CONCEPTS FOR A HIGH LEVEL PROGRAMMING LANGUAGE
FOR
REGIONAL COMPUTER GRAPHICS

by

Bobby Hin Wah Liu
B.Eng., McMaster University, 1977

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in
THE FACULTY OF GRADUATE STUDIES

Department
of
Electrical Engineering

We accept this thesis as conforming to the
required standard

THE UNIVERSITY OF BRITISH COLUMBIA
July, 1979
© Bobby Hin Wah Liu, 1979
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date July 4, 1979.
ABSTRACT

Researchers developing high-level graphics languages have, according to the literature, focused their attention towards line-drawing graphics. Little effort has been devoted to systematic investigations of languages that handle graphical data representing line-drawings as well as solid areas. The latter type of graphical data is usually represented by outlines, and solid area properties are visualized by hatching or filling techniques. A study of the literature on programming languages leads to the conclusion that graphical data should be treated as a data type.

Based on the concept of treating graphical data as a data type in its own right; the mathematical and conceptual aspects of this type of data are investigated and established. Much scattered information, such as the representation of regions, has been unified using formal descriptions. Hatching, one of the many ways of achieving external representation of regions, is also investigated. A hatching algorithm is proposed and implemented that envelops the good features of others, and establishes a framework for hatching algorithms. Its implementation achieves the expected tasks. A proposal for a graphics language demonstrates the usefulness and feasibility of this type of graphics. Some of its features have been implemented. Finally, a fairly complete bibliography serves as a gateway for further research in this area.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLE</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td>x</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. EXTERNAL REPRESENTATION MODELS FOR GRAPHICAL DATA</td>
<td>3</td>
</tr>
<tr>
<td>2.1 General Concepts</td>
<td>3</td>
</tr>
<tr>
<td>2.2 External Representation Models</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Virtues and Limitations of the Lineal and Regional External Representation Models</td>
<td>5</td>
</tr>
<tr>
<td>3. INTERNAL AND EXTERNAL REPRESENTATIONS OF REGIONS</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Background</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Internal Representations of Regions</td>
<td>11</td>
</tr>
<tr>
<td>3.2.1 The 'encoding of the boundaries' Scheme</td>
<td>11</td>
</tr>
<tr>
<td>3.2.2 The 'parallel coding' Scheme</td>
<td>13</td>
</tr>
<tr>
<td>3.2.3 The 'skeleton' Scheme</td>
<td>15</td>
</tr>
<tr>
<td>3.3 A Formal Representation of a Region</td>
<td>18</td>
</tr>
<tr>
<td>4. THE CONCEPTS AND TECHNIQUES OF HATCHING</td>
<td>21</td>
</tr>
<tr>
<td>4.1 Definitions</td>
<td>21</td>
</tr>
<tr>
<td>4.2 A Short Review of Hatching Algorithms</td>
<td>22</td>
</tr>
<tr>
<td>4.3 General Concepts of Hatching</td>
<td>25</td>
</tr>
<tr>
<td>4.4 The Conceptual and Mathematical Elements for Hatching Algorithms</td>
<td>26</td>
</tr>
<tr>
<td>4.4.1 Transformations</td>
<td>27</td>
</tr>
<tr>
<td>4.4.2 Intersection between two lines</td>
<td>28</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Topological Properties.</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Definitions for Singular points</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Generation of a Hatching Line</td>
</tr>
<tr>
<td>4.5</td>
<td>The Developed Hatching Algorithm</td>
</tr>
<tr>
<td>4.5.1</td>
<td>A Formal Representation of a Hatching Pattern in the Developed Hatching Algorithm</td>
</tr>
<tr>
<td>4.5.2</td>
<td>The Algorithm HATCH</td>
</tr>
<tr>
<td>4.5.3</td>
<td>A Description of the Developed Hatching Algorithm</td>
</tr>
<tr>
<td>4.5.3.1</td>
<td>A Short Note on the Manipulation of a Realization Region</td>
</tr>
<tr>
<td>4.5.3.2</td>
<td>Description of the Algorithm HATCH</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Evaluation of the Developed Hatching Algorithm</td>
</tr>
<tr>
<td>4.5.5</td>
<td>Future work</td>
</tr>
<tr>
<td>5.</td>
<td>A SHORT REVIEW OF TWO GRAPHICS LANGUAGES</td>
</tr>
<tr>
<td>5.1</td>
<td>Review of Newman and Sproull's Work</td>
</tr>
<tr>
<td>5.2</td>
<td>Comments on Newman and Sproull's Work</td>
</tr>
<tr>
<td>5.3</td>
<td>Review of Bracchi and Ferrari's Work</td>
</tr>
<tr>
<td>5.4</td>
<td>Comments on Bracchi and Ferrari's Work</td>
</tr>
<tr>
<td>6.</td>
<td>A PROPOSED REGIONAL GRAPHICS LANGUAGE</td>
</tr>
<tr>
<td>6.1</td>
<td>A Formal Description of a Graphical Object in the Proposed Regional Graphics Language, using the Regional External Representation Model</td>
</tr>
<tr>
<td>6.2</td>
<td>The Regional Graphics Language LIGE</td>
</tr>
<tr>
<td>6.2.1</td>
<td>A Graphical Declaration Statement</td>
</tr>
<tr>
<td>6.2.2</td>
<td>A Proposed Set of Graphical Primitives for LIGE</td>
</tr>
<tr>
<td>6.2.3</td>
<td>A Proposed Set of Graphical Operators for LIGE</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Attribute Extraction Statements for LIGE</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.2.5 I/O Statements for LIGE</td>
<td>71</td>
</tr>
<tr>
<td>6.2.6 A Graphical Function for LIGE</td>
<td>73</td>
</tr>
<tr>
<td>7. CONCLUSIONS</td>
<td>75</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>77</td>
</tr>
<tr>
<td>APPENDIX A AREA OF A SIMPLY CLOSED POLYGON</td>
<td>85</td>
</tr>
<tr>
<td>APPENDIX B LIGE HOMOGENEOUS TRANSFORMATION MATRICES</td>
<td>86</td>
</tr>
<tr>
<td>APPENDIX C THE CALCULATION OF THE AREA OF A GRAPHICAL OBJECT.</td>
<td>87</td>
</tr>
<tr>
<td>APPENDIX D THE CONSTRUCTION OF THE LIGE PREPROCESSOR</td>
<td>92</td>
</tr>
<tr>
<td>APPENDIX E AUTHORS SORTED BY SUBJECTS</td>
<td>94</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>C.1</td>
<td>Graphical primitives and its associated k-values</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The lineal external representation model.</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>The regional external representation model.</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Examples of graphical objects that require extra care in defining the starting and end points with a lineal external representation model.</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>Digits and characters described on squares.</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Examples of different types of regions.</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Examples of different types of curves.</td>
<td>10</td>
</tr>
<tr>
<td>3.3</td>
<td>An example of a boundary section enclosed within a boundary section rectangle.</td>
<td>11</td>
</tr>
<tr>
<td>3.4</td>
<td>Freeman's encoding scheme.</td>
<td>12</td>
</tr>
<tr>
<td>3.5</td>
<td>Merrill's scheme.</td>
<td>14</td>
</tr>
<tr>
<td>3.6</td>
<td>Pfaltz and Rosenfeld's scheme.</td>
<td>16</td>
</tr>
<tr>
<td>3.7</td>
<td>A hatched map obtained through the use of Pfaltz and Rosenfeld's scheme.</td>
<td>17</td>
</tr>
<tr>
<td>3.8</td>
<td>Direction convention.</td>
<td>19</td>
</tr>
<tr>
<td>3.9</td>
<td>A convention used for defining a region.</td>
<td>19</td>
</tr>
<tr>
<td>3.10</td>
<td>A simply closed polygon.</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>An example of a hatching pattern.</td>
<td>21</td>
</tr>
<tr>
<td>4.2</td>
<td>An example of a region that has been filled.</td>
<td>22</td>
</tr>
<tr>
<td>4.3</td>
<td>An example of a shaded object.</td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>Shading of compound polygon.</td>
<td>23</td>
</tr>
<tr>
<td>4.5</td>
<td>Example - Plot output</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>Hatching patterns obtained via the use of a windowing technique.</td>
<td>25</td>
</tr>
<tr>
<td>4.7</td>
<td>Conversion of a multiply connected region to a simply connected region.</td>
<td>26</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>Generation of hatching lines, 27</td>
<td></td>
</tr>
<tr>
<td>4.9</td>
<td>Two examples satisfying Theorem 4.2 29</td>
<td></td>
</tr>
<tr>
<td>4.10</td>
<td>Two examples satisfying Theorem 4.3 30</td>
<td></td>
</tr>
<tr>
<td>4.11</td>
<td>An example of a critical point. 30</td>
<td></td>
</tr>
<tr>
<td>4.12</td>
<td>An example of a point of inflection 31</td>
<td></td>
</tr>
<tr>
<td>4.13</td>
<td>Examples of points which are not critical nor points of inflection 31</td>
<td></td>
</tr>
<tr>
<td>4.14</td>
<td>A critical point, ((x_{k+1}, y_{k+1})) 32</td>
<td></td>
</tr>
<tr>
<td>4.15</td>
<td>A point of inflection, ((x_{k+1}, y_{k+1})) 32</td>
<td></td>
</tr>
<tr>
<td>4.16</td>
<td>A point of inflection, ((x_{k+2}, y_{k+2})) 33</td>
<td></td>
</tr>
<tr>
<td>4.17</td>
<td>An example to identify critical points, point of inflection, min-max points. 33</td>
<td></td>
</tr>
<tr>
<td>4.18</td>
<td>An encasing rectangle associated with a hatching pattern. 34</td>
<td></td>
</tr>
<tr>
<td>4.19</td>
<td>An example for sets (Y_k, Y'_k, Y''_k) 35</td>
<td></td>
</tr>
<tr>
<td>4.20</td>
<td>An example with each hatching line defined by one or more segments 37</td>
<td></td>
</tr>
<tr>
<td>4.21</td>
<td>An example with an incorrect HL 38</td>
<td></td>
</tr>
<tr>
<td>4.22</td>
<td>A hatching pattern composed of 3 realization regions, (r_1, r_2, r_3) 40</td>
<td></td>
</tr>
<tr>
<td>4.23</td>
<td>Manipulation of a realization region. 41</td>
<td></td>
</tr>
<tr>
<td>4.24</td>
<td>A realization region composed of a multiply connected region. 47</td>
<td></td>
</tr>
<tr>
<td>4.25</td>
<td>Complement of a realization region. 48</td>
<td></td>
</tr>
<tr>
<td>4.26</td>
<td>Hatching patterns with associated sets of hatching lines at different inclinations, spacings and thickness 50</td>
<td></td>
</tr>
<tr>
<td>4.27</td>
<td>Manipulation of a realization region. 51</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.1</td>
<td>A filled polygon</td>
<td>53</td>
</tr>
<tr>
<td>5.2</td>
<td>Geometric primitives in CADEP.</td>
<td>55</td>
</tr>
<tr>
<td>6.1</td>
<td>A graphical object composed of opaque and line-drawn graphical primitives</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>Graphical primitives of LIG.</td>
<td>62</td>
</tr>
<tr>
<td>6.3</td>
<td>A graphical object</td>
<td>62</td>
</tr>
<tr>
<td>6.4</td>
<td>Graphical primitives</td>
<td>64</td>
</tr>
<tr>
<td>6.5</td>
<td>Graphical primitives</td>
<td>64</td>
</tr>
<tr>
<td>6.6</td>
<td>Graphical primitives defined in LIGE but not in LIG.</td>
<td>65</td>
</tr>
<tr>
<td>6.7</td>
<td>The effects of the complement operator on graphical primitives</td>
<td>68</td>
</tr>
<tr>
<td>6.8</td>
<td>A menu</td>
<td>70</td>
</tr>
<tr>
<td>6.9</td>
<td>An example of the effect of executing the BRUSH statement.</td>
<td>72</td>
</tr>
<tr>
<td>A.1</td>
<td>A simply closed polygon</td>
<td>85</td>
</tr>
<tr>
<td>C.1</td>
<td>A triangle and its instance</td>
<td>87</td>
</tr>
<tr>
<td>C.2</td>
<td>A triangulated polygon and its instance</td>
<td>89</td>
</tr>
<tr>
<td>D.1</td>
<td>The construction of the LIGE preprocessor</td>
<td>93</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENT

I am grateful to my parents and members of my family for their constant encouragement.

