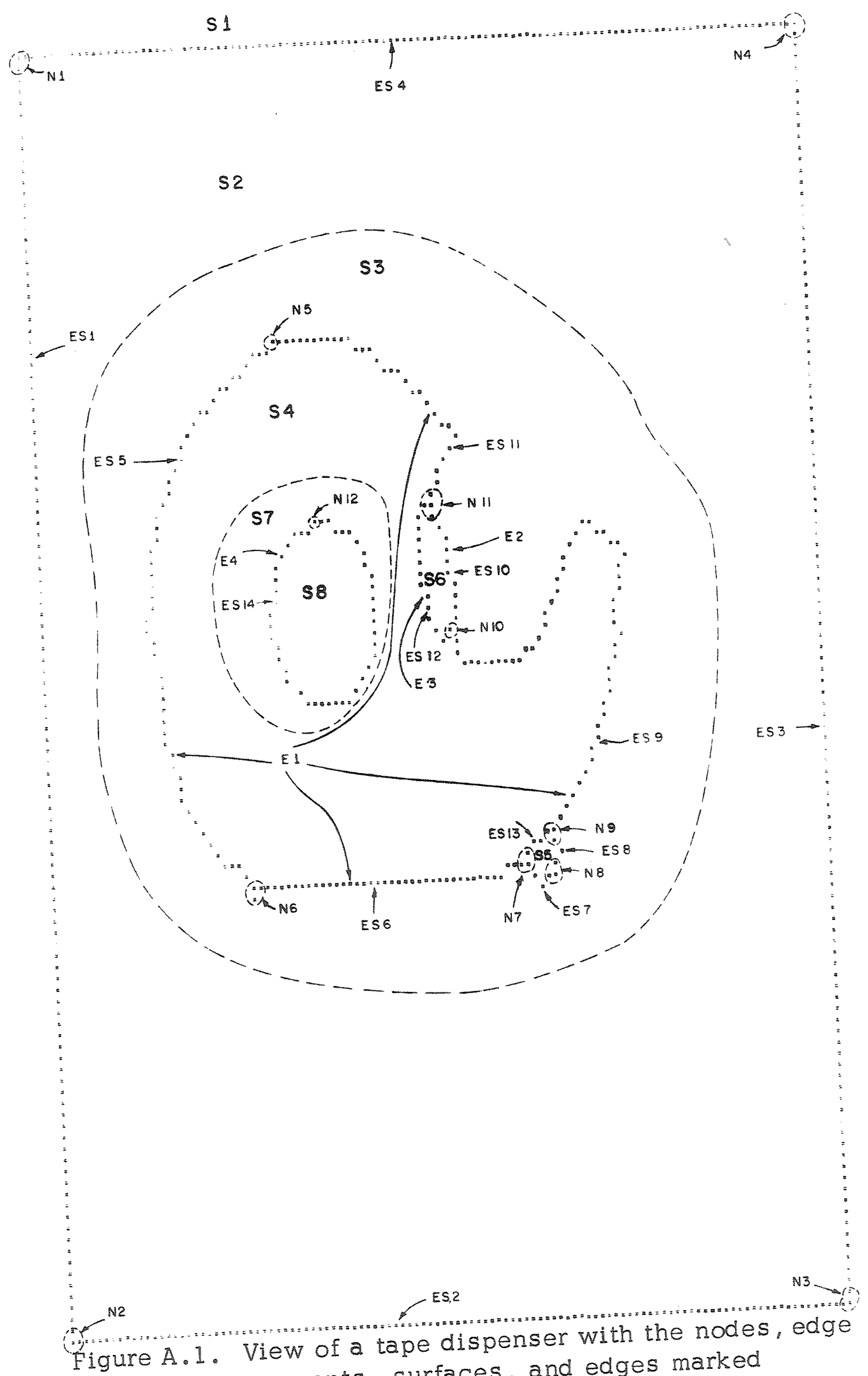


Figure A.1. View of a tape dispenser with the nodes, edge segments, surfaces, and edges marked

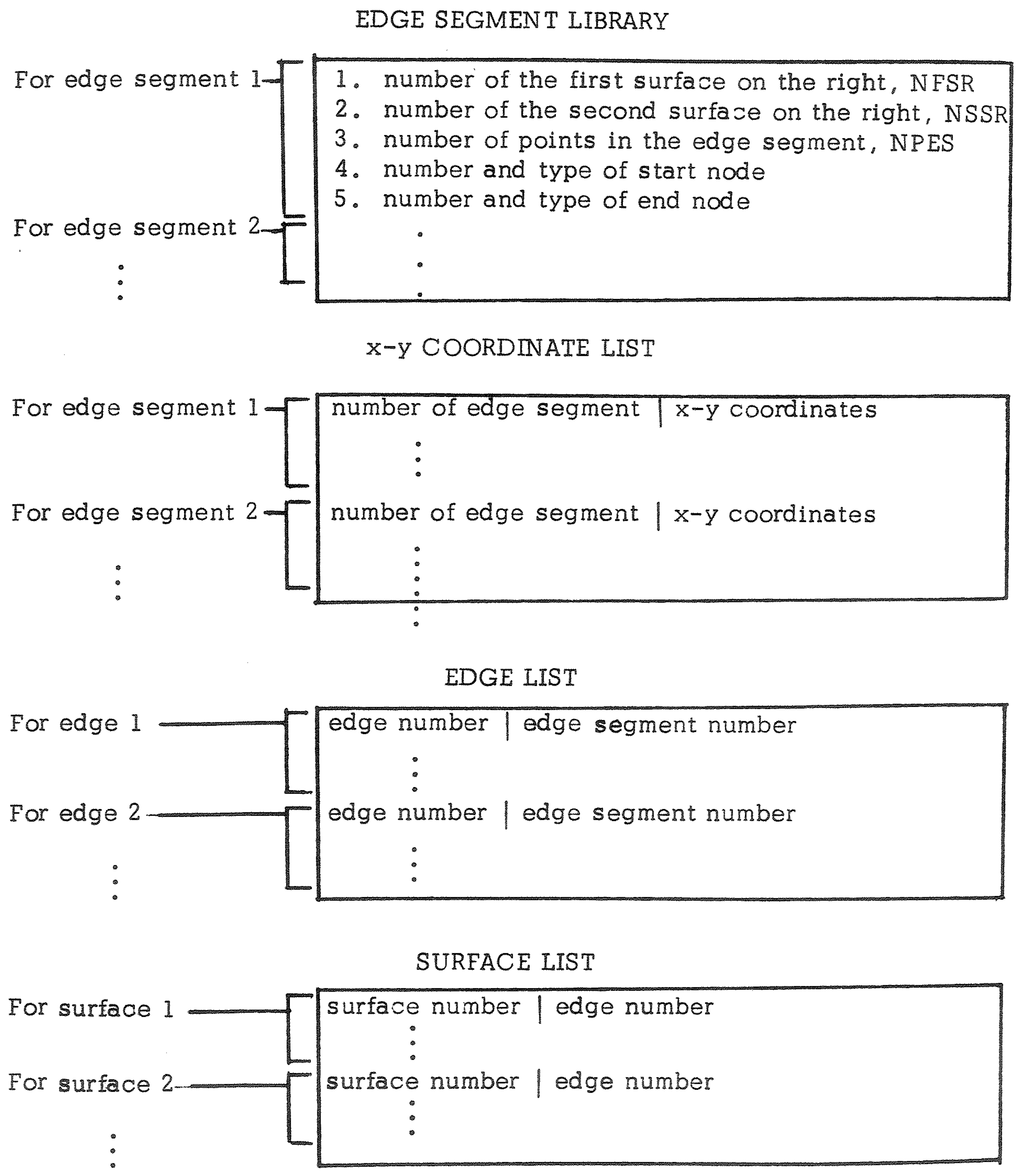

EDGE SEGMENT LIBRARY
For edge segment 1
1. number of the first surface on the right, NFSR
2. number of the second surface on the right, NSSR
3. number of points in the edge segment, NPES
4. number and type of start node
5. number and type of end node
For edge segment 2
x-y COORDINATE LIST
For edge segment 1
number of edge segment | x-y coordinates
For edge segment 2
number of edge segment | x-y coordinates
EDGE LIST
For edge 1
edge number | edge segment number
For edge 2
edge number | edge segment number
SURFACE LIST
For surface 1
surface number | edge number
For surface 2
surface number | edge number

Figure A.2. Structure of the lists formed in Step A

points in the node are concatenated with the number of the node and placed on a node list similar to the x-y coordinate list. After the node has been traced, the value of each point of the node is changed from 1 to -1 in the input edge array. The first time an edge segment is encountered, it is numbered and traced out in the input edge array. As the edge segment is traced, the x-y coordinates of the points on the edge segment are concatenated with the number of the edge segment and placed on the x-y coordinate list, and the value of each point on the edge segment is changed from a 1 to a -1 in the input edge array. In the edge-segment library, five locations are set aside for each new edge segment, as shown in Fig. A.2. After the edge segment has been traced, the number of the surface being outlined is placed in the location, "the number of the first surface on the right" (NFSR). The number of the start node, the number of the end node, and the number of points in the edge segment are placed in their respective locations. During the process of following all the boundaries, each edge segment will be encountered exactly twice. The second time a segment is encountered, it will be traced in the opposite direction. The surface to the right of the edge segment is now the other surface that has the edge segment on its boundary. Therefore, the number of the surface being outlined is placed in "the number of the second surface on the right" (NSSR) in the edge segment library under the number of the edge segment. This completes the information about the edge segment in the edge segment library. As each surface is outlined, an ordered list of the edge segments which form the boundary of the surface is created. As each edge segment is encountered, the surface number being outlined is concatenated with the edge segment numbers and placed on the surface list. The detailed steps of Step A follows.

AII.1) SURFACE:=0; EDGE:=0; NODE:=0

AII.2) Start from the upper left-hand most point in the edge array and search for the first point with a 1 value. If no such point is found, go to AIII, (Step AII is complete). If a point is found, mark the point as a node-point.

AII.3) SURFACE:=SURFACE+1.

AII.4) NODE:+NODE+1; trace out the node; place the x-y coordinates concatenated with NODE on the node list; change the value of each node-point in the node to -1 in the edge array; go to the next edge segment on the boundary of SURFACE.

AII.5) If this edge segment has already been traced, find the number of this edge segment; EDGENUM:=number of edge segments; go to AII.9, (i.e.,if the values of the points on the edge segment are -1).

AII.6) EDGE:=EDGE+1; place the number of the start node in "number of start node" of EDGE in the edge segment library; trace the points in the segment; concatenate EDGE with the x-y coordinates of each point and place on the x-y coordinates list; count the number of points traced; change the value of each point to -1 in the edge array; continue tracing until another node is encountered; place the number of points in the edge segment in edge segment library under EDGE; place SURFACE in "the number of the first surface on the right" in the edge segment library; concatenate SURFACE and EDGE and place on the surface list.

AII.7) If the node has already been traced, (i.e.,if the value of the points of the node are -1), get the number of the node by finding any one of the x-y coordinates of the node in the node list and using the concatenated node number; place the node number in the "number of end node" of EDGE in the edge segment library; and go to AII.10.

AII.8) Place NODE+1 in "number of end node" of EDGE; and go to AII.4.

AII.9) Place SURFACE in "number of second surface on the right" of EDGENUM in the edge segment library; concatenate SURFACE and EDGENUM and place on the surface list; jump to the node at the other end of the edge segment (this node will already have been named and points in the node are concatenated with the number of the node and placed on a node list similar to the x-y coordinate list. After the node has been traced, the value of each point of the node is changed from 1 to -1 in the input edge array. The first time an edge segment is encountered, it is numbered and traced out in the input edge array. As the edge segment is traced, the x-y coordinates of the points on the edge segment are concatenated with the number of the edge segment and placed on the x-y coordinate list, and the value of each point on the edge segment is changed from a 1 to a -1 in the input edge array. In the edge-segment library, five locations are set aside for each new edge segment, as shown in Fig. A.2. After the edge segment has been traced, the number of the surface being outlined is placed in the location, "the number of the first surface on the right" (NFSR). The number of the start node, the number of the end node, and the number of points in the edge segment are placed in their respective locations. During the process of following all the boundaries, each edge segment will be encountered exactly twice. The second time a segment is encountered, it will be traced in the opposite direction. The surface to the right of the edge segment is now the other surface that has the edge segment on its boundary. Therefore, the number of the surface being outlined is placed in "the number of the second surface on the right" (NSSR) in the edge segment library under the number of the edge segment. This completes the information about the edge segment in the edge segment library. As each surface is outlined, an ordered list of the edge segments which form the boundary of the surface is created. As each edge segment is encountered, the surface number being outlined is concatenated with the edge segment numbers and placed on the surface list. The detailed steps of Step A follows.