I would like to express my deepest gratitude to Dr. G.F. Schrack, the supervisor of this project, for his continued interest, encouragement and guidance during this research work and writing of this thesis. Also, my thanks to Dr. M. Davies, my coreader, for his valuable comments on the first draft of my thesis.
1. INTRODUCTION

Computer graphics deals with the generation, representation and manipulation of graphical data using a computer system. In particular, the system handles units of graphical data which are referred to as graphical objects.

Progress can be seen in all areas of computer graphics, especially in the field of raster graphics. One reason for the increasing popularity of raster-scan display devices is their ability to display two types of data, solid tone areas as well as lines and text. Hereafter, graphics that can handle both kinds of data is referred to as regional graphics. In order to be precise, a definition is necessary: regional graphics refers to the theory and techniques of the internal and external representation of graphical objects which are composed of opaque or solid areas as well as conventional line-drawings in a two-dimensional space.

Not only is an opaque graphical object more appealing than a line-drawn object in many situations, but many applications that require realistic description of an object may be found. Examples of computer-aided applications are: technical or non-technical document illustration, block diagram representation, architectural design, urban planning and layout problems such as integrated circuit mask design.

Although researchers have recognized the importance and potential of regional graphics, most work is still confined to the use of existing line-drawing graphics packages to achieve the visualization of regions by filling and hatching techniques. Little effort is devoted to the investigation of the theory (such as the use of data type) and techniques of regional graphics.

Accepting the usefulness, potential, and demand for regional
graphics, research to refine and propose a better theory, to improve techniques, and to develop a language for regional graphics is necessary. Such research should establish a base and initiate further work in this area. This is the purpose of this thesis.
2. EXTERNAL REPRESENTATION MODELS FOR GRAPHICAL DATA

At this point, the explanations of several terms will help the subsequent descriptions and discussions.

A graphical object is any graphical datum which is treated as an indivisible single unit.

A graphical primitive is a graphical datum whose value is a graphical object that cannot or need not be decomposed any further.

Internal representations of graphical data are the storage of data in a digital computer. In particular, an internal representation of graphical data is related to a scheme for describing the data and an associated data structure.

External representations of graphical data are the ways data are depicted on a two-dimensional display surface.

2.1 General Concepts

Much effort has been focused on the internal representation of graphical data, i.e. the study of the data bases and data structures that are required for the implementation of a graphics language. However, little research has been done on the investigation of an external representation model for graphical data. When dealing with graphical data, the model chosen for the external representation of a graphical object is important. The model employed affects the type and number of graphical operators required to handle graphical objects. It also affects the language constructs and the data structures of a graphics language.

2.2 External Representation Models

Two types of external representation models for graphical objects can be identified and are referred to as the lineal and regional
external representation models.

In the lineal external representation model (Figure 2.1), a graphical object is considered to have a starting and an end point and is composed of graphical objects and/or graphical primitives. A graphical primitive is a line segment or a vector. For convenience, a line segment can be considered to have a head and a tail. The basic operation is concatenation. Concatenation is the joining of the head or the tail of a line segment to the head or the tail of another line segment based on a set of rules. Graphics languages which employ the lineal external representation model are GPL [Sm71], AIDS [St71], EULER-G [Ne71], TUNA [Be73], PDL [Ri75,Sh68,Sh69].

![a graphical primitive](image)

Figure 2.1 The lineal external representation model

In the regional external representation model (Figure 2.2), each graphical object is defined on a finite, plane area. For convenience, the plane area is defined to be a unit square, the definition frame, on which a reference point is defined. The reference point is a point about which all graphical transformations and operations are performed. Any point within the definition frame belongs to the associated graphical object and thus the area within the frame is the identification area of the graphical object. A graphical primitive is an element of a convenient set of predefined graphical objects. Superposition is the basic operation involved. Examples of graphics languages which use the regional external representation model are GROAN [Wy74], L [Na71], and LIG [Sc76a,Sc76b, Sc78].
2.3 Virtues and Limitations of the Lineal and Regional External Representation Models

Some basic notions of the two external representation models have been discussed. At this point, a summary of the virtues and limitations of each model is given. The purpose is to help consolidate the understanding of the two models and to justify the choice used in the work reported here.

Lineal external representation model

Virtues:

+ This model lends itself to situations in which connectivity and direction between graphical objects or signs is important.

+ Existing theory and experience with this model facilitates further research in the area.

Limitations:

+ The model can handle only a limited class of graphical objects; care has to be taken in defining the starting and end points of
a graphical object with more than one starting or end point. Examples are shown in Figure 2.3(a-b). As a result, much time and effort may have to be spent in defining such graphical objects. Another limitation in constructing a graphical object is that only two points of concatenation for each primitive are available.

![Figure 2.3 Examples of graphical objects that require extra care in defining the starting and end points with a lineal external representation model](image)

The model cannot handle regional graphics **conceptually**.

+ The model does not have a well-established concept of identification. Identification refers to the selection of a graphical object on a terminal screen.

+ The model encourages different notations, and the reader must understand the notation before understanding the concepts. In particular, different researchers use different ways to parse pictures.

**Regional external representation model**

**Virtues:**

+ This model provides a natural way to represent graphical objects which depict two-dimensional information.
+ The model is suitable for regional graphics without further modification.
+ The model supports operations that are simple and clear. Rules are natural to use and remember.
+ The model's use of the definition frame permits a consistent concept of identification.
+ The model facilitates the calculation of the area of a graphical object (refer to Appendix C).

Limitations:
+ The model does not lend itself to establish connectivity between different objects.
+ The model requires new concepts and so must be developed from first principles.

It can be concluded that each model has its own areas of application: the lineal model has its use in the field of pattern picture recognition whereas the regional model has its use in the field of computer graphics. One can further demonstrate the suitability of the regional representation model for computer graphics by the following argument:

The basic learning of a child begins with the alphabet and the digits.

\[
\text{alphabet} = \{a, b, \ldots, y, z\} \\
\text{digits} = \{0, 1, \ldots, 8, 9\}
\]

From the alphabet (the smallest division in a language), words are constructed. From words, sentences are constructed. Using the same principle, a computer graphics language, LIG has been implemented [Sc76a, Sc78]. An analogy can be made between the natural language and a computer graphics language: a set of graphical primitives comparable to the
characters in the alphabet is chosen. Primitives and characters are described on unit squares (Figure 2.4). A graphical object composed of graphical primitives corresponds to a word composed of characters. A complex graphical object composed of graphical objects is analogous to a sentence composed of words. Hence, this scheme, by the use of a regional external representation model, achieves good man-machine communication. In the sequel, only graphical objects using the regional external representation model are considered.

\[
\begin{array}{c}
\text{AC1G6d}
\end{array}
\]

Figure 2.4 Digits and characters described on squares
3. INTERNAL AND EXTERNAL REPRESENTATIONS OF REGIONS

The use of the regional external representation model for graphical data facilitates the definition of an opaque graphical object. In order to handle this type of graphical object, a thorough understanding of the internal and external representations of regions or areas is required.

A number of definitions or representations of regions have been proposed in various fields. Unfortunately, many of them are only suitable for the authors' particular application. However, all these representations can be classified and unified within a general scheme for solid-area graphical objects.

Towards this end, the description and classification of the internal representations of regions is presented in the following sections and is followed by a formal description of a region in order to unify the concepts.

3.1 Background

Cook [Co67] has provided a definition for plane regions: a region means a connected plane area bounded by one or more closed curves. A mathematical description of a region is the following: a set is called a region if it is the union of an open connected * set with none, some, or all its boundary points. If none or some of the boundary points are included, the region is called an open region. If all the boundary points are included, the region is called a closed region (see e.g. Apostol [Ap73]).

* A space S is called disconnected if $S = A \cup B$, where A and B are disjoint nonempty open sets in S. We call S connected if it is not disconnected [Ap73].
It is possible to distinguish between two basic types of regions: a simply connected region (SCR) and a multiply connected region (MCR). A simply connected region [Ja60] is defined to be a region such that any closed curve within it can be deformed continuously to a point of the region without leaving the region. Alternatively, it can be defined as a region such that no closed curve lying entirely within the region can enclose a boundary point of the region (Figure 3.1a). A multiply connected region [Ja60] is a region which is not simply connected (Figure 3.1b).

![Simply Connected Region](image1)

(a) a simply connected region  (b) a multiply connected region

Figure 3.1 Examples of different types of regions

The above definitions and classification for regions are too general for the work described here because closed curves should be specified to be simply closed curves. A simply closed curve (a Jordan curve, Apostol [Ap73]) is one which is closed and has no self-intersections (Kreyszig [Kr72]). Some examples of different types of curve are given in Figure 3.2(a-b).

![Closed Curve](image2)

(a) a closed curve  (b) a simply closed curve

Figure 3.2 Examples of different types of curves
3.2 Internal Representations of Regions

In the previous section some definitions and a classification of regions have been presented. Numerous techniques for representing regions have been reported in the literature. Basically, there are three major ones:

1) The 'encoding of the boundaries' scheme (the conventional method),
2) the 'parallel coding' scheme,
3) the 'skeleton' scheme.

3.2.1 The 'encoding of the boundaries' Scheme

A region can be defined by simply closed curves. Hence the task of representing a region is reduced to that of a curve. Numerous techniques for boundary representations have been reported in the literature. Some representations retain all of the information about the boundary, while others preserve only what is of interest for a particular application. A few of the important references on this subject are given here.

Each simply closed curve is represented by consecutive line segments. Each line segment, called the boundary section (Loomis [Lo65], Burton [Bu77]), is completely determined by two ordered pairs defining its end points (Figure 3.3). Freeman [Fr61a,Fr61b,Fr62,Fr74] was the first to propose the use of a string of integer code for contour representation (Figure 3.4(a-b)). Detailed discussion on boundary representations and descriptions can be found in Aggarwal [Ag77].

---

Figure 3.3 An example of a boundary section enclosed within a boundary section rectangle
(a) Eight possible directions, numbered from 0 to 7

(b) An example of a contour-line chain (CHAIN = 01024353667)

Figure 3.4 Freeman's encoding scheme
3.2.2 The 'parallel coding' Scheme

Dudani [Du76], Merrill [Me73], Ram [Ra76] are researchers whose work fall into this type of representation scheme for regions. The understanding of the concepts underlying this scheme can be achieved by reviewing Merrill's work.

A review of Merrill's work

A tightly closed boundary (TCB) is defined to be a boundary which has been augmented by additional points at extrema and inflections according to certain rules. A point of inflection is defined when three consecutive line segments exist, the centre segment being horizontal, and the other two are pointing in opposite directions. The point of inflection is then the left endpoint of the horizontal segment. The rules are simple:

1) Each horizontal test line must pass through an even number of boundary points at each extremum. It is necessary to duplicate the extremum at which this condition is not satisfied (Figure 3.5a).

2) Each horizontal test line must pass through an odd number of boundary points at each inflection. However, it is necessary to duplicate the point of inflection at which this condition is not satisfied (Figure 3.5b).

Once the TCB is obtained, it is partitioned into sets so that each set contains the x coordinates of points having the same y coordinate. Each of these sets is then sorted in ascending order of x and is called the y-partition of the TCB. Using the y-partitions of the TCB, a region can then be defined as follows:

\[
\text{Region, } R = \{y_{\min}, y_{\max}, Y_1, Y_2, \ldots, Y_n\}
\]

where \(y_{\min}\) is the smallest y-coordinate of the boundary,

\(y_{\max}\) is the largest y-coordinate of the boundary,
(a) augment an additional point to each extremum when necessary

(b) augment an additional point to each point of inflection when necessary

(c) an example to clarify notations used

Figure 3.5 Merrill's scheme
\[ n = y_{\text{max}} - y_{\text{min}} + 1, \]

\( y_i \) is the \( y \)-partition for the test line, \( y'_i \).

\( y_i \) can be expressed in the form:

\[ y_i = (r_i, x_1, x_2, \ldots, x_j, \ldots, x_{r_i}) \]

where \( r_i \) is the number of \( x \) coordinates, (always an even number).

\( x_j \) are the points of the TCB having \( y_i \)-coordinates.

Example (Figure 3.5c):

\[ r_i = 4, i = 3 \]

\[ y_3 = (r_3, x_1, x_2, x_3, x_4) \]

\[ y_3 = (4, 1, 3, 3, 12) \]

There is an implicit rule that has not been discussed by Merrill. Whenever an inflection is augmented, the point to be augmented must be the 'last' point associated with the inflection (refer to Figure 3.5b).

This is the so called parallel coding scheme. The principles of this scheme are used, with small modifications, in many hatching algorithms, filling algorithms and representations of regions. This scheme is particularly suitable for computing areas and intersection properties. Besides, it has reasonable storage requirements and provides an efficient format for information retrieval.

3.2.3 The 'skeleton' Scheme

The skeletal representation of planar regions proposed by Pfaltz and Rosenfeld [Pf67] is unconventional. Basically, a number of squares are fitted to the inside of a region in such a way that every inside point is contained in a square or squares. The square to be fitted must be the largest possible square in order to obtain the smallest possible total
number of squares. The centre of each square is its position and is assigned a value indicating the size of the square. The final form of the figure has numbers attached at the corresponding positions (Figure 3.6(a-b)) and is called the skeleton of the figure.

(a) A region defined by points P, Q, R, S with radii 3,2,1 and 0.[Pf67]

(b) Skeletons of two irregular regions (bounded by X's) [pf67]

Figure 3.6 Pfaltz and Rosenfeld's scheme
Rosenfeld asserts that this scheme has advantages when it is necessary to test repeatedly if a point is inside or outside a region. This kind of test is required in hatching (Figure 3.7). Multiply-connected region can be handled without modifications. Moreover, set-theoretic operations on regions present no difficulties.

According to Pfaltz and Rosenfeld, the skeleton scheme requires more storage space than does the 'encoding of the boundaries' method. In addition, it is difficult, if not impossible, to determine the area or the perimeter of a region.

Figure 3.7 A hatched map obtained through the use of Pfaltz and Rosenfeld's scheme [Pf67]

The skeleton scheme of Pfaltz and Rosenfeld is a specialized method for representing a region. It does not lend itself readily for use with hatching or filling algorithms.
3.3 A Formal Representation of a Region

A region can be defined by a two-tuple (an ordered pair):

\[ R = \langle I, E \rangle \]

where \( I \) denotes an internal representation of a region \( R \),

E denotes a model for the external representation of a region \( R \).

E can be described by a three-tuple:

\[ E = \langle A, B, F \rangle \]

where \( A \) is a scheme to produce a visual effect of the existence of a
region (such as hatching, colouring), \( A \) may be the empty set
(Loomis [Lo65]),

B denotes a set of simply closed curves,

F denotes a definition frame within which a region is defined.

If \( B = \{b_1\} \), then \( R \in \text{SCR} \),

if \( B = \{b_i | i > 1\} \), then \( R \in \text{MCR} \),

where SCR denotes a set of simply connected regions,

MCR denotes a set of multiply connected regions.

For any boundary, \( b_i \):

\[ b_i = \langle K, D, Q \rangle \]

where \( K = \Phi_s \Phi_k \) denotes a simply closed curve formed by a set of consecu-
tive line segments, \( \{s_k | k = 1, \ldots, \text{nthside}\} \). Mathematically, \( k = 1, 2, \ldots \); when \( k = 1 \) or \( 2 \), a closed polygon of zero area is formed;
thus in practice, \( k \geq 3 \). The circles associated with the brackets
are used to emphasize the fact that the curve is closed,

D = \{a, c\}, denoting a set of two elements specifying the direction
along which the boundary is defined; 'a' denotes an anticlockwise
direction and 'c' denotes a clockwise direction (Figure 3.8),

Q = \{1, r\}, denoting a set of two elements specifying on which side
the region lies when moving along the boundary according to the specified element of \( D \); \( l \) denotes that the region lies to the left of the boundary, \( r \) denotes that the region lies to the right of the boundary (Figure 3.9).

Each line segment, \( s_k \) can be specified by two ordered pairs:

\[
s_k = \langle (x_k, y_k), (x_{k'}, y_{k'}) \rangle
\]

where

\[
k' = \begin{cases} 
  k + 1 & \text{when } k \neq \text{nthside} \\
  1 & \text{when } k = \text{nthside}
\end{cases}
\]

(Figure 3.10)

Figure 3.8 Direction convention

To curve A, the region is on the left, to curve B, the region is on the left.

Figure 3.9 A convention used for defining a region
Figure 3.10 A simply closed polygon
4. THE CONCEPTS AND TECHNIQUES OF HATCHING

Hatching is one way to achieve an external representation of a region. In addition, the capability to hatch an arbitrary simply closed polygon is found useful in many applications.

There has been some confusion in the use of the terms hatching, shading and filling. Some researchers use the term 'shading' in a context for which the more precise term 'hatching' should have been used. The following definitions will provide a clear description of hatching, filling and shading.

4.1 Definitions

Hatching refers to the drawing of a set of parallel lines of uniform thickness at a given angle and spacing (called the hatching lines) to give a visual effect of the existence of a region or to distinguish one region from another (Figure 4.1).

![Hatching Pattern](image.png)

Figure 4.1 An example of a hatching pattern

Filling refers to the assigning of a certain gray-level, symbol or colour to each pixel belonging to a region (Figure 4.2).
Figure 4.2 An example of a region that has been filled

Shading refers to the filling or the hatching of a region in order to depict a three-dimensional object on a two-dimensional display surface (Figure 4.3).

Figure 4.3 An example of a shaded object [Ne78]

4.2 A Short Review of Hatching Algorithms

Pavlidis [Pa78a, Pa78b] has given an excellent presentation on various filling algorithms for raster graphics. However, neither he nor others seem to have investigated hatching algorithms systematically.

A hatching algorithm was implemented by Phillips [Ph76] using ideas proposed by Dwyer [Dw67]. An example of an output is shown in
Figure 4.4. Phillips, however, does not give any details of the algorithm.

Figure 4.4 Shading of compound polygons (Phillips [Ph76])

Another hatching-filling algorithm, developed by Ison [Is73], is oriented toward vector graphics. This algorithm will be called a hatching-filling algorithm because it is capable of drawing parallel lines as well as putting out desired symbols, called filling symbols, for a given polygon. An example of a plot output is shown in Figure 4.5, where LABEL 1 and LABEL 2 are hatching patterns whereas LABEL 3 is a filling pattern. As Ison states, one of the restrictions for this algorithm is that a hatching line which overlaps exactly with an outline segment may not be included. As is shown in LABEL 1 and 2, the outline segments are constructed in such a fashion that no hatching line overlaps with an outline segment exactly. This restriction reduces the usefulness of the algorithm on many occasions. Furthermore, the more outline segments a figure is made
up of, the higher the chance is for overlapping. Also, when the outline segments are not perpendicular to each other, no test is given as to whether a filling symbol will be partially outside the outline or not.

Figure 4.5 Example - Plot output (Ison [Is73])
Pfaltz and Rosenfeld [Pf67] proposed a hatching algorithm for an arbitrary planar region represented in skeleton form. However, their work is focused on the representation of region. Details of the algorithm are not given (Figure 3.7).

4.3 General Concepts of Hatching

Hatching an arbitrary region defined by a set of simply closed consecutive line segments can be considered to be a repetition of a three-step task:

1) with a specified angle, find all the intersections of a line with the outline segments describing the region;

2) with the obtained set of intersecting points, manipulate it, if necessary, according to a set of rules to ensure a correct set of points for step 3;

3) with the elements of the set obtained in step 2, draw lines to produce a hatching line.

By relaxing the definition of hatching lines to allow non-parallel lines, a broader class of hatching patterns such as those shown in Figures 4.6(a-b) can be identified. Thus, hatching can be defined to

![Diagram](image_url)

(a) (b)

Figure 4.6 Hatching patterns obtained via the use of a windowing technique
be a set of arbitrary curves with equal distance from each other. With this definition, a hatching pattern can be obtained by a windowing technique. Windowing refers to the formation of a domain enclosed by a simply closed polygon. Here, the boundary of a region to be hatched is the window. For example a pattern (Figure 4.6b) can be obtained by a set of concentric circles or a set of arbitrary defined shapes [Pe78, Ve79], expanding out from or shrinking to a point. Such patterns will not be considered here.

Most of the hatching algorithms are limited to hatching a simply connected region. Whenever a multiply connected region is to be hatched, it has to be converted to a simply connected region. This can easily be done by introducing invisible cuts or virtual edges (Figure 4.7). The number of cuts corresponds to the number of holes inside the region.

![Conversion of a multiply-connected region to a simply connected region](image)

**Figure 4.7** Conversion of a multiply-connected region to a simply connected region

### 4.4 The Conceptual and Mathematical Elements for Hatching Algorithms

Before the developed hatching algorithm is discussed in detail, the following sections will be helpful to the understanding of both the developed hatching algorithm as well as other algorithms.
4.4.1 Transformations

In order to have simple computations as well as a consistent analysis, transformation of all coordinates and equations to a new coordinate system with its abscissa parallel to the hatching lines is necessary.