AII.1) SURFACE:=0; EDGE:=0; NODE:=0

AII.2) Start from the upper left-hand most point in the edge array and search for the first point with a 1 value. If no such point is found, go to AIII, (Step AII is complete). If a point is found, mark the point as a node-point.

AII.3) SURFACE:=SURFACE+1.

AII.4) NODE:+NODE+1; trace out the node; place the x-y coordinates concatenated with NODE on the node list; change the value of each node-point in the node to -1 in the edge array; go to the next edge segment on the boundary of SURFACE.

AII.5) If this edge segment has already been traced, find the number of this edge segment; EDGENUM:=number of edge segments; go to AII.9, (i.e.,if the values of the points on the edge segment are -1).

AII.6) EDGE:=EDGE+1; place the number of the start node in "number of start node" of EDGE in the edge segment library; trace the points in the segment; concatenate EDGE with the x-y coordinates of each point and place on the x-y coordinates list; count the number of points traced; change the value of each point to -1 in the edge array; continue tracing until another node is encountered; place the number of points in the edge segment in edge segment library under EDGE; place SURFACE in "the number of the first surface on the right" in the edge segment library; concatenate SURFACE and EDGE and place on the surface list.

AII.7) If the node has already been traced, (i.e.,if the value of the points of the node are -1), get the number of the node by finding any one of the x-y coordinates of the node in the node list and using the concatenated node number; place the node number in the "number of end node" of EDGE in the edge segment library; and go to AII.10.

AII.8) Place NODE+1 in "number of end node" of EDGE; and go to AII.4.

AII.9) Place SURFACE in "number of second surface on the right" of EDGENUM in the edge segment library; concatenate SURFACE and EDGENUM and place on the surface list; jump to the node at the other end of the edge segment (this node will already have been named and traced); go to the next edge segment on the boundary of SURFACE. If this edge segment has not already been traced, go to AII.6; if it has, EDGENUM:=number of the edge segment.

AII.10) If EDGENUM is not the same as the number of first edge segment on the surface list for SURFACE, go to AII.9.

AII.11) If every edge segment has a number for the "number of the second surface on the right", go to AII.2.

AII.12) Find the number of the first edge segment which does not have a number for the "number of the second edge segment on the right"; EDGENUM:=number of edge which does not have a NSSR; proceed in the direction opposite the direction of the original trace on the edge segment; SURFACE:=SURFACE+1, go to AII.9.

AIII) This step marks the edge segments and nodes that are on the perimeter. For each point on the perimeter point list (from Step AI), find the edge segment or node that contains the point. If it belongs to a node, concatenate a 1 onto the number of the node everywhere it occurs in the edge segment library. If the point belongs to a edge segment, set the "number of points in the edge segment" of that edge to zero in the edge segment library.

AIV) This step creates a list of edges; each edge is represented by a list of one or more edge segments. First we remove the edge segments which form the boundaries of the "noise surfaces" from the edge segment library. The total number of points on the boundary of each surface is computed. This procedure is done for each surface by summing the "number of points in the edge segment" of all the edge segments on the surface list of the surface. If the total is less than a preset number (we use 8), then the "number of points in the edge segment" (NPES) is set to zero for each edge segment on the surface list. This procedure removes the small surfaces usually caused by glare or noise in the original scene. The surface list of each surface is examined to try to find edge segments with nonzero NPES in an attempt to join some of edge segments to form longer edges. Two or more edge segments with nonzero NPES form an edge if they have the same NFSR and NSSR and appear sequentially in any surface list. An edge list is constructed in which the number of each edge is concatenated with the number(s) of the edge segment(s) that form the edge. This concept of edges can be illustrated by Fig. A.3. The edge segments are indicated ES-, the nodes by N-, the surfaces by S-, and the edges by E-. The edge segments ES11, ES5, ES6, and ES9 would be placed on the edge list for edge 1 because ES1, ES2, ES3, and ES4 have their NPES's set to zero (on the perimeter). ES7, ES8, and ES12 have their NPES's set to zero because they form the boundary of a surface with less than 8 points, in this case only 5 points. The details of Step AIV follow.

AIV.1) Sum all the values of NPES for all the edge segments on the surface list for the surface. If the sum is less than a preset number (8), place zero in the NPES of each edge segment on the surface list for the surface.

AIV.2) EDGE:=0; SURF:=0.

AIV.3) SURF:=SURF+1; if SURF $>$ SURFACE go to BI, (Step A is complete).

AIV.4) Search the surface list of the SURF for the first edge segment number that does not already appear in the edge list and has NPES greater than zero; BEGIN:= this edge segment number; if none are found, go to AIV.3.

AIV.5) Search the rest of the surface list for the next edge segment with NPES greater than zero, whose NFSR or NSSR is different from the NFSR and NSSR of BEGIN, and which does not already appear on the edge list. If such an edge segment is found, place its number in BEGIN.

AIV.6) EDGE=EDGE+1; Concatenate EDGE with BEGIN and place in edge list; circular search the surface list of SURF for edge segments which have nonzero NPES and have the same NFSR and NSSR as BEGIN. For each edge segment found, concatenate EDGE with its number and place on the edge list. If an edge segment is found that has a nonzero NPES, but does not match NFSR and NSSR of BEGIN, go to AIV.7. If the circular search of the surface list of SURF returns to the edge segment BEGIN, go to AIV.3.

AIV.7) Place the number of the edge segment in BEGIN and go to AIV.6.

APPENDIX B

This appendix describes the data structure and the algorithm of Step B. The input data is the set of four lists from Step A: x-y coordinate list, edge segment library, edge list, and surface list. In this step, two lists are formed --the unknown-edge library and the lines list--and are passed to Step C to be used in either learning or recognizing the object in the scene. The structure of these lists is shown in Fig. B1.

Step B

BI) A list of the x-y coordinates of the points on an edge is created by using the x-y coordinates of the points on the edge segments that form the edge and by filling in any gaps between edge segments. If there is more than one edge segment on the edge list for an edge, the gap between the end of one edge segment and the start of the next edge segment is approximated by a straight line. Points approximating a straight line between the ends of the edge segments are placed in the coordinate list of the edge. If a surface has only one edge for its boundary, (i.e., the surface is isolated like a hole), the x-y coordinates of the edge are copied again onto the end of the edge's coordinate list. The result is an edge which represents two passes around the surface. This step is done to simplify some comparisons in Step C. The following are the detailed steps of BI.

BI.1) EDGE - has the number of edges in the scene from STEP A; EDGENUM:=0; POINTER:=0.

BI.2) EDGENUM:=EDGENUM+1; If EDGENUM >EDGE, go to Step C.

BI.3) For edge EDGENUM, find the number of the first edge segment on edge list and place the number of the edge segment in EDGESEG; COUNT:=0.

BI.4) Transfer the x-y coordinates of the edge segment EDGESEG from the x-y coordinates list to the edge coordinate list of EDGENUM; COUNT:=COUNT + number of points in EDGESEG.

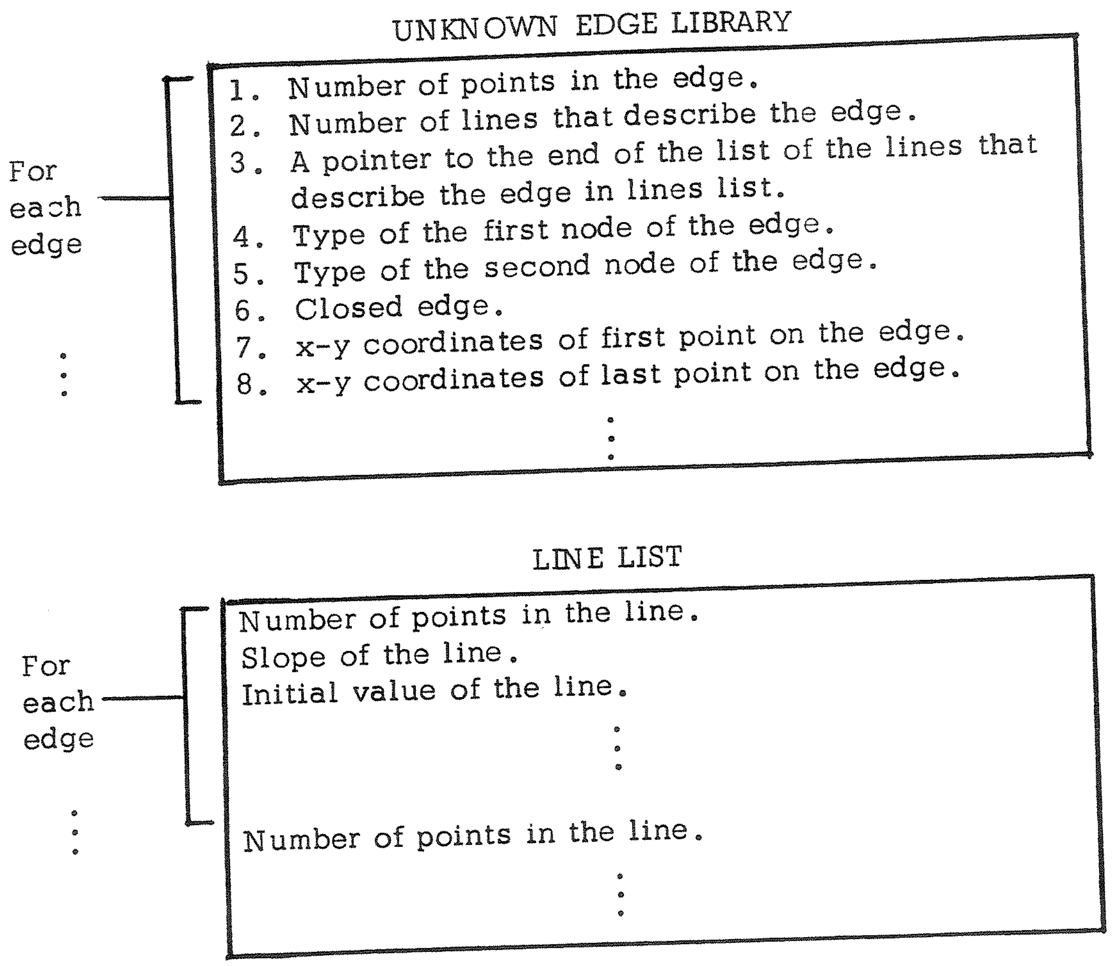

UNKNOWN EDGE LIBRARY
For each edge
1. Number of points in the edge.
2. Number of lines that describe the edge.
3. A pointer to the end of the list of the lines that describe the edge in lines list.
4. Type of the first node of the edge.
5. Type of the second node of the edge.
6. Closed edge.
7. x-y coordinates of first point on the edge.
8. x-y coordinates of last point on the edge.
LINE LIST
For each edge
Number of points in the line.
Slope of the line.
Initial value of the line.
Number of points in the line.

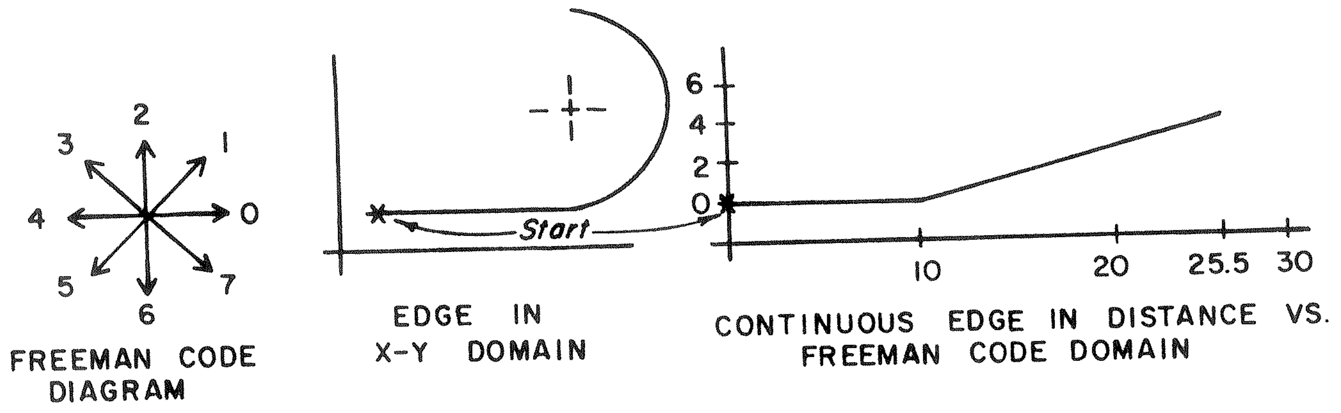
Figure B.1. List Structure for Step B

BI.5) If EDGESEG is the last edge segment on the edge list of EDGENUM, go to B5.7.