Consider an arbitrary region defined by a boundary shown in Figure 4.8. A preliminary step towards the construction of a hatching pattern is to find all points of intersection of the boundary segments and lines shown. For convenience, these lines, used for generating hatching lines, are called generative lines. The equation for each line $L$ in the $x$-$y$ plane is of the form: $y = mx + c$ where $m$ is its slope, and $c$ its $y$-intercept. The same line $L$ can be described, without loss of information, by $y' = c'$ in the $x'$-$y'$ plane with its abscissa parallel to $L$. Computations using $y' = c'$ rather than $y = mx + c$ are much easier and simpler. Besides, the computations are more consistent since all generative lines are parallel to the abscissa and the calculations involved always deal with solving for points of intersections between horizontal lines and boundary segments.

![Diagram showing generation of hatching lines](image)

Figure 4.8 Generation of hatching lines
The $x'-y'$ plane is effectively obtained in two ways:

1) by rotating the $x$-$y$ plane $\theta$ radian about a reference point, or

2) by rotating the region defined in the $x$-$y$ plane $-\theta$ radian about a reference point.

In the sequel, all manipulations of coordinates are carried out in the rotated plane, the $x'$-$y'$ plane, unless otherwise stated. A horizontal line refers to a line parallel to the abscissa of the plane associated with a pattern or a line parallel to the hatching lines in the untransformed coordinate system. In particular, a horizontal generative line in the rotated plane is called a scanline, or a test line.

4.4.2 Intersection between two lines

As stated earlier, a step towards the construction of a hatching pattern is to obtain points of intersection between boundary segments and generative lines; for this, the understanding of how a point of intersection is defined and obtained is essential.

Mathematically, in a finite region two lines can only intersect in three ways: at no point, at one point, and at an infinite number of points. For the work discussed here, infinite number of points of intersection will be treated as no intersection. To be precise, a systematic analysis with one of the lines being horizontal, will be presented.

Point of intersection between a line segment and a horizontal line

A line segment $S$ is defined by its end points called 'head' and 'tail'. Mathematically, $S = \langle(x_h, y_h), (x_t, y_t)\rangle$. Let the equation of a horizontal line $L$ be $y = k$. 
Theorem 4.1

A point of intersection \((x^*, y^*)\) between \(L\) and \(S\) exists only if:

a) \(y_h \neq y_t\) (they are not parallel)

b) \(y_h \geq k \geq y_t\) or \(y_h \leq k \leq y_t\) (they meet).

It is then given respectively by:

\[
(x_t + \frac{(k - y_t)(x_h - x_t)}{y_h - y_t}, k) \text{ or } (x_h + \frac{(k - y_h)(x_h - x_t)}{y_h - y_t}, k)
\]

4.4.3 Topological Properties

The principle of most hatching algorithms and representations for regions is based on two well-known theorems:

Theorem 4.2

If a test line is drawn from a point outside a given region defined by a closed polygon to a point which is also outside the region, then the number of crossings between the line and the segments defining the boundary must be an even number (Figure 4.9(a-b)).

![Figure 4.9](image)

(a) (b)

Figure 4.9 Two examples satisfying Theorem 4.2

Theorem 4.3

If a test line is drawn from a point outside a given region defined by a closed polygon to a point which is inside the region, then the number of crossings between the line and the segments defining the boundary must be an odd number (Figure 4.10(a-b)). For proofs, see
It is well-known that the above topological properties of a closed polygon always hold unless the following restrictions are violated.

1) All boundaries must be simply closed.
2) The test line does not pass through any singular points.

A singular point can either be a critical point or a point of inflection. For an example of a critical point, consider a polygon composed of line segments as shown in Figure 4.11. The horizontal line HBH' cuts the section ABC of the boundary at two coinciding points of intersection at B on AB and another at B on BC. Provision has to be made for this kind of situation by 'fusing' them together. Point B is the critical point.

Figure 4.10 Two examples satisfying Theorem 4.3

Figure 4.11 An example of a critical point
For an example of a point of inflection, consider the polygon shown in Figure 4.12. The horizontal line \( HBC' \) cuts the section \( ABCD \) of the boundary at the two points \( B \) and \( C \), one point lying on \( AB \) and the other on \( CD \). According to the discussion in Section 4.4.2, the infinite number of points of intersection on \( BC \) will be treated as no intersection. Hence, \( BC \) appears to be non-existent and points \( B \), \( C \) should be treated as one point by fusing them together. Point \( B \), the left most point, is the point of inflection. Figure 4.13(a-h) contains examples, all of which satisfy Theorem 4.2 and Theorem 4.3. However, a more formal examination of critical points and points of inflection is necessary.

![Diagram of polygon ABCDEA with points B and C and line HBCH']

Figure 4.12 An example of a point of inflection

![Examples of points which are not critical nor points of inflection](a), (b)
4.4.1 Definitions for Singular points

Definition: Critical point (Figure 4.14(a-b))

Given two consecutive line segments $S_k, S_{k+1}$ of the same boundary. If $(y_{k+2} > y_{k+1} > y_k)$ or $(y_{k+2} < y_{k+1} < y_k)$ then $(x_{k+1}, y_{k+1})$ is called a critical point.

Figure 4.14 A critical point, $(x_{k+1}, y_{k+1})$

Definition: Point of inflection (Figure 4.15, 4.16)

Given three consecutive line segments $S_k, S_{k+1}, S_{k+2}$ of the same boundary. Let $P_{k+1} = (x_{k+1}, y_{k+1})$ and $P_{k+2} = (x_{k+2}, y_{k+2})$

If $(y_{k+3} > y_{k+2} \text{ and } y_{k+2} = y_{k+1} \text{ and } y_{k+1} > y_k)$

or $(y_{k+3} < y_{k+2} \text{ and } y_{k+2} = y_{k+1} \text{ and } y_{k+1} < y_k)$

then one of the two points $P_{k+1}, P_{k+2}$, whichever lies on the left, is a point of inflection.

Figure 4.15 A point of inflection, $(x_{k+1}, y_{k+1})$
Figure 4.16 A point of inflection, \((x_{k+2}, y_{k+2})\)

For completeness, the min-max points are defined here.

**Definition: Min-Max points**

Given two consecutive line segments \(S_k, S_{k+1}\) of the same boundary.

If \((y_{k+1} > y_k \text{ and } y_{k+1} > y_{k+2})\) then \((x_{k+1}, y_{k+1})\) is a maximum.

If \((y_{k+1} < y_k \text{ and } y_{k+1} < y_{k+2})\) then \((x_{k+1}, y_{k+1})\) is a minimum.

Minimum and maximum points are neither critical points nor points of inflection.

**An example**

In order to clarify the definitions for critical points, points of inflection and min-max points, an example is given in Figure 4.17. G, F are minimum points. D is a point of inflection, A is a critical point and I, H, B, C, E are none of the above.

Figure 4.17 An example to identify critical points, points of inflection, min-max points
4.4.5 Generation of a Hatching Line

A hatching pattern is a set of hatching lines, each of which is generated in a number of steps. Each of these hatching lines is associated with a scanline $y_k$ running across the region to be hatched. The scanline $y_k$ will intersect the boundaries at least two times as long as all the boundaries are simply closed and the scanline $y_k$ lies between $y_{\text{min}}$ and $y_{\text{max}}$ (Figure 4.18). $y_{\text{min}}$, $y_{\text{max}}$ are the minimum and maximum $y$ coordinates of an encasing rectangle for the region. The encasing rectangle is one with its sides parallel to the $x$ and $y$ axes. Using the conditions for the intersection of two lines, a set of $x$ coordinates of all points of intersection between outline segments and the scanline $y_k$ can be derived.

![Diagram](image-url)

Figure 4.18 An encasing rectangle associated with a hatching pattern
The set $Y^k$

Denote $Y^k$ to be the set of $x$-coordinates of all points of intersection between the scanline $y_k$ and the boundaries (Figure 4.19).

$$Y = y_k$$

It can be expressed as:

$$Y^k = \{x^k_p, \ldots, x^k_q \mid 1 \leq p < q \leq 2, i < j\}$$

where the subscripts $p, q$ denote the order in which the $x$ coordinates are generated, the superscripts $i, j$ denote the positions of the elements in the modified sets of $Y^k$.

Note that $x^k_p$ should be written as $i^k(x^*_p)$. Refer to Theorem 4.1 in Section 4.4.2.

In Figure 4.19, $Y^k = \{x^*_1, x^*_2, x^*_3, x^*_4, x^*_5\}$

$$= \{x_6, x_5, x_0, x_0\}$$

The set $Y^{k'}$

The elements of $Y^k$ first have to be sorted in ascending order of $x$ for a hatching algorithm. The sorted set becomes:

$$Y^{k'} = \{x^{k'}_p, \ldots, x^{k'}_q \mid i^{k'}_p \leq i^{k'}_q, 1 \leq p < q \leq 2, i < j\}$$
In Figure 4.19, \( Y^k' = \{ (x_1)^k', (x_5)^k', (x_3)^k', (x_1)^k' \} \)

\[ = \{ x_0, x_0', x_5, x_6 \} \]

The set \( Y^k'' \)

The resulting set \( Y^k' \) may still not be in the desired form. Now the provision mentioned earlier must be made for singular points to ensure a correct set of x coordinates for hatching. For each scanline \( y_k \), there are two identical x coordinates describing a critical point. If such a point exists, one of the coordinates is removed from the set \( Y^k' \).

For an inflection, two consecutive boundary points are involved and the right most point is removed from the set \( Y^k' \). In Figure 4.19, \( x_6 \) is dropped. In both cases, the two x coordinates are effectively fused into one and the provision is thus referred to as the 'fusing' technique.

Once the set \( Y^k' \) has undergone the process of 'fusing', the set \( Y^k'' \) is obtained. This set is of the form:

\[ Y^k'' = \{ x_p^k'', \ldots, x_q^k'' \mid x_p^k'' < x_q^k'', 1 <= p < q =< 2, i < j \} \]

The number of elements in the set \( Y^k'' \) is always even.

In Figure 4.19, \( Y^k'' = \{ (x_4)^k'', (x_3)^k'' \} = \{ x_0, x_5 \} \)

The set \( HL^k \)

Once the set \( Y^k'' \) is obtained, a hatching line \( HL^k \) associated with the scanline \( y_k \) can be defined. A hatching line is defined by one or more segments (Figure 4.20). \( HL^k \) is of the form:

\[ HL^k = \{ (x_p^k', y_k), (x_q^k', y_k) \mid i = 1, 3, \ldots \} \]

In Figure 4.19, \( HL^k = \{ (x_4)^k'' , y_k), (x_3)^k'' \} \)

\[ = \{ (x_0, y_k), (x_5, y_k) \} \]
Figure 4.20 An example with each hatching line defined by one or more segments
Notice, without fusing, an even number of singular points would give an even number of intersections at the boundaries, resulting in an incorrect hatching line composed of two zero length segments (Figure 4.21). Thus, an even number of intersections at the boundaries is a necessary but not a sufficient condition for generating a correct hatching line. The process of fusing will guarantee a correct hatching line.