BI.6) EDGESEGL:=EDGESEG; place the number of the next edge segment in EDGESEG; form a set of points which approximate a straight line between the last x-y coordinate of EDGESEGL and the first x-y coordinate of EDGESEG, place these points on the edge coordinate list and add the number of points to COUNT; go to B5.4.

BI.7) If for any surface on the surface list EDGENUM is the only edge on the surface list, copy the edge coordinate list onto the end of the edge coordinate list and double COUNT.

BII) In this step the x-y coordinates of an edge are converted into a Freeman code [6]. The Freeman code is transformed into a smoothed-compensated-extended-Freeman code. A Freeman code of eight directions is used. If the Freeman code of an edge is plotted against distance as in Fig. B2.1, straight lines in the edge appear as straight horizontal lines on the graph. Arcs in the edge appear as straight lines in the graph whose slopes are proportional to the curvature of the arcs in the edge. If the edge is quantized as in Fig. B2.2, the discrete points in the graph approximate straight lines. This observation suggests a method of edge description--fit straight lines to the points in the graph of the Freeman code, and this is the method we use. However, the Freeman code has two major problems which must first be corrected. Figure B3.1 illustrates the first problem with the Freeman code. Assume we use the Freeman code shown, then even though the x-y graph represents a continuous edge, the graph of the code has large jumps when the code goes from 7 to 0 or from 0 to 7. These jumps can be removed by forming what we call an extended-Freeman code, EFC. Let the Freeman code of an edge be represented by $\{f_i\}_1^n$ and let $\{e_i\}_1^n$ represent the EFC, then define $e_1=f_1$. If $|e_1-f_2| > 4$, add/subtract multiples of 8 to/from f_2 until $|e_1-f_2| \leq 4$. Then set $e_2=f_2$.



B2.1. Continuous edge in the Freeman Domain

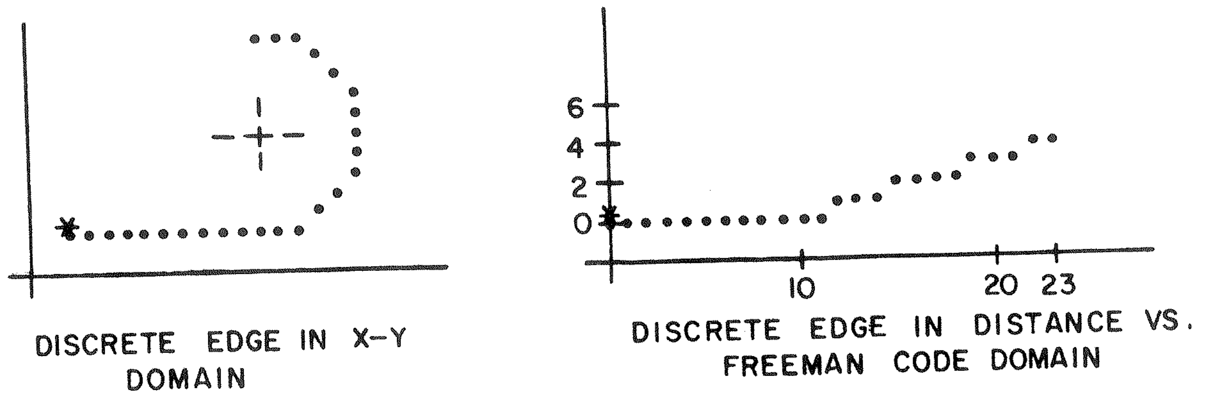
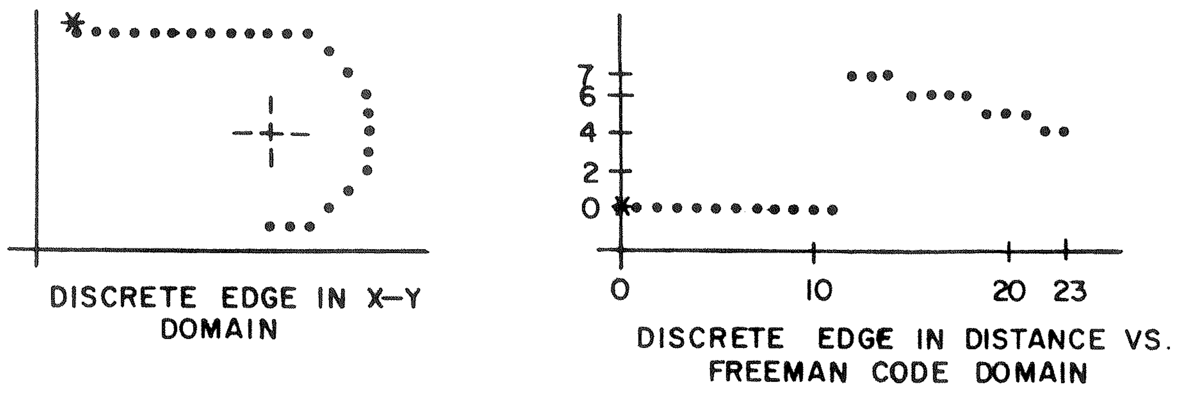

DISCRETE EDGE IN X-Y DOMAIN
6
4
2
0
10
20
23
DISCRETE EDGE IN DISTANCE VS. FREEMAN CODE DOMAIN

Figure B.2. Discrete Edge in the Freeman Domain



B3.1. Freeman code of an edge.

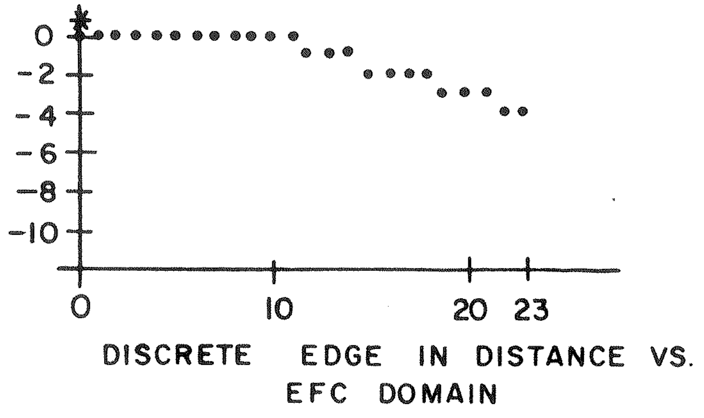
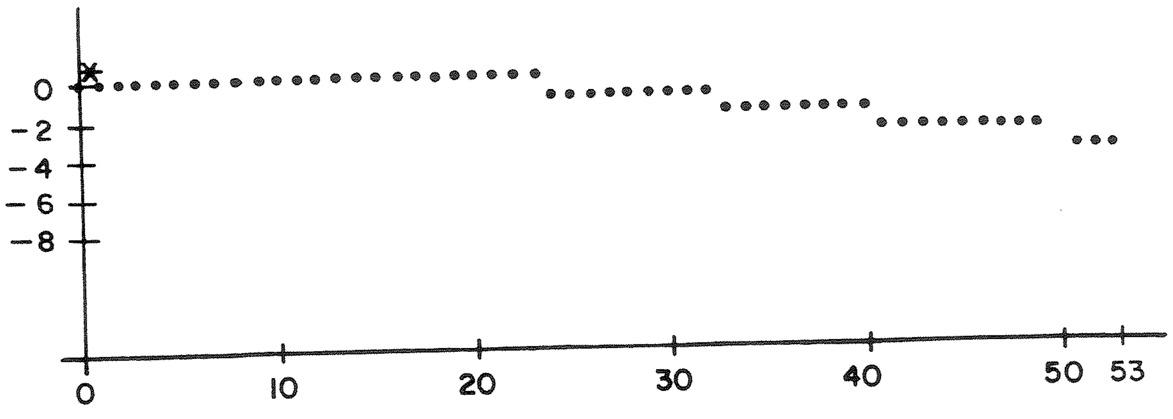


Figure B.3. Extended Freeman code of an edge

Do this for the successive values of e_i. In other words, for $2 \leq i \leq n$, $e_i = f_i + 8*k$ such that $|e_{i-1} - e_i| \leq 4$, where $k=0, \pm 1, \pm 2, \ldots$ The EFC graph of the edge of Fig. B3.1 is plotted in Fig. B3.2. To plot the Freeman code, a distance of one unit between code points was assumed. This assumption results in the second problem with the Freeman code representation: if we let the horizontal or vertical distance between points be one unit in the x-y domain, then the distance in the diagonal direction between points is $\sqrt{2}$ units. One way to get an accurate Freeman code versus distance graph is to use the same relative distances between points as in the x-y domain. However, the computation time of Step BIII can be greatly reduced if we have integer distance coordinates. Although method we use results in an error of approximately 4 percent, we do obtain integer distance coordinates. We assume that the diagonal distance is 1.5 units, and a compensated-extended Freeman code, CEFC is formed from the EFC. If the ECF value is even, it is repeated twice in the CEFC; if the EFC value is odd, it is repeated three times in the CEFC. The distance between points in the CEFC domain is defined to be one unit. For the edge of Fig. B3.1 the CEFC graph is shown in Fig. B4.1. Next the CEFC is smoothed. At present a smoothing window of nine points is used. A smoothed-compensated-extended-Freeman code (SCEFC) is formed from the CEFC. Let s_i be the i^{th} term of the SCEFC and $\{c_i\}_1^m$ be the CEFC, then define $s_i = \sum_{j=1}^{9} c_{i-5+j}$. This results in a graph in which the sharp steps between levels are removed. Figure B4.2 is the SCEFC of Fig. B4.1. Now the detailed step of BII.

BII.1) Let $\{x_i, y_i\}_1^n$ be the sequence of x-y coordinates on the edge coordinate list and n:=COUNT. Let $\{f_i\}$ be the Freeman code of the edge coordinate list. Then define

$\Delta x_i = x_{i+1} - x_i$; $\Delta y_i = y_{i+1} - y_i$; and f_i by



B4.1. CEFC of an edge.

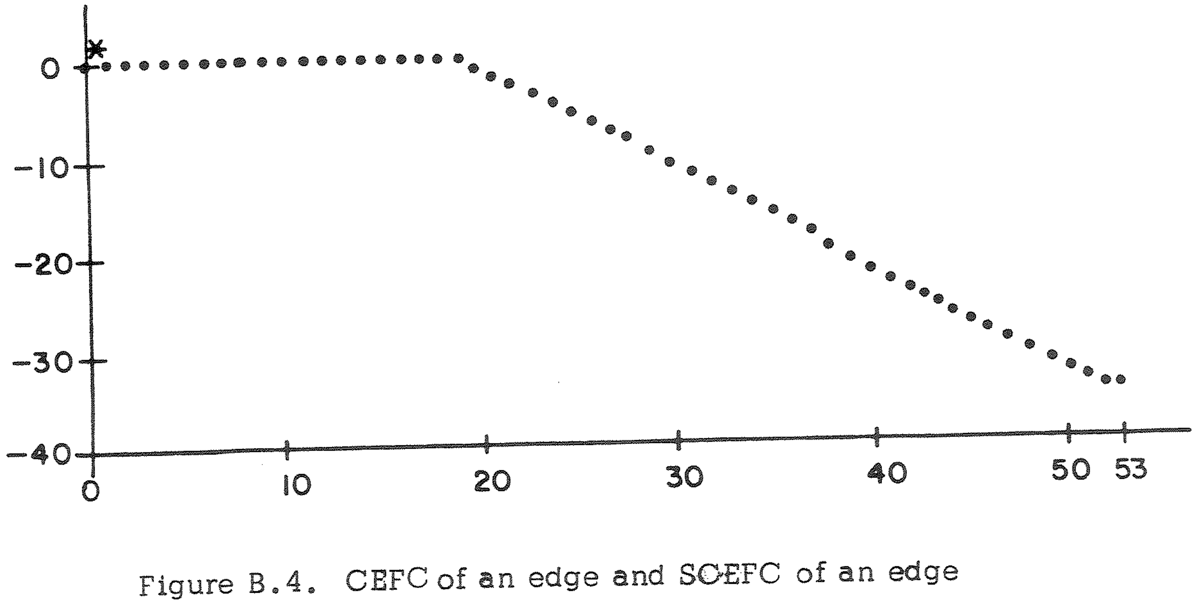


Figure B.4. CEFC of an edge and SCEFC of an edge

$$f_i = \begin{cases} 0 \text{ if } \Delta x_i = 1,\ \Delta y_i = 0 \\ 1 \text{ if } \Delta x_i = 1,\ \Delta y_i = 1 \\ 2 \text{ if } \Delta x_i = 0,\ \Delta y_i = 1 \\ 3 \text{ if } \Delta x_i = -1,\ \Delta y_i = 1 \\ 4 \text{ if } \Delta x_i = -1,\ \Delta y_i = 0 \\ 5 \text{ if } \Delta x_i = -1,\ \Delta y_i = -1 \\ 6 \text{ if } \Delta x_i = 0,\ \Delta y_i = -1 \\ 7 \text{ if } \Delta x_i = 1,\ \Delta y_i = -1 \end{cases}$$

for $1 \leq i \leq n-1$.