![Diagram showing a scanline with vertices (x_a, y_a), (x_b, y_b), (x_c, y_c), and (x_d, y_d).](https://via.placeholder.com/150)

**Figure 4.21 An example with an incorrect HL:**

\[
\text{HL} = \{(x_a, y_a), (x_b, y_b), (x_c, y_c), (x_d, y_d)\}
\]

Displacing slightly each vertex that is either a critical point or a point of inflection is another solution to this problem. This method is satisfactory if the vertices are represented carefully, using only integer arithmetic (Newman and Sproull [Ne78]).

### 4.5 The Developed Hatching Algorithm

The first step in the development of an algorithm for a certain task is to survey the related literature. A summary of all the weaknesses or restrictions in proposed schemes then may result in a table of problems to be resolved. This method is being used in the development of the algorithm.
A Formal Representation of a Hatching Pattern in the Developed Hatching Algorithm (Figure 4.22)

A hatching pattern, HP is defined by a two-tuple:

\[ HP = <R, D> \]

where \( R \) is a set of realization regions; a realization region is a set of regions defined within a definition frame with the same hatching pattern.

\[ R = \{ r_i \mid i = 1, 2, \ldots, n \} \]

with \( n \) the maximum number of realization regions defining the hatching pattern.

\( D \) is the definition plane on which the hatching pattern is defined. It may be considered as a window. A window is a simple domain enclosed by a set of consecutive edges that form a simply closed polygon. Hence, \( D \) can be expressed as follows:

\[ D = \{ \text{edge}_m \mid m = 1, \ldots, n' \} \]

where \( n' \) is the number of edges used to form the window. The circles associated with the brackets emphasize the fact that the boundary must be closed.

\( r_i \), an element of \( R \), is a realization region consisting of one or more regions defined in a definition frame.

Each realization region \( r_i \) can be described completely by a two-tuple:

\[ r_i = <B, A> \]

where \( B \) is a set of boundaries defining a realization region. \( A \) is a three-tuple of attributes for a visual effect. Each boundary is defined by a simply closed curve composed of line segments.

Define \( s_{i,j,k} \) as the \( k^{th} \) line segment of the \( j^{th} \) boundary of the \( i^{th} \) realization region. Each line segment, \( s_{i,j,k} \), can be described by its head and its tail: \( s_{i,j,k} = <h_{i,j,k}, t_{i,j,k}> \). Both the head and the tail
Figure 4.22 A hatching pattern composed of 3 realization regions, $r_1$, $r_2$, $r_3$. 
of the line segment can be described by the x, y coordinates in the form:

\[ h_{i,j,k} = \langle x_{i,j,k}, y_{i,j,k} \rangle \]
\[ t_{i,j,k} = \langle x_{i,j,k}, y_{i,j,k} \rangle \]

where \( k' = \begin{cases} k + 1 & \text{when } k \text{ does not denote the last line segment} \\ 1 & \text{when } k \text{ denotes the last segment} \end{cases} \)

\( <x, y> \) are the coordinates of a point in the x-y plane.

A is defined by a three-tuple and specifies the visual effect of a realization region.

\[ A = <g, d, t'> \]

where \( g \) is the gap or the spacing between the hatching lines,
\( d \) is the angle of direction of the hatching lines,
\( t' \) is the thickness of each hatching line.

4.5.2 The Algorithm HATCH

Purpose:

To hatch a realization region defined by simply closed polygons and to manipulate (scale, rotate and translate) the realization region if desired.

Type of routine:

A self contained program written in FORTRAN.

Restrictions:

1) The polygons are simply closed:
   a) no self-intersection,
   b) the starting and end point must be identical,

2) the intersection of the regions defining the realization region is the empty set.

How to use:

The user can invoke the routine either in a Fortran program or
in an LIG program. The calling sequence is:

CALL HATCH (NO_OF_BOUNDARIES, NO_OFPTS_IN_EACH_BOUNDARY,
XCOORS_OF_ALLPTS, YCOORS_OF_ALLPTS, HATCH_DIR,
HATCH_GAP, WIDTH_OF_HATCH_LN, X_LOC, Y_LOC,
XSC, YSC, ORIENT)

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_OF_BOUNDARIES</td>
<td>Integer</td>
<td>The number of boundaries defining the realization region.</td>
</tr>
<tr>
<td>NO_OFPTS_IN_EACH_BOUNDARY</td>
<td>Integer</td>
<td>The array containing the number of points defining each boundary.</td>
</tr>
<tr>
<td>XCOORS_OF_ALLPTS</td>
<td>Real</td>
<td>The array containing the x coordinates of all points defining the realization region.</td>
</tr>
<tr>
<td>YCOORS_OF_ALLPTS</td>
<td>Real</td>
<td>The array containing the y coordinates of all points defining the realization region.</td>
</tr>
<tr>
<td>HATCH_DIR</td>
<td>Real</td>
<td>The direction of the hatching lines in radians.</td>
</tr>
<tr>
<td>HATCH_GAP</td>
<td>Real</td>
<td>The spacing between hatching lines.</td>
</tr>
<tr>
<td>WIDTH_OF_HATCH_LN</td>
<td>Integer</td>
<td>The multiple of the default width.</td>
</tr>
<tr>
<td>X_LOC</td>
<td>Real</td>
<td>The x coordinates of the realization region with respect to (.5,.5).</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>TYPE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Y_LOC</td>
<td>Real</td>
<td>The y coordinates of the realization region with respect to (.5,.5).</td>
</tr>
<tr>
<td>XSC</td>
<td>Real</td>
<td>The scale factor of the abscissa of the realization region with respect to (.5,.5).</td>
</tr>
<tr>
<td>YSC</td>
<td>Real</td>
<td>The scale factor of the ordinate of the realization region with respect to (.5,.5).</td>
</tr>
<tr>
<td>ORIENT</td>
<td>Real</td>
<td>The angle of rotation of the realization region about (.5,.5) in radians.</td>
</tr>
</tbody>
</table>

**4.5.3 A Description of the Developed Hatching Algorithm**

**4.5.3.1 A Short Note on the Manipulation of a Realization Region**

This section will aid in the understanding of the description of the hatching algorithm. Consider the realization region shown in Figure 4.23a and assume that the hatching lines are at an angle of HATCH_DIR radians with the x-axis. The manipulation of the realization region is a two-step task:

1) transform the vertices defining the realization region. This transformation is associated with the angle of rotation ORIENT of the realization region (Figure 4.23b),

2) hatch the transformed realization region with the generative lines at the angle of (HATCH_DIR + ORIENT) radians with the x-axis (Figure 4.23c).
Figure 4.23 Manipulation of a realization region

(a) a realization region

(b) transformation of the vertices defining the realization region

(c) the transformed realization region
4.5.3.2 Description of the Algorithm HATCH

1) Set HATCH_DIR ← HATCH_DIR + ORIENT

2) If the five attributes <X_LOC, Y_LOC, XSC, YSC, ORIENT> are not the default values, transform the vertices that define the realization region, otherwise no transformation is necessary.

3) Rotate the realization region to a new coordinate system with its abscissa at an angle of HATCH_DIR radians with the x-axis: if HATCH_DIR ≠ 0, then rotate the coordinates of all data points defining the realization region by - HATCH_DIR, else ignore and go to the next step.

4) Repeat step 5 for j = 1, ..., NO_OF_BOUNDARIES.

5) Find the critical points and the points of inflection of boundary j:
   each vertex of the closed polygon defined by the boundary j is examined to see if it is a critical point or a point of inflection.

6) Determine YMAX and YMIN, the maximum and minimum y-coordinates of the encasing rectangle for the realization region.

7) Compute the number of scanlines, NDIV:
   \[ NDIV = \text{INT} \left( \frac{YMAX - YMIN}{\text{HATCH_GAP}} \right) \]

8) Perform hatching: repeat step 9 for k = 1, ..., NDIV.

9) Repeat step 10 to 11 for NW = 1, WIDTH_OF_HATCH_LN.

10) Obtain the general equation for \( y^k \):
    \[ y = y^k = YMAX - k \times \text{HATCH_GAP} - (NW - 1) \times \text{default thickness of a line} \]

11) Compute the scanlines' points of intersection with the boundaries to obtain the set \( Y^k \), sort it and obtain \( Y^{k'} \). With the provisions for critical points and points of inflection, obtain \( Y^{k''} \), from which \( HL^k \) is then derived.

12) Rotate the realization region back to the original coordinate system:
    if HATCH_DIR ≠ 0 then rotate the coordinates of all data points defining the realization by HATCH_DIR, else return.
4.5.4 Evaluation of the Developed Hatching Algorithm

Virtues:

1) A realization region, composed of one or more multiply connected regions, can be hatched (Figure 4.24).
2) The complement of a realization region can be obtained readily by assigning a virtual unit square (Figure 4.25).
3) The thickness of, inclination of and the spacing between hatching lines are user defined (Figure 4.26(a-b)).
4) A realization region can be manipulated (translated, rotated and scaled) (Figure 4.27).
5) The area of a realization region can be calculated easily because it is described by encoding its boundaries [Appendix A].

Limitations:

1) Although the algorithm can easily be modified into a hatching-filling one, it is suitable for vector graphics only.
2) The algorithm cannot handle polygon clipping*.

4.5.5 Future work

1) To convert this algorithm into a hatching-filling one is desirable.
2) The efficiency of the algorithm should be evaluated and improved if necessary.
3) The algorithm should be modified to handle polygon clipping (see, for example Newman and Sproull [Ne78]).

* Any part of a polygon that lies outside the screen gets clipped off. This process is referred to as polygon clipping.
Figure 4.24 A realization region composed of a multiply connected region
Figure 4.25 Complement of a realization region
(a) hatching lines of the default thickness

Figure 4.26 Hatching patterns with associated sets of hatching lines at different inclinations, spacings and thicknesses
(b) hatching lines of a multiple of the default thickness

Figure 4.26. Hatching patterns with associated sets of hatching lines at different inclinations, spacings and thicknesses
Figure 4.27 Manipulation of a realization region
5. A SHORT REVIEW OF TWO GRAPHICS LANGUAGES

Although researchers have felt the need for regional graphics, most work has been limited to the use of existing line-drawing graphics packages to achieve the visualization of regions by the use of hatching or filling techniques. Newman and Sproull's [Ne75a] work is a typical example. Bracchi [Br71], based on the concept of data type, suggested a language for treating geometric patterns in a two-dimensional space. The language constructs and the capability of this hypothetical language seem to be reasonably good. However, the external representation of patterns or regions has been overlooked. Nake [Na71], using the regional external representation model, proposed a language, but it is limited to graphics, with no numerical capability. However, the underlying concept is good and it forms the skeleton of LIG as well as that of the proposed language. In the following sections, reviews and comments are given on Newman et al. and Bracchi et al.'s work.

5.1 Review of Newman and Sproull's Work [Ne75a]

Newman and Sproull discuss the design of gray-scale graphics software for raster scan display devices. With their package, the user can define and display line-drawn as well as solid-area graphical objects. Basically, it is an extension of the conventional line-drawing graphics package but with four new functions, BEGINFILL, ENDFILL, SETCOLOR, UPDATE.

The BEGINFILL and ENDFILL are functions that mark the start and the end of the filling operation. The SETCOLOR function is used for assigning a colour or a gray level to a given graphical object. A parameter list is associated with this function for gray level or colour selection. The UPDATE function is used for updating the pictures on the screen in order to describe the most recent picture stored internally.
Consider the construction of a filled green polygon shown in Figure 5.1 by the following instruction sequence:

BEGINFILL
SETCOLOR (GREEN)
MOVETO (XA,YA)
DRAWTO (XB,YB)
DRAWTO (XC,YC)
DRAWTO (XD,YD)
DRAWTO (XE,YE)
DRAWTO (XF,YF)
ENDFILL

MOVETO (Xi,Yi) and DRAWTO (Xj,Yj) are conventional line-drawing instructions. They are used to define the boundary of the object to be filled.

Figure 5.1 A filled polygon

The UPDATE function requires an efficient algorithm for handling the overlapping of opaque objects. Two algorithms, derived from hidden-surface algorithms are discussed in detail in [Ne75a].

5.2 Comments on Newman and Sproull's Work

A data type for opaque graphical objects is not used in this package. Rather, the package uses the functions BEGINFILL and ENDFILL to produce visually opaque objects by filling the areas enclosed by the boundaries. Because an opaque object is not associated with a name, the
display of the same object elsewhere on the screen will require a definition of the boundary again.

5.3 Review of Bracchi and Ferrari's Work [Br71]

CADEP (Computer Assisted Description of Pattern) is the language proposed by Bracchi et al. for treating geometric patterns in a two-dimensional space. It is a hypothetical language that has not been implemented. Although it has been designed for automatic mask generation, it is claimed to be useful in many applications such as placement problems, block diagram representations etc.

The language is an extension of Fortran with graphic-oriented Fortran compatible statements. There are three basic data type variables in CADEP, the conventional Fortran variables, geometric variables and graphic variables.

Just as each conventional Fortran variable has a value, so does each geometric variable and graphic variable have a value. However, the value of a geometric or graphic variable is the entire data structure representing the pattern associated with the variable. All variables except conventional Fortran have to be declared (e.g. GEOMETRIC GV1, GV2, ...). Each declared geometric variable represents a geometric entity lying in the same geometric plane. This plane is not bounded and points on it are defined by x and y coordinates. Complicated patterns can be created on the geometric plane by defining geometric primitives (defined to be the smallest geometric variables), by using geometric expressions and manipulation functions, and by allowing patterns to overlap.

There are eleven geometric primitives in CADEP. These are listed in Figure 5.2. All primitives are two-dimensional patterns and thus a line is a rectangle of very small width.
LNC \((X_1, Y_1, X_2, Y_2, W)\)

LNP \((X_1, Y_1, \rho, \theta, W)\)

ARC \((X_1, Y_1, X_2, Y_2, r, W)\)

CRCL \((X_1, Y_1, r)\)

SECT \((X_1, Y_1, r, \alpha_1, \alpha_2)\)

RECT \((X_1, Y_1, X_2, Y_2)\)

RECS \((X_1, Y_1, S_1, S_2)\)

POLY \((X_1, Y_1, X_2, Y_2, \ldots, X_N, Y_N)\)

FPLANE \(\) The entire geometric plane

HPLANE \((X_A, Y_A, \alpha)\)

EMPTY \(\) The complement of FPLANE

---

Figure 5.2 Geometric primitives in CADEP [Br71]
Three operations are allowed in the geometric plane, intersection, union and difference.

In order to describe physical objects whose shapes or patterns have been defined in the geometric plane, variables of type GRAPHIC are necessary. Since CADEP has been designed for automatic mask generation and so is concerned with different levels, the graphic data type is further subdivided into two data types, UNILEVEL and MULTILEVEL. Variables of type UNILEVEL represent physical two-dimensional entities lying in a particular graphic plane. Variables of type MULTILEVEL represent the same physical object and are composed of several unilevel patterns lying in different graphic planes. Examples of declaration statements for unilevel (lev1, lev2) and multilevel (mult1, mult2) variables respectively are:

\[
\text{LEVEL}(i) \text{ lev1, lev2, ... } \text{ where } i \text{ denotes a level and } i = 1, 2, 3, ...
\]

\[
\text{GRAPHIC mult1, mult2, ...}
\]

Unlike the geometric plane, the overlapping of patterns is not allowed in the graphic plane. The graphic plane is defined by the physical size of the display unit. Operators for unilevel and graphic variables are the union operator (+) and the concatenation operator (*) respectively.

5.4 Comments on Bracchi and Ferrari's Work

In computer graphics, the features provided for testing the relationships (such as distance, inclusion, ...) between patterns are not important in practice (for computer-aided design purposes). However, essential features such as the use of default values for geometric primitives are not specified.

A SETCOLOR function or other means of assigning colour or gray-level to a graphical object is not discussed. The distinction between
patterns or graphical objects using colours, gray-levels or hatching patterns is an important aspect of a regional graphics language. From the illustration of geometric primitives, it seems that hatching patterns may be proposed. It is not stated explicitly whether hatching patterns are used or how the patterns are specified in language constructs.

The distinction between the two data types: GEOMETRIC and GRAPHIC for the description of patterns is not necessary. Instead, one data type for defining and representing patterns is adequate. This can be done by allowing patterns to be defined using unmodified or modified primitives in a plane. This plane is finite and defined by the physical size of the display unit. Any part or even the entire pattern, not falling within the finite plane, are clipped. The use of this scheme (using the concept of definition frame) has been implemented successfully in LIG.

There are a number of good features in CADEP. The use of data types for describing and treating geometric patterns as a named unit datum is appropriate. Further, the concept of level, an important feature in a good regional graphics language, is found in CADEP.
6. A PROPOSED REGIONAL GRAPHICS LANGUAGE

The first step towards the creation and implementation of a high-level programming language is the definition of the language itself. This requires a clear statement of the purpose of the language. Towards this end, a formal description of a graphical object in the proposed language is presented. This is followed by a proposed set of primitives, operators and language statements. The conceptual and technical characteristics are discussed as well.

6.1 A Formal Description of a Graphical Object in the Proposed Regional Graphics Language, using the Regional External Representation Model

A graphical object may be defined by the four-tuple:

\(<N, D, R, X>\)

where \(N\) is the name of the graphical object,

\(D\) is the definition frame defined by a five-tuple of pairs of coordinates,

\(R\) is the coordinate pair defining the reference point of the graphical object,

\(X\) is the set of transformed graphical primitives used.

Let \(B \in X\), then \(B\) can be described by a two-tuple:

\(B = <P', O>\)

where \(P'\) is an element of the set of graphical primitive \(P\),

\(O\) is a subset of \(M\) which is the set of modification operators of the form:

\(M = \{<\text{ANGLE, parameter}_1>, <\text{SCALE, parameter}_2, \text{parameter}_3>, <\text{AT, parameter}_4, \text{parameter}_5>, \ldots\}\)

where \(\text{ANGLE, SCALE, AT}\) are modification operators.
Example: Consider the description of the graphical object shown in Figure 6.1.

![Diagram of a graphical object composed of opaque and line-drawn graphical primitives]

Figure 6.1 A graphical object composed of opaque and line-drawn graphical primitives

\[ \begin{align*}
N &= \text{Girl} \\
D &= \{(0., 0.), (1., 0.), (1., 1.), (0., 1.), (0., 0.)\} \\
R &= (0.5, 0.5) \\
X &= \{B_1, B_2, B_3, B_4, B_5, B_6, B_7\} \\
B_1 &= \langle \text{DISK}, O_1 \rangle \\
B_2 &= \langle \text{RTRIANGLE}, O_2 \rangle \\
B_3 &= \langle \text{LINE}, O_3 \rangle \\
B_4 &= \langle \text{LINE}, O_4 \rangle \\
B_5 &= \langle \text{LINE}, O_5 \rangle \\
B_6 &= \langle \text{LINE}, O_6 \rangle \\
B_7 &= \langle \text{LINE}, O_7 \rangle
\end{align*} \]

DISK, RTRIANGLE, LINE are graphical primitives. \(O_1, \ldots, O_7\) are sets of modification operators with associated parameters.
A graphical primitive $P'$, chosen from $P$ can be described by a four-tuple:

$$P' = <N', D, R, S, A>$$

where $N'$ is the name of the graphical primitive,
D is the definition frame,
R is the reference point,
S is the graphical datum associated with $N'$,
A is a two-tuple of attributes given by:

$$A = <C, L>$$

where $C$ denotes the colour assigned,
L denotes the level assigned.

$N'$ is an element of set $V$. $V$ is a set of reserved names with a defined syntax.

6.2 The Regional Graphics Language LIGE

The high level graphics programming language LIG has been in operation for many years. During this period, it has been found by many programmers to be a graphics language which is easy to learn and easy to use.

This language, although one of the few graphics languages that utilize the regional external representation model, should be considered as a 'weak' regional graphics language. The term 'weak' is used since the language was designed to handle line-drawing graphics, but has been refined and expanded to handle both line-drawn and solid-area graphical objects.

In view of the general acceptance of LIG, a 'weak' regional graphics language, it is used as a starting point for the proposed regional graphics programming language, hereafter called LIGE. It is an extension of LIG and is capable of dealing with graphical object such as
Because LIGE includes LIG, discussion of features available in LIG will not be presented. Rather, attention is focused on those features that are available in LIGE only. In the sequel, knowledge of LIG is assumed.

6.2.1 A Graphical Declaration Statement

The association of a name with a graphical object will be defined as a graphical variable. A graphical variable has as its value a graphical object. All graphical variables in LIGE are of the same type, GRAPHICAL. The data type GRAPHICAL represents the kind of two-dimensional information from which solid areas as well as line-drawings can be composed. A graphical declaration statement consists of the reserved word GRAPHICAL followed by a list of names or identifiers. For example:

```
* GRAPHICAL EX_1, EX_2;
* GRAPHICAL A, B, BB(10);
```

6.2.2 A Proposed Set of Graphical Primitives for LIGE

Using the regional external representation model, LIG treats all graphical objects as surfaces. There are six graphical primitives in LIG (Figure 6.2). Each graphical primitive is defined on a planar finite surface. For convenience, the surface is defined to be a unit square. Any portion (excluding the unit square) that appears black or opaque belongs to that portion of the graphical primitive that has a certain colour assigned. The ratio of the areas between the latter portion and the unit square is called the opaque value. The opaque value range from 0 (BLANK) to 1 (TILE). However, although this set of graphical primitives is not sufficient to construct a graphical object such as Figure 6.1, it is adequate for constructing a wire-frame graphical object such as
Figure 6.2 Graphical primitives of LIG

Figure 6.3 A graphical object
Figure 6.3. Thus, the set of primitives in LIGE must be extended to include primitives with higher opaque values than those in LIG. Furthermore, since graphical primitives are for the users' convenience, their choice must come from the user. A graphical primitive such as Figure 6.4a may be more useful than the one shown in Figure 6.4b for a particular user. In general, the following two rules may be found useful in choosing the set of graphical primitives. First, a convenient minimal set of graphical primitives is desired. Secondly, the graphical primitive chosen must be the smallest component possible. For instance, the graphical primitive such as Figure 6.5a is more useful than the one in Figure 6.5b. The reason is obvious: Figure 6.5b can be constructed using Figure 6.5a but not vice versa. With this in mind, a proposed set of graphical primitives for LIGE is presented.