BII.2) Let $\{e_i\}$ be the expanded-Freeman code. Then define e_i by

```
        e_1 = f_1;
        For i:=2, Step 1 Until n Do
Again:  If |e_{i-1} - f_i| ≤ 4 Then e_i:=f_i Else Begin
            If e_{i-1} - f_i < 0 Then f_i: = f_i - 8 Else f_i: = f_i + 8;
             Go to Again; End
```

BII.3) Let $\{c_j\}$ be the compensated-expanded-Freeman code. Then define c_j by

```
j:=1;
For  i:=1 Step 1 Until n-1 Do Begin
    c_j: = e_i; j:=j+ 1;
    c_j: = e_i; j:=j+1;
    c_j: = e_i;
    If e_i is odd Then j: = j+1; End
m:=j
```

BII.4) Let $\{s_i\}$ be the smoothed-compensated expanded-Freeman code. Let NFIL be the number of points to be summed for each point in the SCEFC. For $i < \frac{NFIL-1}{2}$ define s_i by

$$s_i = \sum_{j=1}^{NFIL-i} c_j + \sum_{j=2}^{i+1} c_j .$$

For $\frac{NFIL-1}{2} \le i \le m - \frac{NFIL-1}{2}$, define s_i by

$$s_i = \sum_{j=\frac{-NFIL+1}{2}}^{\frac{NFIL-1}{2}} c_{j+i}.$$

For $i > m - NFIL$, define s_i by

$$s_i = \sum_{j=i}^{m} c_j + \sum_{j=2m-NFIL-i+1}^{m-1} c_j.$$

BIII) A series of straight lines is fitted to the points of the SCEFC using a mean-square error method. Each line can be represented by three numbers: number of points on the line, slope of the line, and initial value of the line. The description of the edge in the form of sets of the three number representing the lines is placed on a lines list. A pointer to the location of the description of the edge is placed in the unknown edge library under the number of the edge. In addition, the following information about the line description of the edge is placed in the unknown edge library under the number of the edge: the number of points in the edge, the number of lines that describe the edge, and whether the edge is closed,(i.e., the only edge in the boundary of a surface.) The structures of these two tests are shown in B1.

The sequence $\{s_i\}_1^m$ is passed to Step BIII as the SCEFC for an edge. The mean square error of a straight line fitted to all the data is found. If the MSE is greater than a preset constant, the point in the SCEFC data which has the greatest distance from the line is found. The distance coordinate of this point is placed in END. The SCEFC data is divided into two lists: from the start to END and from END+ 1 to the end of the SCEFC. Straight lines are MSE fitted to the data on these lists. If the MSE of either list is greater than the preset constant, the data for that list is divided as above and the process repeated. This process is repeated until each list of SCEFC data has a MSE less than a preset constant. The following equations are the equations for a MSE straight line fit. Let p be the initial value and q the slope of the straight line. Let the SCEFC be represented by the coordinate pairs (d_i, s_i) where d_i is the distance. By the method by which the SCEFC list was constructed, the distance between consecutive SCEFC data points is one unit, i.e., $d_{i+1} = d_i+1$. Then p and q are found by solving

$$\sum_{i=1}^{N} s_i = p*N + q \sum_{i=1}^{N} d_i$$

and

$$\sum_{i=1}^{N} s_i*d_i = p \sum_{i=1}^{N} d_i + q \sum_{i=1}^{N} d_i^2 \quad ,$$

where N is the number of data points.

The distance between the MSE line and any data point is given by $\ell_i = s_i - p - q\, d_i$. The square error, E is given by

$$E = \sum_{i=1}^{N} \ell_i^2 = \sum_{i=1}^{N} s_i^2 - p \sum_{i=1}^{N} s_i - q \sum_{i=1}^{N} s_i d_i .$$

We now use the fact that the SCEFC data points are one unit apart to simplify some of the computation. If we let $d_i = 1$, then

$$\sum_{i=1}^{N} d_i = \frac{N(N+1)}{2}$$

and

$$\sum_{i=1}^{N} d_i^2 = \frac{N(N+1)(2N+1)}{6} \quad .$$

If the data started at M and ended at N, then

$$\sum_{i=M}^{N} d_i = \frac{N(N+1) - M(M-1)}{2}$$

and

$$\sum_{i=M}^{N} d_i^2 = \frac{N(N+1)(2N+1) - M(M-1)(2M-1)}{6} \quad .$$

Let

$$D1(M,N) = \sum_{i=M}^{N} d_i ; \quad D2(M,N) = \sum_{i=M}^{N} d_i^2 ;$$

$$S1(M,N) = \sum_{i=M}^{N} s_i ; \quad S2(M,N) = \sum_{i=M}^{N} s_i^2 ;$$

$$DS(M,N) = \sum_{i=M}^{N} s_i d_i ;$$

and

$$E(M,N) = \sum_{i=M}^{N} \ell_i^2 .$$

Let $DN(M,N) = (N-M+1) * D2(M,N) - D1(M,N) * D1(M,N)$. For a straight line over the points M to N of the SCEFC data,

$$p = (D2(M,N)*S1(M,N) - D1(M,N)*DS(M,N))/DN(M,N)$$

and

$$q = ((N-M)*DS(M,N) - D1(M,N)*S1(M,N))/DN(M,N).$$

The mean square error would be

$$MSE = E/(N-M+1) = (S2(M,N) - p \cdot S1(M,N) - q \cdot DS(M,N))/(N-M+1).$$

Let us define some variables used in the line fit algorithm:

COUNT is the number of data points in the SCEFC;

SLIST(·) is the SCEFC list of data points;

CONSTANT is a preset constant;

LINES(·) is the lines list; and

UNKNOWN(·,·) is the unknown-edge library.

Let us present the details of the line fitting algorithm.

BIII.1) CONST:=CONSTANT; S1(1):=0; S2(1):=0; DS(1):= 0; COUNT:=COUNT+ 1. Do the following for INDEX=2 through COUNT.

S1(INDEX): = S1(INDEX-1) + SLIST(INDEX-1);

S2(INDEX): = S2(INDEX-1) + $(SLIST(INDEX-1))^2$;

DS(INDEX): = DS(INDEX-1) + SLIST(INDEX-1)*INDEX-1.

Define the functions S1(M,N): = S1(N) - S(M-1); S2(M,N): = S2(N) - S2(M-1); and DS(M,N): = DS(N) - DS(M-1).

BIII.2) INDX:=1; STACK(INDX,1):=1; STACK(INDX,2):=COUNT; CONST:=CONST*1.25; INDX1:=POINTER+1 (POINTER defined in BI.1).

BIII.3) M:=STACK(INDX,1); N:=STACK(INDX,2); INDX:=INDX-1;

q: = ((N-M+1)*DS(M,N)-D1(M,N)*S1(M,N))/DN(M,N);

p: = (D2(M,N)*S1(M,N)-D1(M,N)*DS(M,N))/DN(M,N);

MSE: = (S2(M,N)-p*S1(M,N)-q*DS(M,N))/(N-M+1);

If MSE $\leq$ CONST, go to BIII.8.

BIII.4) If N-M $\leq$ 12, go to BIII.8.

BIII.5) Use $\ell_i = s_i - p - q\, d_i$ to find the point NPT where ℓ_i is maximum on (M+6,N-6).

BIII.6) INDX:=INDX+1; STACK(INDX,1):=NPT; STACK(INDX,2):= N; INDX:=INDX+1; STACK(INDX,1):=M; STACK(INDX,2):=NPT.

If INDX $\geq$ 16, go to BIII.2.

BIII.7) Go to BIII.3.

BIII.8) If INDX1 $\geq$ POINTER+35, go to BIII.2.

LINES(INDX1):=N-M+1;

LINES(INDX+1):=p;

LINES(INDX+2):=q;

INDX1:=INDX1+3.

BIII.9) If INDX > 0, go to BIII.3.

BIII.10) UNKNOWN(EDGENUM,1):=COUNT-1 (number of points in the edge).

UNKNOWN(EDGENUM,2):=(INDX1-INDX)/3 (number of lines that describe the edge);

POINTER:=INDX1+POINTER:

UNKNOWN(EDGENUM,3):=POINTER (points to the end of the list of lines on lines list);

UNKNOWN(EDGENUM,4):= type of first node of edge (1 if it is a perimeter node, 0 if it is a real node);

UNKNOWN(EDGENUM,5): = type of last node of edge;

UNKNOWN(EDGENUM,6): = closed edge (1 if the edge forms the total boundary around a surface, -1 otherwise);

UNKNOWN(EDGENUM, 7): = x-y coordinates of the first point on the edge;

UNKNOWN(EDGENUM,8): = x-y coordinates of last point on the edge.

Go to BI.2.

To give the reader an idea of how well the series of lines describes an edge, Fig. B5 and Fig. B6 were plotted. The jagged lines are the lines connecting the x-y coordinates of the points on the edges. The smooth lines are derived from the line descriptions of the edges.

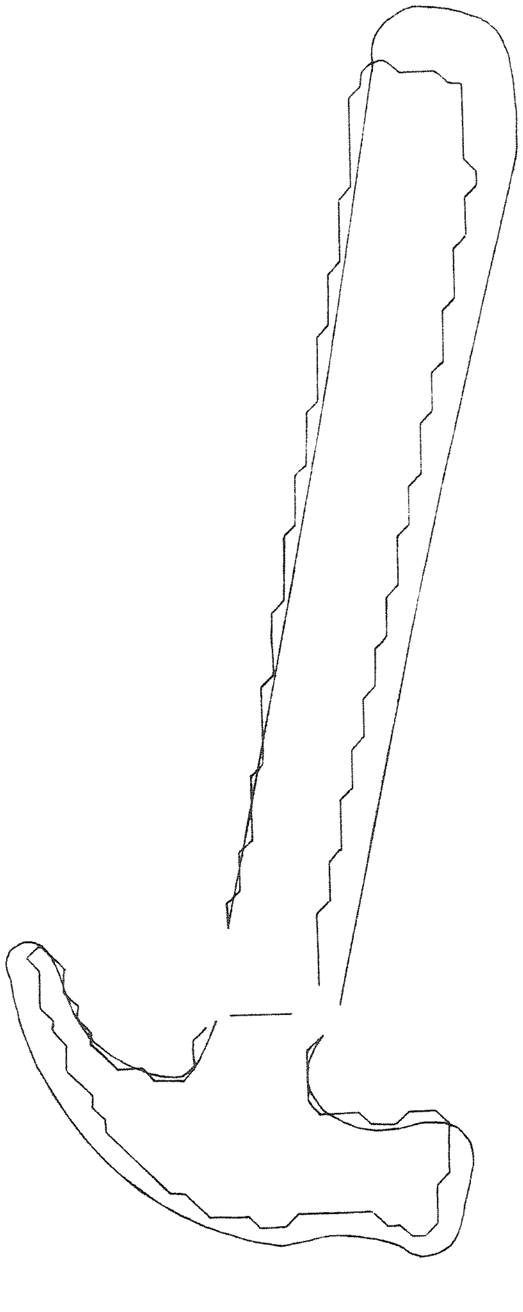


Figure B.5. The SCEFC description of the lines of a hammer plotted in the x-y domain

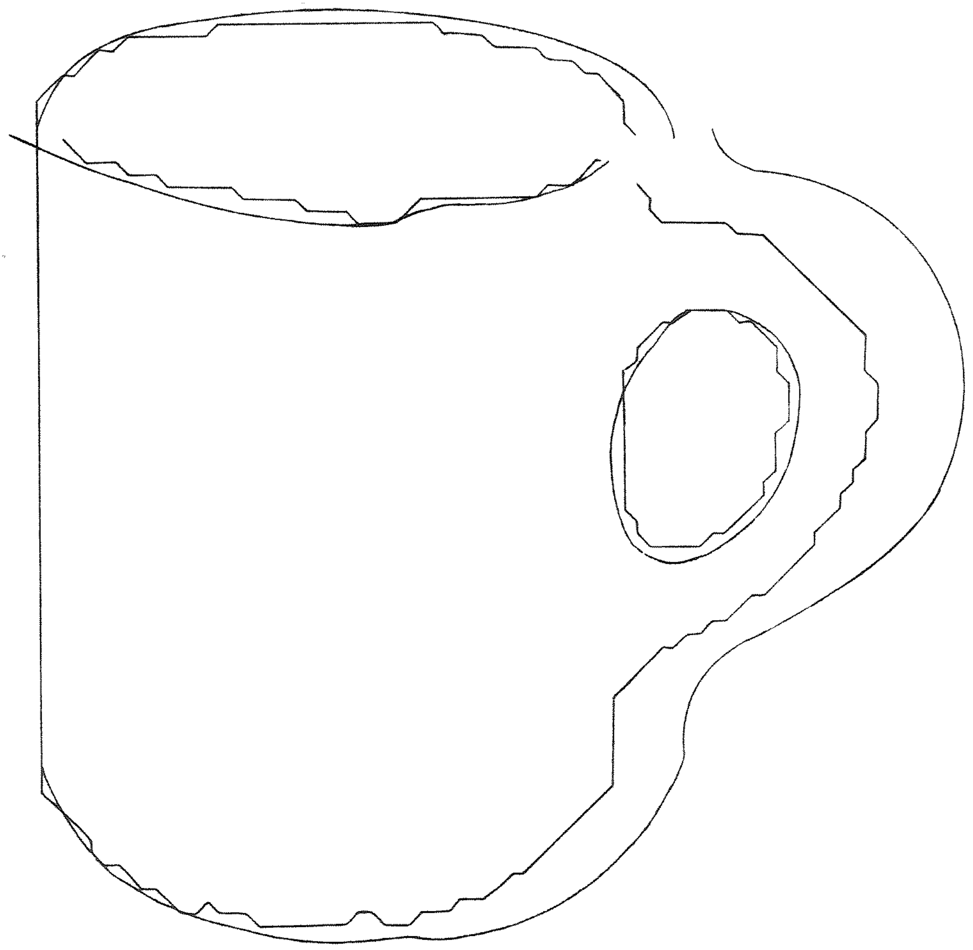


Figure B.6. The SCEFC description of the lines of a cup plotted in the x-y domain

APPENDIX C

This appendix describes the data structure and the algorithm of Step C. Step C consists of two different but similar algorithms: the algorithm for learning a description of an object and the algorithm for recognizing a partial view of an object. Both algorithms use the two lists created in Step B--the unknown-edge library and the lines list. The unknown-edge library contains a description of the edges in the scene. The lines list contains the descriptions of the lines that were fitted to SCEFC data. Both algorithms also use the following four lists: the known-objects list, the object-description list, the known-edge list, and the edge-description list. These four lists are created before any object is learned. They are initialized with no data in them, and they are stored in an external permanent file (magnetic tape). The learning program updates these lists; the recognition program uses the lists but does not change them. For each known object, the known-object list contains the name of the object and a pointer to the description of the object in the object-description list. For each known object, the object-description list has a description of each edge of the object. This description contains the name of the edge, an alternative name for the edge if one exists, and the length of the edge. The known-edge list contains the names of the known edges and their descriptions. These are the names used in the object-description list. For each known edge, the following information is stored in the known-edge list: the length of the edge, the number of straight lines that describe the edge, whether the edge is closed, a pointer to the set of lines that describe the edge in edge-description list, and a pointer to curvature information about the edge in the edge-description list. For each edge in the known-edge list, the edge-description list contains the length, slope, and initial value of each of the lines that describe the edge and measures of the curvature of the edge. The structures of these list are shown in Fig. C1 and Fig. C2. To restate how an edge is described, each edge is approximated in the SCEFC domain by a series of straight lines. The slope of each line is directly proportional to the curvature of the edge at the equivalent position. The slope of the line times the length of the line is proportional to the angle subtended by the equivalent portion of the edge. The total positive angle subtended, TPAS, is the sum of the products of the slope of the line times its length for all lines with positive slopes. The total negative angle subtended, TNAS, is computed similarly. The TPAS and the TNAS are calculated and stored in the edge-description list. These are the measures of the curvature of the edge discussed earlier.

Let us make some general observations about comparing an unknown edge and a known edge. Both edges are represented in the form of a series of lines with lengths, slopes, and initial values. Let us designate the known edge as the reference edge. Picture the reference edge as a piecewise continuous series of straight lines drawn on a graph starting at 0 and ending at, say, 40, as shown in Fig. C3.1. The length, initial value, and slope of each line of the reference edge are found in the known-edge list under the number of the edge. Also picture the unknown edge as a piecewise continuous series of straight lines drawn on a rubber sheet that can be stretched only lengthwise, as shown in Fig. C3.2. The length, initial value, and slope of each line of the unknown edge are found in the lines list under the number of the edge. There are three independent operations that can be performed on the two lines. The rubber sheet can be stretched or contracted (scaled) with respect to the reference edge. The sheet can be shifted either forward or backward with respect to the reference edge and a window can be placed over the reference edge and the unknown edge so only a portion of each series of lines appear. We call

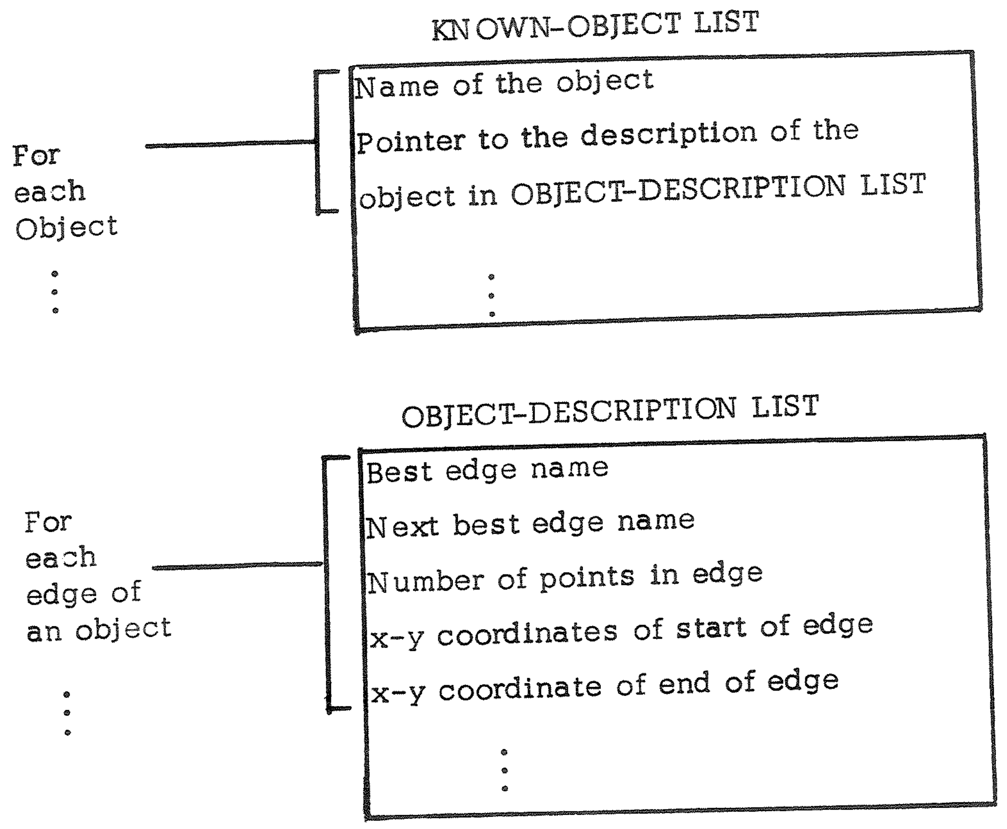

KNOWN-OBJECT LIST
For each Object
Name of the object
Pointer to the description of the object in OBJECT-DESCRIPTION LIST
OBJECT-DESCRIPTION LIST
For each edge of an object
Best edge name
Next best edge name
Number of points in edge
x-y coordinates of start of edge
x-y coordinate of end of edge

Figure C.1. Structure of Known-Object List and Object-Description List

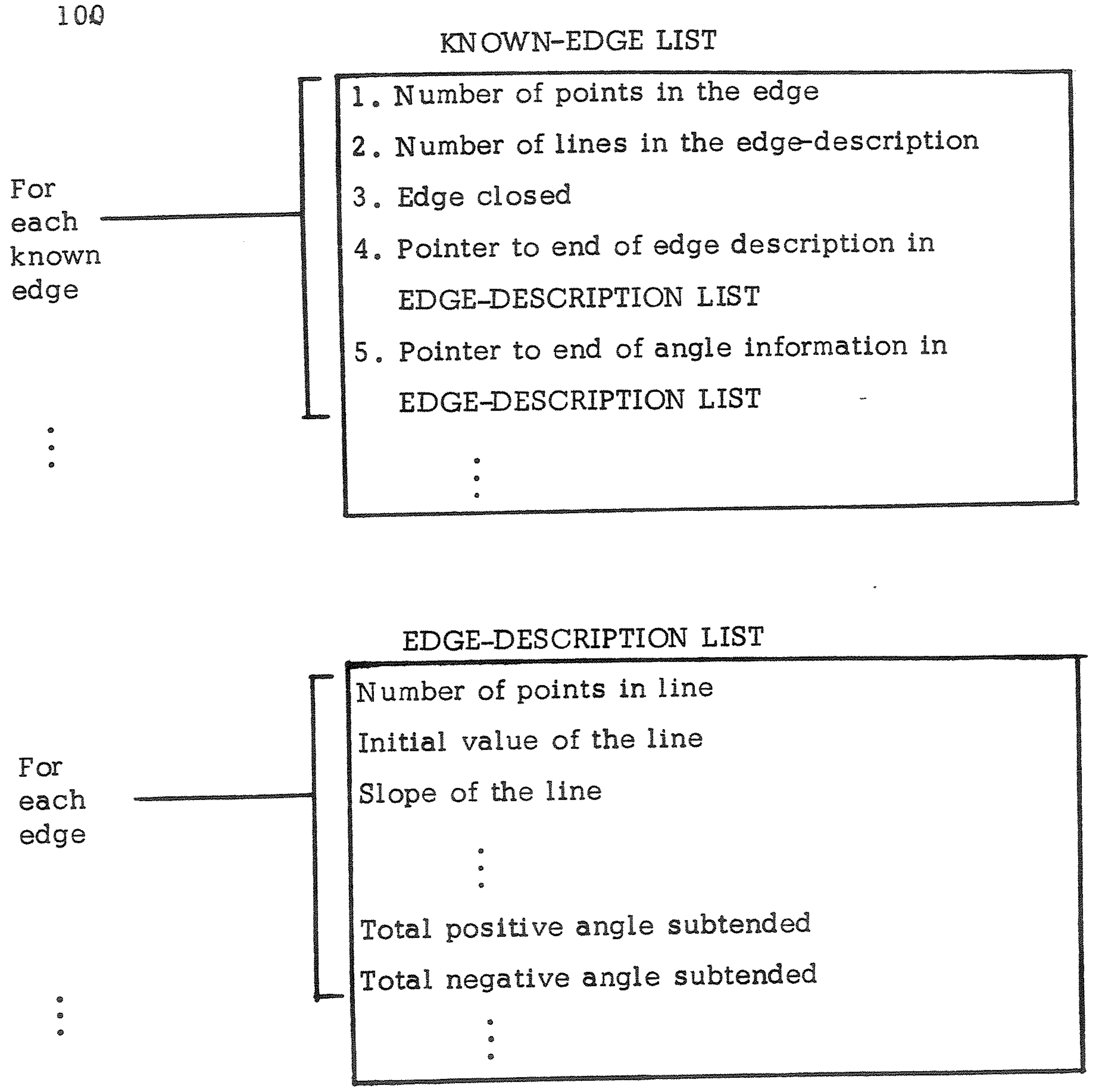
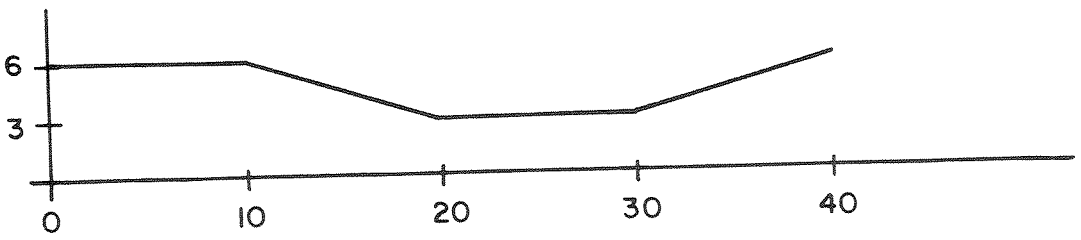
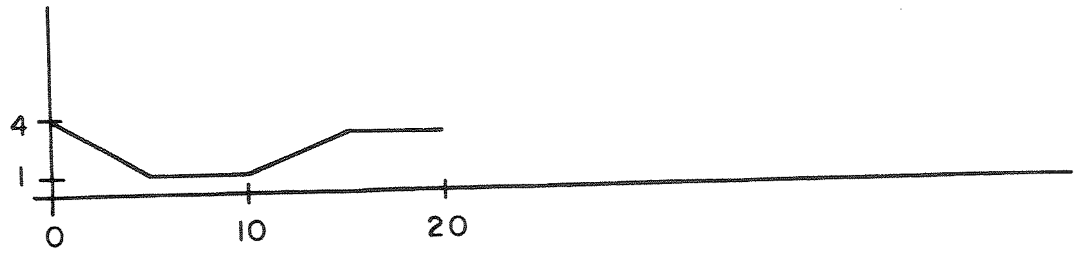