Denote \( P \) to be the set of graphical primitives for LIGE. Then \( P \) can be described as:

\[
P = \{ \text{BLANK, LINE, SQUARE, CIRCLE, SCIRCLE, TRIANGLE, DISK, TILE, RTRIANGLE, LSECTOR, QRING} \}
\]

The first six primitives are identical to that of LIG while the last five primitives (Figure 6.6) are available only in LIGE.
Figure 6.4  Graphical primitives

Figure 6.5  Graphical primitives
Figure 6.6 Graphical primitives defined in LIGE but not in LIG
6.2.3 A Proposed Set of Graphical Operators for LIGE

In the previous section, the graphical primitives for graphical objects are discussed. In this section, the discussion is focused on how complex graphical objects can be built up or defined with the help of graphical operators.

There are two kinds of graphical operators in LIGE, the unary and the binary graphical operators.

The unary graphical operators

An unary graphical operator is one that acts on a single graphical operand only. Each operator has one or more numerical arguments. The unary operators available in LIGE are:

1) the scaling operator (SCALE),
2) the position operator (AT),
3) the rotation operator (ANGLE),
4) the line operator (LINE),
5) the naming operator (AS),
6) the colour operator (COLOUR),
7) the priority operator (LEVEL),

Note that the colour and priority operators are not available in LIG.

By default, a graphical primitive is of colour BLACK and of level 1. Level 1 is the lowest level and the next higher is level 2, etc. A graphical object of a higher level than another is said to have a higher priority in visibility than the other.

The following two examples will demonstrate how the operators may be used:
Example 1

* GRAPHICAL A;
...

* A :- TILE COLOUR 'GREEN' LEVEL 2 ;
...

A is a graphical object with colour attribute GREEN and level attribute 2.

Example 2

* GRAPHICAL A, B;
...

* B :- DISK COLOUR 'BLUE' LEVEL 3 <m₁>

* + RTRIANGLE COLOUR 'RED' LEVEL 2 <m₂> ;

* A :- B COLOUR 'GREEN' LEVEL 2 <m₃> ;
...

where <m₁>, <m₂>, <m₃> denote modifications.*

B is graphical object composed of two graphical primitives, DISK and RTRIANGLE, each with different colour and level attributes. A is a graphical object composed of two graphical primitives DISK and RTRIANGLE both having the same colour (green) and level (2) attributes.

A complement operator could also be introduced. Its effect is to interchange the colours of the graphical object and the background.

Some examples are given in Figure 6.7 to show the effects of the complement operator on graphical primitives. Note that graphical objects...

* Modification refers to the redefinition of the default position, scalings, orientation, colour and level of a graphical object using unary operators AT, SCALE, ANGLE, COLOUR, and LEVEL.
can be obtained without the use of the complement operator (Figure 6.7 (a-b)) since superposition of graphical objects with different levels and appropriate colours can achieve the same visual effect as that of the complement operator (Figure 6.7(c-d)). For these reasons, the complement operator is not included in LIGE.

Figure 6.7 The effects of the complement operator on graphical primitives
The binary graphical operators

A binary graphical operator requires two graphical operands as arguments. An argument can be a modified or unmodified graphical variable. There are two binary graphical operators in LIGE, the superposition (denoted by '+') and the deletion (denoted by '-') operators.

Example

* GRAPHICAL FRUIT, ORANGE, Q;

... 

* FRUIT :- FRUIT + ORANGE <m> AS Q ;

* FRUIT :- FRUIT - Q ;

where <m> denotes one or more modifications.

Before the LIGE language statements are presented, the importance and usefulness of the BLANK primitive deserves some discussion. There are two common usages of the BLANK primitives:

1) Initialization

The following is an example comparing a segment of an LIGE program with that of a Fortran program.

* GRAPHICAL SAMPLE, A(10); REAL SAMPLE, A(10)

... ... 

* SAMPLE :- BLANK; SAMPLE = 0.

DO 10 I = 1, 10 DO 10 I = 1, 10

* SAMPLE :- SAMPLE + A(I); SAMPLE = SAMPLE + A(I)

10 CONTINUE 10 CONTINUE

2) Identification

Since an instance of the BLANK primitive can be identified, it

**The value of a graphical variable, which has been modified, is called an instance of that graphical variable.
is useful to use the BLANK primitive for producing a menu (Figure 6.8).

6.2.4 Attribute Extraction Statements for LIGE

XLOC, YLOC, XSCALE, YSCALE, ANGLE are attribute extraction functions in LIG. A number of additional statements are available in LIGE language. These are concerned with the physical properties of graphical objects.

Often, the area of a graphical object is desired. The following LIGE statement:

```
* GRAPHICAL GR_OBJ ;
REAL AOBJ
...
AOBJ = AREA(GR_OBJ) ;
...
```

will assign the value of the area of GR_OBJ to the numerical variable AOBJ.

The sum of the areas of all the graphical objects currently displayed can be obtained by executing the following LIGE statement:
REAL AOBJ

* AOBJ = AREA_DISPLAY ; /* AOBJ is a numerical variable. */

Two functions, XCG, YCG are available for finding the x-coordinate and y-coordinate of the centre of gravity of a graphical object (always assumed to be of uniform density).

6.2.5 I/O Statements for LIGE

In LIG, the I/O statements are the DISPLAY, HARDCOPY, DRAW, ERASE, CURSOR and READ CURSOR. The I/O statements for LIGE, in addition to those of LIG, are the PAINT, BRUSH, DISPLAY LEVEL, DISPLAY COLOUR.

1) The PAINT statement

When a region on the screen requires special attention, statements of the form:

* PAINT WITH MASK <xcoor>, <ycoor>, <nvertices> ;
* PAINT WITH MASK <xcoor>, <ycoor>, <nvertices> COLOUR <string> ;

are used where <xcoor> denotes an array that contains the x coordinates of the vertices defining the mask, <ycoor> denotes an array that contains the y coordinates of the vertices defining the mask, <nvertices> denotes the number of vertices defining the mask and <string> denotes a colour. In the first example, the default colour, 'BLACK', is assumed. The region thus obtained cannot be transformed as it is not stored in the data base.

2) The BRUSH statement (Figure 6.9)

While a draw statement is used in both line and regional graphics, a brush statement is specific to regional graphics. For example, the execution of the statements

* BRUSH FROM xprev, yprev TO xcurr, ycurr ;
* BRUSH FROM xprev, yprev TO xcurr, ycurr WIDTH w ;

will have the effect of producing a line with thickness w unit, by default
Figure 6.9 An example of the effect of executing the BRUSH statement
(the first example), $w$ is 0.05. The host language variables are $x_{prev}$, $y_{prev}$, $x_{curr}$, $y_{curr}$ and $w$.

**The DISPLAY LEVEL and the DISPLAY COLOUR statements**

The DISPLAY LEVEL and the DISPLAY COLOUR statements will be useful in application such as integrated circuit mask design. For instance, all transistors of a particular kind are represented with a particular colour. Execution of the DISPLAY COLOUR statement with that colour as an input parameter will effectively display all transistors of that particular kind on the screen.

3) **The DISPLAY LEVEL statement**

Only those parts of graphical objects with the specified level or range of levels are displayed.

* DISPLAY LEVEL $L_1,L_2$; (a range of levels)
* DISPLAY LEVEL $L_1$; (one particular level)
* DISPLAY LEVEL ; (all existing levels)

$L_1, L_2$ denote level numbers.

4) **The DISPLAY COLOUR statement**

Only those parts of graphical objects with the specified colour or set of colours are displayed.

* DISPLAY COLOUR $C_1,C_2,C_3,...$; (a set of colours)
* DISPLAY COLOUR $C_1$; (one particular colour)
* DISPLAY COLOUR ; (all existing colours)

$C_1, C_2, C_3$ denote colour strings.

6.2.6 **A Graphical Function for LIGE**

A graphical function refers to a special operator which yields a result whose value is a graphical object. The result may be dependent on a set of input parameters associated with the function.
REGION is a graphical function with a set of input parameters, viz. arrays \( X \) and \( Y \) of x-coordinates and y-coordinates of the vertices defining the graphical object, the number of vertices defining the region \( (NVERT) \) and the colour specified for the graphical object. By default, colour is black.

Example

\[
\begin{align*}
\text{GRAPHICAL GF, GG ;} \\
\text{REAL } X(10), Y(10) \\
\ldots \\
\text{GF :- REGION}(X,Y,NVERT,'BLACK') \text{ SCALE } .5,.2 ; \\
\text{GG :- REGION}(X,Y,NVERT) \text{ SCALE } .5,.2 ; /* Default */
\end{align*}
\]

* Note that the value of the function REGION is a graphical object which is a simply connected region. A function which can generate a multiply connected region will require more input parameters such as the number of boundaries defining the region and the number of vertices defining each of these boundaries.
A systematic attempt has been made to unify the theory and various techniques for regional graphics. A regional external representation model facilitates a consistent description and provides a convenient means for dealing with both line-drawn and opaque graphical objects. In order to understand how this type of graphical object can be handled, a thorough study of the internal and external representations of regions has been carried out. A good internal representation scheme for a region is one that makes it easy to manipulate and efficiently evaluate the physical properties of the region. External representations of regions rely on the type of graphics display system available. A vector graphics system achieves the external representation of regions by the use of hatching patterns. A good hatching algorithm is one that is capable of producing a large set of distinctive hatching patterns conveniently and efficiently. A reasonably large set of this kind can be guaranteed if a hatching algorithm allows a user to define the thickness and inclination of the hatching lines.

The mathematical and conceptual aspects of hatching algorithms have been discussed. Through a systematic investigation, some of the fundamental concepts of hatching algorithms were established. Based on these established concepts, a hatching algorithm was then developed. It has been found to be successful in achieving the expected tasks.