KNOWN-EDGE LIST
For each known edge
1. Number of points in the edge
2. Number of lines in the edge-description
3. Edge closed
4. Pointer to end of edge description in EDGE-DESCRIPTION LIST
5. Pointer to end of angle information in EDGE-DESCRIPTION LIST
EDGE-DESCRIPTION LIST
For each edge
Number of points in line
Initial value of the line
Slope of the line
Total positive angle subtended
Total negative angle subtended

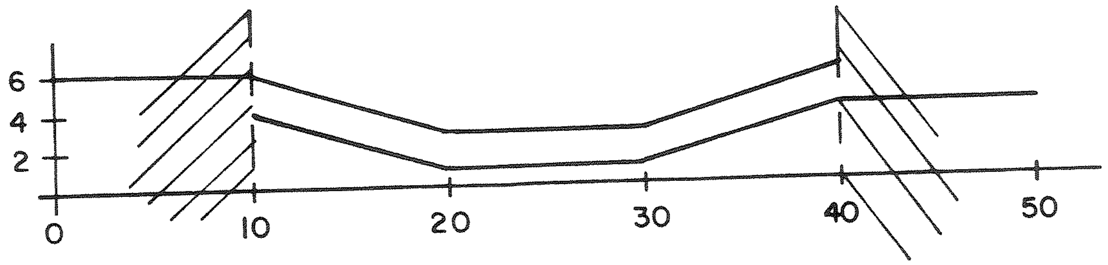
Figure C.2. Structure of Known-Edge List and Edge-Description List



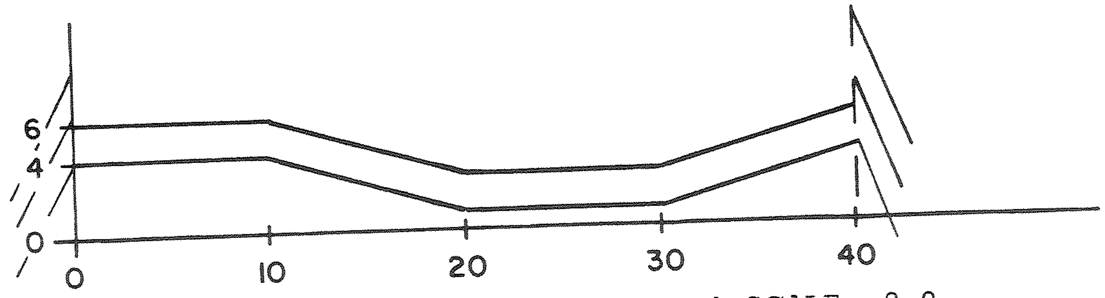
C3.1. Known edge in SCEFC domain.



C3.2. Unknown edge in SCEFC domain.



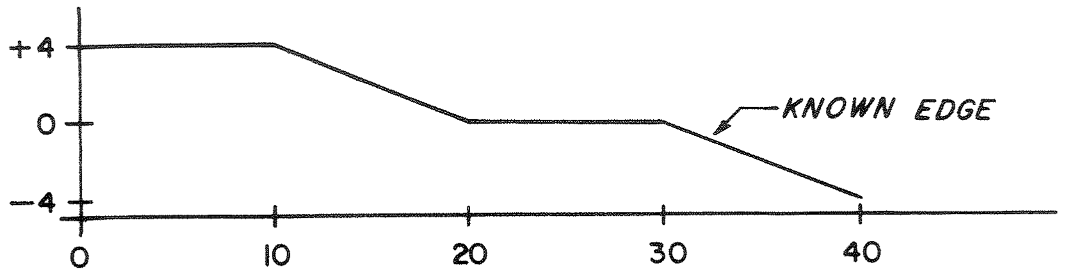
C3.3. Unknown edge and known edge with SCALE = 2.0, SHIFT = 0.25, TRUNK1 = 0.25, and TRUNK2 = 1.0.

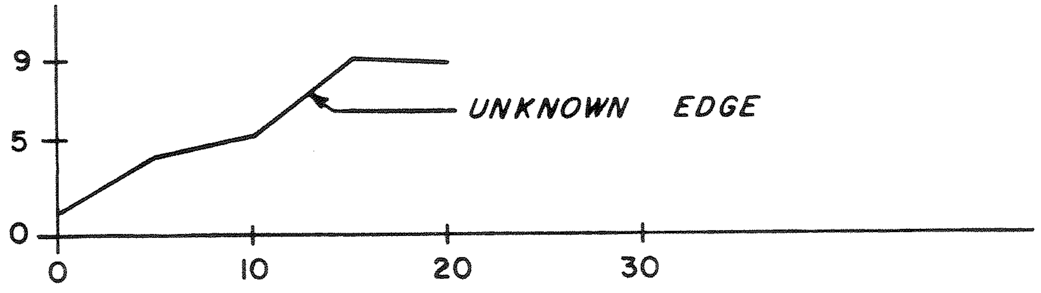


C3.4. Unknown edge and known edge with SCALE = 2.0, SHIFT = 0.0, TRUNK1 = 0.0, TRUNK2 = 1.0, and with the known edge forward and the unknown edge reversed.

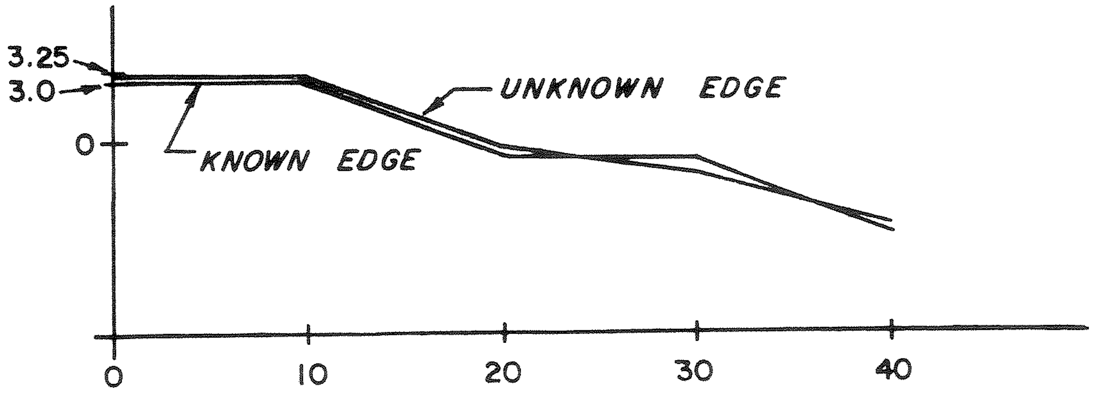
Figure C.3. Comparison between a known edge and un unknown edge.

this truncating the edges. In this algorithm the truncating, scaling, and shifting are performed on the unknown edge referenced to the known edge. First let us discuss scaling. From its respective list, we know the length of each edge. The number SCALE is defined as the length of the reference divided by the length of the unknown edge. To make the unknown edge equal in length to the reference edge, multiply the length of each line of the unknown edge by SCALE and divide the slope of each line by SCALE. For instance, if we wanted the unknown edge to be half the length of the reference edge, use SCALE/2 in place of SCALE. Note: In a line compari-ison, the scaled length of the unknown edge will never be greater than the length of the reference edge because the reference edge is a complete edge whereas the unknown edge may be a partial edge. For the edge of Figs. 2.1 and 2.2, SCALE = 40/20 = 2. Shifting refers to the position of the start of the unknown edge in relation to the start of the reference edge. SHIFT is defined as the position of the start of the unknown edge minus the position of the start of the reference edge normalized by the length of the reference edge. For example, in Fig. B3.3 with SCALE = 2.0, the unknown edge is shifted right one-quarter of the length of the known edge and SHIFT becomes 0.25. For SHIFT = -0.25, the unknown edge would have been shifted to the left of the reference edge by the same amount. Truncation is defined with respect to the reference edge. Define TRUNK1 to be the normalized front portion of the reference edge which is cut off, and TRUNK2 to be the nor-malized portion of the reference edge past which everything is cut off. Or $0 \leq$ TRUNK1 $\leq$ TRUNK2 ≤ 1, and only examine the relative portion of the reference edge between TRUNK1 and TRUNK2. In Fig. C3.3, the slashed area represents the parts to be cut off; in this case, TRUNK1 = 0.25 and TRUNK2 = 1.0. An edge has two directions--forward and reverse. When the lines that describe a forward edge are fetched, they are fetched starting with the line at the top of the list. When the lines are fetched for a reverse edge, the description of the edge is reversed. This procedure is done by starting the fetch with the last line of the description. The negative of the slope is used and a new initial value is calculated for each line fetched. The new initial value is the final value of the line, representing the sum of the old initial value plus the old slope times the length of the line. Let the direction of the unknown edge be defined as forward in Figs. C3.2 and C3.3. Then in Fig. C3.4 the direction of the unknown edge is changed to reverse, and we use SCALE = 2.0, SHIFT = 0.0, TRUNK1 = 0.0 and TRUNK2 = 1.0. An obvious method of comparing the description of two edges becomes evident from Fig. C3.3 and C3.4. Use the area between the graphs of the two edges as the metric. There is only one problem: how to compensate for the same edge being at different orientations in the x-y domain. The two edges of Fig. C3.4 have the same shape in the x-y domain; they are just rotated versions of each other in the x-y domain. The solution to this problem is simple: find the average value of the known edge and subtract it from the initial value of each line of the known edge's description. Do the same for the unknown edge. This procedure removes the effect of rotation on the edge. Thus, given two edges described by a series of straight lines, we can scale, shift, and truncate one line with respect to the other. We can also reverse the direction of a line and remove the effect of rotation between lines. Let us illustrate these concepts with an example. The known edge and the unknown edge are shown in Fig. C4.1. In Fig. C4.2, the average value of the known edge has been subtracted from the known edge. Also, the average value of the unknown edge has been subtracted from the unknown edge. The unknown edge has been reversed and scaled with SCALE = 2.0. The measure of the difference between the two edges would be the normalized sum of the areas between the different





C4.1. Known edge and unknown edge.



C4.2. Known edge minus its average and unknown edge minus its average, reversed with SCALE = 2.0, SHIFT = 0.0, TRUNK1 = 0.0, and TRUNK2 = 1.0.

Figure C.4. Comparison between a known edge and an unknown edge that is independent of direction.

lines. With this in mind, let us discuss the learning and recognition algorithms.

Step CL

CLI) The learning step consists of two main parts: learning the names of the edges of the object and constructing a description of the object from the edge information. The learning step receives the two lists--unknown-edge library and the lines list--from Step B and reads in the current version of the known-object list, object-description list, known-edge list, and edge-description list. It also reads in the name of the object it is learning, placing it in the known-object list. Each edge in the unknown-edge library is named by a method described later. The name or names, the length of the edge, and other data about the edge are placed on the object-description list under the name of the object. After all the edges have been named and their descriptions placed on the object-description list, a pointer to this information is placed with the name of the object in the known-object list.

CLI.1) From Step B we get EDGE, the number of edges on the unknown-edge library. Read in the known-object list, object-description list, known-edge list, and edge-description list. Also, read in the pointers KOLP, ODLP, KELP, EDLP which point to the end of each list, respectively.

CLI.2) Read in the name of object being learned, place the name on the end of the known-object list, and move KOLP to end of list. EDGENUM: = 0.

CLI.3) EDGENUM: = EDGENUM + 1, if EDGENUM > EDGE, go to CLI.6.

CLI.4) Call the subroutine Compare-learn (CLII).