Adopting the concept of treating graphical data as a data type, the regional graphics language LIGE has been proposed. Some features of the language have been implemented with all graphical variables belonging to the type GRAPHICAL. A complete list of the required features for a language will not be available at an early stage. Thus, first a proposed language will have to be tested in actual use. In order to have the
maximum feedback, users from various fields should attempt writing
different application programs. The advantages and disadvantages of the
language can then be considered, and refinements of the language incor-
porated. The improved language will then lead to further improvements.
Hence, the proposal of LIGE constitutes the first step towards the con-
struction of a high-level regional graphics programming language.
BIBLIOGRAPHY

The following abbreviations are used in the bibliography:

ACM: Association for Computing Machinery
CAD: Computer-Aided Design
CGIP: Computer Graphics and Image Processing
FJCC: Fall Joint Computer Conference
IEEE: The Institute of Electrical and Electronic Engineers
IFIP: International Federation of Information Processing Societies
JACM: Journal of Association for Computing Machinery
SJCC: Spring Joint Computer Conference
AFIPS: American Federation of Information Processing Societies
UAIDE: User of Automatic Information Display Equipment
SIGGRAPH: ACM Special Interest Group on Computer Graphics


APPENDIX A

AREA OF A SIMPLY CLOSED POLYGON

Given an arbitrary simply connected n-sided polygon defined by vertices: \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n), (x_1, y_1)\), the area of the polygon (Figure A.1) is given by:

\[
A = \frac{1}{2} \sum_{i=1}^{n} (x_i y_{i+1} - x_{i+1} y_i) \quad \text{Equation A.1}
\]

where \(k = \begin{cases} i + 1 & \text{if } i \neq n \\ 1 & \text{if } i = n \end{cases} \)

Figure A.1 A simply closed polygon

Denote \((\bar{x}, \bar{y})\) to be the coordinates of the centre of gravity of the n-sided polygon. The \(\bar{x}, \bar{y}\) can be proved (Hall [Ha76]) to be given by:

\[
\bar{x} = \frac{1}{6A} \sum_{i=1}^{n} (x_i + x_k) \, (x_i y_k - x_k y_i) \quad \text{Equation A.2}
\]

\[
\bar{y} = \frac{1}{6A} \sum_{i=1}^{n} (y_i + y_k) \, (x_i y_k - x_k y_i) \quad \text{Equation A.3}
\]

where \(A\) is given by Equation A.1.
APPENDIX B

LIGE HOMOGENEOUS TRANSFORMATION MATRICES

The homogeneous transformation matrices used in LIGE are identical to those in LIG. Using these matrices, a series of transformations: rotation, translation, and scaling in any order, can easily be carried out. A systematic investigation of the advantages of these matrices has been carried out in Chan [Ch75]. A point \( P_1 \) in matrix form is transformed to a corresponding point \( P_2 \) (also in matrix form) using the \( A \) and \( B \) matrices, i.e. \( P_2 = A . P_1 + B \) (Equation B.1), where \( . \) denotes matrix multiplication,

\[
A = \begin{bmatrix}
XSCALE \times \cos\theta & - C \times \sin\theta \\
D \times \sin\theta & YSCALE \times \cos\theta
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
XLOC + (D \times \sin\theta - XSCALE \times \cos\theta) \times .5 \\
YLOC - (C \times \sin\theta + YSCALE \times \cos\theta) \times .5
\end{bmatrix}
\]

with \( C = \begin{cases} 
YSCALE & \text{if rotation is not first in the transformation}, 
XSCALE & \text{otherwise}.
\end{cases} \)

with \( D = \begin{cases} 
XSCALE & \text{if rotation is not first in the transformation}, 
YSCALE & \text{otherwise}.
\end{cases} \)

\( XLOC = \) the horizontal axis \( x \) location

\( YLOC = \) the vertical axis \( y \) direction

\( XSCALE = \) the horizontal axis scale factor

\( YSCALE = \) the vertical axis scale factor

\( \theta = \) the rotation angle in radians.
APPENDIX C

THE CALCULATION OF THE AREA OF A GRAPHICAL OBJECT
(which is represented by the regional external representation model)

![Diagram of a triangle and its instance]

Figure C.1 A triangle and its instance

It is a matter of algebra to prove that the ratio of the areas of the triangle ABC and its associated definition frame DEFG is equal to that of the modified triangle A'B'C' and its associated definition frame D'E'F'G' (Figure C.1(a-b)). This ratio depends on the topology of the triangle ABC and is, for convenience, called the k-value. Besides, the area of the modified definition frame is always a constant and is equal to \( (S_x \times S_y) \). \( S_x \) and \( S_y \) are the scale factors of the abscissa and ordinate with respect to a reference point. This area is independent of the order of transformation (rotation, translation and scaling). The k-value can be expressed as:
Area of a triangle

\[ k = \frac{\text{Area of the triangle}}{\text{Area of the modified triangle}} \]

\[ = \frac{\text{Area of the definition frame associated with the triangle}}{\text{Area of the definition frame associated with the modified triangle}} \]

\[ = \frac{s_x \times s_y}{s_x \times s_y} \quad \text{Equation C.1} \]

The coordinates of all points in Figure C.1b, given those in Figure C.1a, are calculated using equation B.1. The areas of triangles and their associated definition frames are computed using Equation A.1.

Each graphical primitive in LIGE such as the one in Figure C.2a is approximated by a simply closed n-sided polygon (Figure C.2b). Each n-sided polygon can be triangulated into a number of triangles (Figure C.2c). Hence the result of the transformations of these triangles describing the polygon is equivalent to that of a transformation of the polygon (Figure C.2d).

Denote \( A_i, A'_i \) to be the areas of a triangle \( i \) and that of its instance respectively (Figure C.2(c-d)). Let \( k_i \) be the \( k \)-value of the triangle.

Using equation C.1:

\[ \frac{A_i}{1} = \frac{A'_i}{s_x \times s_y} = k_i \]

\[ \quad \text{..} \quad \]

\[ \frac{A_n}{1} = \frac{A'_n}{s_x \times s_y} = k_n \]
Figure C.2 A triangulated polygon and its instance
Hence,
\[
\frac{A_1 + A_2 + \ldots + A_n}{1} = \frac{A_1' + A_2' + \ldots + A_n'}{S_x \ast S_y} = k_1 + k_2 + \ldots + k_n
\]
or
\[
\frac{\text{Area of the } n\text{-sided polygon}}{1} = \frac{\text{Area of its instance}}{S_x \ast S_y} = k
\]
where \(k\) is a constant which is related to the topology of the polygon.

Area of a manipulated polygon = \(S_x \ast S_y \ast k \ldots \text{Equation C.2}\)

**Example**

Given a graphical primitive DISK and an instance from manipulation with scale factors \(S_x, S_y\) and angle of rotation \(\alpha\) radians. Then the area of the instance of DISK (using Equation C.2) is equal to

\[S_x \ast S_y \ast k_{\text{DISK}} = S_x \ast S_y \ast 0.78539.\]

In order to demonstrate the usefulness of Equation C.2, an example is given.

**Example**

Consider the graphical assignment statement (Figure 6.1):

* GIRL :- DISK \(<m_6>\) + LINE \(<m_1>\) + LINE \(<m_2>\)

* + LINE \(<m_3>\) + RTRIANGLE \(<m_7>\) + LINE \(<m_4>\)

* + LINE \(<m_5>\);

where \(<m_1>, <m_2>, \ldots, <m_7>\) denote modifications.

The area of the graphical object GIRL

= Area of [DISK \(<m_6>\)] + Area of [RTRIANGLE \(<m_7>\)]

+ Sum of areas of all instances of LINE

= \(k_{\text{DISK}} \ast (S_x \ast S_y)\)\(<m_6>\) + \(k_{\text{RTRIANGLE}} \ast (S_x \ast S_y)\)\(<m_7>\)

+ \[\sum_{i=1}^{5} \text{k}_{\text{LINE}} \ast (S_x \ast S_y)\] \(<m_i>\)

* Refer to Table C.1
Notation: \( (S_x \cdot S_y) \langle m_i \rangle \) denotes the \( S_x \) and \( S_y \) associated with the modifications \( \langle m_i \rangle \).

Hence, the calculation of the area of a graphical object is reduced to the summation of products. Each product is given by \( k \cdot S_x \cdot S_y \), where \( k \) is a constant associated with the graphical primitive used, \( S_x \), and \( S_y \) are the scale factors associated with the modified graphical primitive.

<table>
<thead>
<tr>
<th>Graphical Primitive</th>
<th>k-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLANK</td>
<td>0.</td>
</tr>
<tr>
<td>LINE</td>
<td>0.</td>
</tr>
<tr>
<td>TRIANGLE</td>
<td>0.</td>
</tr>
<tr>
<td>SQUARE</td>
<td>0.</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>0.</td>
</tr>
<tr>
<td>SCIRCLE</td>
<td>0.</td>
</tr>
<tr>
<td>TILE</td>
<td>1.</td>
</tr>
<tr>
<td>DISK</td>
<td>0.78539</td>
</tr>
<tr>
<td>RTRIANGLE</td>
<td>0.5</td>
</tr>
<tr>
<td>QRING</td>
<td>0.14726</td>
</tr>
<tr>
<td>LSECTOR</td>
<td>0.19634</td>
</tr>
</tbody>
</table>

Table C.1: Graphical primitives and its associated k-values
APPENDIX D

THE CONSTRUCTION OF THE LIGE PREPROCESSOR
(Figure D.1)

Some features of LIGE have been implemented using the compiler writing system XPL. With the aid of the XPL, the construction of the LIGE preprocessor is simplified. More important, use of XPL increases flexibility during the experimental design of a language because the syntax and the associated semantics can be modified independently of one another.

Using the XPL system the language extension LIGE is being treated as an independent and separate language. The Backus-Naur-Form (BNF) notation is used to define the language syntax for LIGE. The syntax is then checked for correctness by a component of the XPL system, the ANALYZER. Once the syntax is error free, a table which is essentially a set of declarations, is produced. The table is in a format that is ready to be inserted into a compiler framework, the modified SKELETON. There are two slots in the modified SKELETON; one for the table mentioned above and the others for the LIGE syntax-semantics. The semantics are written in XPL and the procedure is called SYNTHESIZE. Whenever a grammar rule is applied by the parser for reducing the input string, the procedure SYNTHESIZE is called upon by the SKELETON. The SKELETON with these components represents the LIGE preprocessor in source code form. The LIGE preprocessor is compiled with the XPL compiler.

For the preprocessor development at UBC, the EXPL (Extended XPL) compiler [Ba75], a slightly modified version of the XPL compiler, is used. It is superior to the XPL compiler in that it permits separate compilation of procedure modules, thus allowing more efficient debugging.
Figure D.1 The construction of the LIGE preprocessor
APPENDIX E

AUTHORS SORTED BY SUBJECTS

Application
[Ad76a] [Ad76b] [Al75]
[Er70] [Fe78] [Fr70]
[Fr75] [Ge72] [Le61]
[Lo78] [Na78] [Ph76]

Contour Representation
[Bu77] [Fr61a] [Fr61b]
[Fr62] [Fr74] [Lo65]
[Mo68a] [Mo68b] [Mo69]
[Ph71] [Za69]

Hatching and/or Filling
[Bh78] [Is73] [Li78]
[Pa78a] [Pa78b] [Pe78]
[Ph72] [Ph76] [Pf67]
[Ve79]

Hidden Surface and/or Line
[Fr67] [Su73] [Su74b]
[Wi77]

Memory Consideration
[Ch77] [Ja75b] [Ta77]

Regional Graphics Language
[Br71] [Ch75] [Na71]
[Ne75a] [Ne78] [Sc76a]
[Sc76b]
### Region Representation

<table>
<thead>
<tr>
<th>[Cl70]</th>
<th>[Co67]</th>
<th>[Du76]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Me73]</td>
<td>[Oc76]</td>
<td>[Pa77]</td>
</tr>
<tr>
<td>[Pf67]</td>
<td>[Ro73]</td>
<td>[Sa78]</td>
</tr>
<tr>
<td>[St70]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Text

<table>
<thead>
<tr>
<th>[Ch78]</th>
<th>[Gi78]</th>
<th>[Ne78]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Ro76a]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Windowing Technique

<table>
<thead>
<tr>
<th>[Be69]</th>
<th>[Dw67]</th>
<th>[Ja75a]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Ph76]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>