CLII.5) On the object-description list, place the following: best edge name, next best edge name if there is one, number of points in the edge EDGENUM, x-y coordinate of the start of the edge, x-y coordinate of the end of the edge. Move ODLP to end of object-description list, place the value of ODLP at the end of known-object list, and move KOLP to the end of the list. Go to CLI.3.

CLI.6) Write out the new lists and pointers to a permanent file; end of the program.

CLII) Let us discuss the method used to find the names of an edge. The description of an unknown edge is compared to the descriptions of all the edges in the known-edge list. This method of comparison will be discussed later. If the unknown edge is not "similar" to any edge on the known-edge list, it is given a name, and the following data about the edge is transferred from the unknown-edge library to the known-edge list: number of points in the edge, number of lines that describe the edge, and whether the edge is closed. The set of lines that describe the edge is transferred from the lines list to the edge-description list. The total positive angle subtended, TPAS, and the total negative angle subtended, TNAS of the edge are computed and stored in the edge-description list. Pointers to the set of lines and the angle numbers are placed with the other data about the edge in the known-edge list. The name of this new known edge is returned as the name of the unknown edge. If the unknown edge is "similar" to one edge in the known-edge list, the name of that edge is returned as the name of the unknown edge. If the unknown edge is "similar" to two or more known edges, the name of the two "most similar" edges are returned. Let us discuss the method of comparing an unknown edge to the edges on known-edge list. The TPAS and TNAS are calculated for the unknown edge. Starting with the first edge on the known-edge list, the following procedure is performed for each known edge. If the unknown edge is closed and the reference edge is not closed, or vice versa, the reference edge is rejected, and the process starts over with the next edge. Three angle comparisons are made: known TPAS to unknown TPAS, known TNAS to unknown TNAS, and known TPAS plus known TNAS to unknown TPAS plus unknown TNAS. The known edge is rejected if any of the three comparisons are not within a preset tolerance. If an edge is rejected, the direction of the known edge is reversed, the angle numbers are reversed, and the angle tests are tried again. If it fails again, the process starts over with the next edge on the known-edge list. If the known edge passes the angle tests, then a normalized-sum-mean-difference, NSMD between the two edges is computed by a method described in the introduction to Step C. If the NSMD for the two edges is less than a preset number, the name of the known edge is placed on a list ordered by the NSMD values. The process goes to the next edge on the known-edge list. After all the edges have been processed, the list of names is examined. If it is empty, the unknown edge is given a name and learned, and this new name is returned. Otherwise, the name of the edge with the lowest NSMD and the name of the edge with the second lowest NSMD, if it exists, are returned as the name of the edge.

CLII.1) Subroutine compare-learn.

CLII.2) Calculate the total positive angle subtended and the total negative angle subtended by the following formulas:

UPOSA: = Σ(slope of the line)x(length of the line for all lines with positive slope.)

UNEGA: = Σ(slope of the line)x(length of the line for all lines with negative slope.)

ANGLE: = (UPOSA - UNEGA)/4 (note: UNEGA is negative, so

ANGLE is 1/4 the total angle subtended by the edge). If ANGLE $\leq$ NFIL, ANGLE: = NFIL (NFIL is the order of the smoothing filter, for NFIL = 9, ANGLE = 9 is equivalent to 45°). REFERENCE:=0.

CLII.3) REFERENCE:=REFERENCE+1; if REFERENCE $>$ total number of known edges go to CLII.15. SCALE: = number of points in the edge REFERENCE/number of points in the unknown edge (EDGENUM). RPOSA:= total positive angle subtended (from edge-description list for REFERENCE); RNEGA: = total negative angle subtended.

CLII.4) If one edge is closed and the other edge is not closed, go to CII.3.

CLII.5) TRUNK1:=0.0; TRUNK2:=1.0; SHT:=0.0; SC:=1.0; LOOP1:=1; FORWRDU:=1; FORWRDR:=1 (FORWRDU and FORWRDR indicate the direction of the unknown edge and the known edge, respectively, with 1 being the forward direction and -1 being the reverse direction).

CLII.6) If the unknown edge is not closed, go to CLII.7; TRUNK1:= .25; TRUNK2:= .75; SHT:= -.25; LOOP1:= 10; (this step is skipped if the edge is not closed; if it is closed, TRUNK1, TRUNK2, and SHT will cause the middle half of the known edge to be compared with half of the unknown edge since they are scaled by SCALE to be the same length. As will be seen, with LOOP1 = 10 ten shifts and comparisons of the two edges will be tried. It should be noticed that the closed edge description represents two passes around the edge, and, therefore, one half of the description represents a complete pass around the edge.)

CLII.7) LP2:=1.

CLII.8) If $|$UPOSA-RPOSA-UNEGA+RNEGA$|$ $>$ ANGLE or if $|$UPOSA-RPOSA$|$ $>$ ANGLE or if $|$UNEGA-RNEGA$|$ $>$ ANGLE, go to CLII.13.

CLII.9) LP:=1, SHIFT:=SHT.

CLII.10) Call subroutine normalize-sum-mean-difference (CLIII). The difference in areas is returned in NSMD.

CLII.11) If NSMD $\leq$ TRES1, go to CLII.14. (THRES1 is a preset constant.)

CLII.12) SHIFT:=SHIFT+.05; LP1:=LP1+1; if LP1 $\leq$ LOOP1, go to CLII.10.

CLII.13) TEMP:=RPOSA; RPOSA:=-RNEGA; RNEGA:=-TEMP; FORWRDR:=-1; LP2:=LP2+1; if LP2 $\leq$ 2, go to CLII.8.

CLII.14) If NSMD $>$ THRES1, go to CLII.3; otherwise place (REFERENCE, NSMD) on a list BESTEDGE in ascending order according to the value of NSMD; go to CLII.8.

CLII.15) If BESTEDGE is empty, go to CLII.16; otherwise return to the calling program (CLI) the number with the smallest NSMD value as the name of the edge. If there is more than one entry in BESTEDGE, return the number with second smallest NSMD value as an alternate name of the edge. Return to the calling program.

CLII.16) (If the algorithm comes to this step, it has failed to find a name for the unknown edge. The following procedure learns the edge.) Transfer the following from the unknown-edge library to the known-edge list: the number of points in the edge, the number of lines in the edge description, whether the edge is closed. Transfer the values for the description of the lines from the lines list to the edge-description list. Move the EDLP to the end of the list and place the value of EDLP at the end of the known-edge list. Place UPOSA and UNEGA at the end of the edge description list. Move the EDLP to the end of the test and place its value at the end of the known-edge list. Move the KELP to the end of the known-edge list. Increment the number of known edge and return this number as the name of the unknown edge. Return to the calling program.

CLIII.1) Subroutine normalized-sum-mean-difference. (The values for TRUNK1, TRUNK2, SHIFT, FORWRDU, FORWRDR, and SCALE are set in the calling program.)

CLIII.2) UEND:=SCALE* length of unknown+ SHIFT* length of reference edge; START:=max (TRUNK1, SHIFT)* length of reference edge; END:=min (TRUNK2* length of reference edge, UEND); NSMD:=2* TRES1; LENGTH:=END-START, if LENGTH $\leq$ 0, return to calling program.

CLIII.3) For the unknown edge, find the average value, UAV of the series of lines in the FORWRDU direction over the interval START to END. Start the unknown edge at SHIFT* length of reference edge.

CLIII.4) For the reference edge, find the average value, RAV of the series of lines in the FORWRDR direction over the interval START to END. Start the edge at 0.

CLIII.5) Find the sum of the differences of the means between the unknown edge shifted by -UAV and the reference edge shifted by -RAV over the interval START to END.

CLIII.6) NSMD:= sum of the differences of the mean/LENGTH; return to calling program.

Step CR

CRI) The recognition portion of the algorithm consists of two main steps: naming the edges of the unknown object and finding the named edges in the description of known objects. Each edge in the unknown-edge library has an edge type associated with it. An edge is type 1 if the node at each end of the edge is a perimeter node. An edge is type 2 if one of the nodes is a perimeter node and the other is a real node. An edge is type 3 if both nodes are real nodes. The weight of an edge is the length of the edge times the type. The weight of an edge is stored under the name of the unknown edge. The edges in the unknown-edge library are examined in order of their type, starting with type 3. A list of possible names of the edge is created in Step CRII. For each object in the known-object list, a comparison is made between the names of the edges in the known object's description and the names on the list of possible names. If a name of an edge in the object's description is matched to one of the possible names for an edge of the partial view, the status of the edge in the description of the object is updated. When the object-description list is read in, the status of each edge is zero. If an edge is matched to a type 3 edge from the partial view, the status of the edge is changed to 1. From this point on, whenever this edge is examined and its status checked, it will be skipped and no attempt will be made to match it to another edge from the partial view. In other words, once an edge in an object description has been matched to an edge in the partial view that has two real nodes, the edge in the object description is no longer available to be matched to other edges of the partial view. If an edge is matched to a type 2 edge, 2 is added to status of the edge, if an edge is matched to a type 1 edge, 3 is added to the status of the edge. When the status of an edge exceeds 5, no further attempt is made to match it to any other edges of the partial view. This procedure allows the following matches of an edge in an object's description to edges from the partial view: 3 type 2 edge matches; 2 type 2 edge matches and 1 type 3 edge match; or 2 type 1 edge match. Multiple matches of type 2 and type 3 edges from the partial view to an edge in an object's description are allowed for this reason: in a partial view, an edge of an object may be broken into a number of edges by the perimeter. After the list of possible names for an edge of the partial view has been created, the names in the list are compared to the names in the description of each object. If a name in the list is the same as a name of an edge in an object's description and the status of the edge is compatible to the edge type, the status of the edge is updated and the name and length of the edge are placed in a two-dimensional array that is referenced by the combination of the unknown-edge name and the object name. For each edge on the list of possible names, Step CRII also returns the length of the edge. If the edge has two real nodes, the length of the edge is just the length of the unknown edge. If the edge extends out of the scene, Step CRII estimates the total length of the edge including the portion not seen and returns this as the length of the edge. At the completion of the above portion of the algorithm, we have a vector in which the weight of each unknown edge is stored and can be referenced by the name of the unknown edge. We also have a two-dimensional array that is referenced by the ordered pair (unknown-edge name, object name). Initially all the entries in the two-dimensional array are zero. If the name of an edge on the list of possible edge names for an unknown edge was in description of an object and if the unknown edge satisfied the status condition in the description of the object, the entry in the two-dimensional array, for the ordered pair (unknown-edge name, object name), is the name of the edge and its length.

For each object, an average scale factor is computed. Initially SCALE and WEIGHT are zero. For each unknown edge, the following is performed. The entry under the ordered pair (unknown-edge name, object name) is checked; if it is zero, processing goes to the next unknown edge. If it is not zero, the name of the edge is extracted. This name is compared to the names of all the other edges in the array under the object name. If it is found to match another name, processing goes to the next unknown edge. This is done because if the edge matched the name of another edge in the object's description, there are at least two edges with the same name in both the partial view and the object's description.

Therefore, we would not know which edge in the partial view to match to which edge in the object description. If the name is unique then a weighted scale factor is computed. The weighted scale factor is the product of the weight of the unknown edge times the length of the edge in the array divided by the length of the edge in the description of the object in the known-object list. This number is added to SCALE to form the new SCALE. The weight of the edge is added to WEIGHT. Then the next unknown edge is processed. After all the edges are processed the average scale factor is SCALE divided by WEIGHT.

For each object a weight is computed. Initially the weight is zero. For each unknown edge the following is performed. The entry under the ordered pair (unknown-edge name, object name) is checked, if it is zero,processing goes to the next unknown edge. If it is not zero, the name of the edge is extracted. This name is compared to each name in the object's description. If it matches a name in the object's description, the length of the edge in the array is divided by the length of the edge in the object's description. This gives a edge scale factor. If this edge scale factor is within a fixed percentage of the average scale factor for the object, the weight of the edge is added to the weight of the object. Processing then goes to the next unknown edge. If it is not within the fixed percentage, the search continues until all the edges in the object's description have been examined, then processing goes to the next unknown edge. This results in a weight for the object. A weight for each object is computed by this method. The list of weights of the objects is searched. The names and normalized weights of the three objects having the largest three weights are printed out. The object's weight is normalized by the sum of the weights of all the edges. And the program ends.

CRI.1) Read in the known-object list, object-description list, known-edge list, edge-description list and the pointer KOLP, ODLP, KELP, EDLP, which point to the end of each list, respectively. OBJWEIGHT(.):=0; SUMWEIGHT:=0; TP:=4.

CRI.2) EDGENUM:=0; if TP $\leq$ 0, go to CRI.12.

CRI.3) EDGENUM:=EDGENUM+1; if EDGENUM $\geq$ EDGE, go to CRI.2. (EDGE is the number of edge in the unknown-edge library.)

CRI.4) TYPE:=3 - type of start node of the edge EDGENUM - type of end node of EDGENUM (a perimeter node is denoted by a 1, a real node by a 0. The type of node is found in the unknown-edge library under the edge's number.) WEIGHT:=TYPE*length of EDGENUM: SUMWEIGHT:=SUMWEIGHT+ WEIGHT; EDGEWEIGHT (EDGENUM):= WEIGHT.

CRI.5) Call compare-recognize (CRII). The program returns with the best names in BEST(·) and the corresponding length in LENGTH(·). These may be anywhere from 0 to 6 names in BEST(·).

CRI.6) OBJ:=0.

CRI.7) REFEDGE:=0; OBJ:=OBJ+ 1; if OBJ > the number of known objects, go to CRI.2.

CRI.8) REFEDGE:=REFEDGE+ 1; for the known object OBJ, if REFEDGE > number of edges in the object's description, go to CRI.7.

CRI.9) If the status of REFEDGE is 1 or greater than 6, go to CRI.8.

CRI.10) If either name of REFEDGE of OBJ is on BEST, go to CRI.11, otherwise go to CRI.8.

CRI.11) Add 1 to the status of REFEDGE if TYPE=3, 2 if TYPE=2, and 3 if TYPE=1. ARRAY(OBJ, EDGENUM, EDGE):=REFEDGE=BEST(I); ARRAY(OBJ, EDGENUM, LENGTH):=LENGTH(I); go to CRI.8.

CRI.12) OBJ:=0, SCALEFT:=0; WEIGHT:=0.

CRI.13) EDGENUMB:=0; OBJ:=OBJ+ 1; if OBJ > the number of known objects, go to CRI.27.

CRI.14) EDGENUMB:=EDGENUMB+ 1; if EDGENUMB > EDGE, go to CRI.18.

CRI.15) If ARRAY(OBJ, EDGENUM, EDGE):=0, go to CRI.14.

CRI.16) If ARRAY(OBJ, EDGENUM, EDGE) is the same as any other edge in OBJ's description, go to CRI.14.

CRI.17) SCALEFT:=EDGEWEIGHT(EDGENUM)*ARRAY(OBJ, EDGE-NUM, LENGTH); length of ARRAY(OBJ, EDGENUM, EDGE) in OBJ's description; WEIGHT:=EDGEWEIGHT(EDGENUM); go to CRI.14.

CRI.18) SCALEFT:=SCALEFT/WEIGHT; EDGENUM:=0; OBJ-WEIGHT(OBJ):=0.

CRI.19) EDGENUM:=EDGENUM+ 1, if EDGENUM > EDGE, go to CRI.12.

CRI.20) If ARRAY(OBJ, EDGENUM, EDGE):=0, go to CRI.19.

CRI.21) REFEDGE:=0.

CRI.22) REFEDGE:=REFEDGE+ 1, if REFEDGE > number of edges in OBJ's description, go to CRI.19.

CRI.23) If ARRAY(OBJ, EDGENUM, EDGE) is not equal to the edge corresponding to REFEDGE in OBJ's description, go to CRI.22.

CRI.24) EDGESCALE:=ARRAY(OBJ, EDGENUM, LENGTH)/length of the edge corresponding to REFEDGE in OBJ's description.

CRI.25) If |EDGESCALE - SCALEFT| < SCALEFT/4, OBJWEIGHT (OBJ)= OBJWEIGHT(OBJ)+EDGEWEIGHT(EDGENUM) and go to CRI.19.

CRI.26) Go to CRI.22.

CRI.27) LOOP1:=3;

CRI.28) LOOP1:=LOOP1-1, if LOOP1 ≤ 0 end of the program.

CRI.29) Find the value of OBJ such that OBJWEIGHT ≥ OBJWEIGHT(·).

Find the name that corresponds to OBJ. Print out " 'name of OBJ' WITH WEIGHT'1000*OBJWEIGHT/SUMWEIGHT'." Print out any other desired information like the edge array of the partial view with the edges labeled. OBJWEIGHT(OBJ):=0; go to CRI.28.

CRII) This step forms a list of possible names for an edge of the partial view. In a partial view, a type 3 edge occurs when the whole edge is in the scene. In this case, when comparing the edge to a known edge, the overall length of the unknown edge and the length of the known edge are made equal by the appropriate value of SCALE. A type 2 edge occurs when one end of the edge is in the scene and the edge extends out of the scene. Therefore, we do not know what the true length of the edge should be. In this case, we start the known edge and the unknown edge with the same length and do the edge comparison. If it fails, the length of the unknown edge is decreased in reference to the known edge. The end of the unknown edge with the real node is the fixed end of the edge, causing the end of the unknown edge with the perimeter node to lie somewhere along the known edge. Scaling and comparing are done a number of times. If it fails every time, the direction of the known edge is reversed and the process starts over. If the edge is type 1, then both ends of the edge extend out of the scene, and we have only a middle portion of an edge. In this case, both the scale and the position of the unknown edge must be varied. The comparisons are performed for all the combinations; if it fails, the direction of the known edge is reversed; and the process starts over. Whenever a known edge compares within a preset number to the unknown edge, the name of the known edge is placed on a list ordered by the value of the NSMD, and the length of the edge is placed in a corresponding position in an ordered list, LIST(•).

The estimate of the length of the unknown edge is the quotient of the number of point in the reference edge divided by the value of

SCALE which gave the best match between the unknown edge and the reference edge. If the unknown edge is type 3, then the estimate of the length is the length of the unknown edge. If the unknown edge is type 2, or type 1, this procedure estimates the total length of the unknown edge, including the portion of the edge which extends out of the partial view. After all the edges in the known library have been examined, the best six names and their corresponding lengths starting with the smallest NSMD value are returned to the calling program.

CRII.1) Subroutine Compare-recognize (this subroutine is similar to compare-learn).

CRII.2) Calculate the total positive angle subtended and the total positive angle subtended and the total negative angle subtended for the unknown edge by the following formulas:

UPOSA:= Σ(slope of the line)*(length of the line for all lines with positive slope;)

UNEGA:+ Σ(slope of the line)*(length of the line for all lines with negative slope;)

ANGLE:=(UPOSA-UNEGA)/4; if ANGLE $\leq$ NFIL, ANGLE:=NFIL (see CLII.2); REFEDGE:=0; FORWRDU:=1; EDGETYPE:=1 + the type of start node + 2 * the type of end node (if a node is a perimeter node, then type = 1, otherwise type = 0).

CRII.3) If EDGETYPE $\neq$ 3, go to CRII.4, otherwise TEMP:=UNEGA; UNEGA:=-UPOSA; UPOSA:=-TEMP; FORWRDU:=-1; (If EDGETYPE=3, then the start node is the perimeter node and the end node is a real node. This step reverses the direction of the unknown edge so the effective start node is a real node.)

CRII.4) REFEDGE:=REFEDGE+1; if REFEDGE $>$ total number of known edges return to calling program. SCALE: = number of points in the known edge REFEDGE/number of points in the unknown edge

EDGENUM; TRUNK1:=0.0, TRUNK2:=0; SHT:=0.0; SCL:=1.0; SCLE:=0.045; SHFT:=0.05; NSMD1:=NSMD:=2*THRES1. If EDGETYPE=2 or 3, go to CRII.6. If EDGETYPE=4, go to CRII.7.

CRII.5) LOOP1:=1; LOOP2:=1; If one of the two edges is closed and the other open, go to CLII.4. If the known edge is open, go to CRII.8. Otherwise TRUNK1:=0.25; TRUNK2:=0.75; SHT:=-0.25; LOOP1:=10; and go to CRII.8. (If the last portion of this step is executed, then both edges are closed. The last portion of the step causes the middle half of the known edge to be compared to half of the unknown edge. If the comparison fails, the unknown edge is shifted by .05 of its length and compared again, up to 10 times.)

CRII.6) LOOP1:=1; LOOP2:=0; if the edge REFERENCE is closed, go to CRII.4. If UPOSA-UNEGA < NFIL, return to calling program. (This checks whether the partial edge curves by at least 45°, i.e., NFIL is equivalent to 45°. If it does not curve by at least 45°, the edge is too straight and will match to many reference edges; and, therefore, the match will be useless.) Otherwise go to CRII.8. (If the reference edge is closed, then it cannot be the unknown edge which has a real node and, therefore, open.)

CRII.7) LOOP1:=18; LOOP2:= 20. If the edge REFERENCE is open, go to CRII.8. If UPOSA-UNEGA < 2*NFIL, return to calling program. (This checks whether the partial edge curves by at least 90°. If it does not curve by at least 90°, the edge usually cannot be successfully matched to a reference edge and give a reasonable estimate of the true length of the edge.) Otherwise, SCL:=.5; SCLE:=.0225; SHT:= 0.025. (If the last part of this step is executed, then the reference edge is closed. But the unknown edge has two perimeter nodes and therefore is open. The last part of the step scales the reference edge properly so that it can be compared to a portion of an open edge.)

CRII.8) RPOSA:= total positive angle subtended by the edge REFERENCE;RNEGA: = total negative angle subtended (from the edge description list); FORWRDR:=1; LOOP3:= 2.

CRII.9) LP1:=1; SHIFT:=SHT; LOOP3:=LOOP3-1; if LOOP3 ≤ 0, go to CRII.19.

CRII.10) LP2:=1; SC:=SCL.

CRII.11) SCALE:=SC*number of points in reference edge/number of points in unknown edge.

CRII.12) If SHIFT+ SC > 1.05 and REFERENCE is open, go to CRII.16. (SHIFT+ SC > 1.05 would mean that after the unknown edge was shifted and scaled the end of the unknown edge would extend past the end of the reference edge. Therefore, this position is rejected.) If REFERENCE is closed and EDGETYPE:=1, go to CRII.14. (The next step calculates RPOSA and RNEGA for the partial edge. If the reference edge is closed and the unknown edge is type 1, RPOSA and RNEGA are already calculated.)

CRII.13) START:=number of points in REFERENCE*(SHIFT+TRUNK1); if START < 0, START:=0. END:=START+ number of points in REFERENCE* SC*(TRUNK2-TRUNK1). With the reference edge starting at zero, calculate the total positive angle subtended, RPOSA and the total negative angle subtended, RNEGA over the interval START to END.

CRII.14) If |UPOSA-RPOSA-UNEGA+ RNEGA| > ANGLE or if |UPOSA-RPOSA| > ANGLE or if |UNEGA-RNEGA| > ANGLE, go to CRII.16.

CRII.15) Call normalized-sum-mean-difference (CLIII). The difference is returned in NSMD. If NSMD < NSMD1, then SCLE:=SCALE; NSMD1:NSMD.

CRII.16) SC:=SC-SCLE; LP2:=LP2+ 1; if LP2 $\leq$ LOOP2, go to CRII.11.

CRII.17) SHIFT:=SHIFT+ SHFT; LP1:=LP+ 1; if LP1 $\leq$ LOOP1, go to CRII.10.

CRII.18) TEMP=RPOSA; RPOSA:=-RNEGA; RNEGA:=-TEMP; FORWRDR:=-1, go to CRII.9.

CRII.19) If NSMD1 $\geq$ THRES1, go to CRII.4. (THRES1 is a preset constant.)

CRII.20) Place the ordered pair (REFERENCE, NSMD1) on a list BEST(·) ordered by the value of NSMD1. The ordered pair with the smallest value for NSMD1 is first on the list. EDGELENTH:=number of points in the reference edge/SCLE. Place the value of EDGELENTH in the corresponding position on the ordered list LENGTH(·) by the value of NSMD1. Go to CRII.4.

REFERENCES

1. L. G. Roberts, "Machine Perception of Three-Dimensional Solids," Optical and Electro-Optical Information Processing, Vol. 197, J. T. Tippell et al., editions, 1965, pp. 159-197.

2. S. A. Underwood, Visual Learning and Recognition by Computer, Ph.D. Dissertation, The University of Texas at Austin, May 1972.

3. C. R. Brice and C. L. Fennema, "Scene Analysis Using Regions," Artificial Intelligence, 1970, pp. 205-226.

4. H. G. Barrow and R. J. Popplestone, "Relational Descriptions in Picture Processing," Machine Intelligence, 1971, pp. 377-396.

5. J. W. McKee and J. K. Aggarwal, "Finding the Edges of the Surfaces of Three Dimensional Curved Objects by Computer," Pattern Recognition, Vol. 6, No. 3 or 4.

6. H. Freeman, "Computer Processing of Line-Drawing Images," Computing Surveys,Vol. 6, No. 1, March 1974.