ICEMENDR: Intelligent Capture Environment for Mechanical Engineering Drawing

by

Manoj D. Muzumdar

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1999
[June 1999]

© Manoj D. Muzumdar, MCMXCIX. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

ARCHIVES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 15 1999
LIBRARIES

Author .....
Department of Electrical Engineering and Computer Science
May 21, 1999

Certified by .....
Randall Davis
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

ICEMENDR: Intelligent Capture Environment for Mechanical Engineering Drawing

by

Manoj D. Muzumdar

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 1999, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

I designed and implemented an intelligent environment for recognizing simple mechanical engineering sketches. This involves the analysis of complex mechanical engineering parts and their components and consists of creating a hierarchical recognition system capable of parsing these parts with simpler geometric primitives. The system seeks to provide an intuitive pencil-and-paper-like interface for sketch recognition by allowing incremental recognition of what a user draws on the system. The system's knowledge is arranged in simple Recognizer modules that have very specialized information on a particular aspect of recognizing a part.

Thesis Supervisor: Randall Davis
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my thesis advisor, Randall Davis, as well as Luke Weisman, and Christine Alvarado for their suggestions, advice, and discussion time. I would also like to thank MIT AI Intelligent Room for the use of its computer facilities, and Ford Motor Company for its generous funding. A special thank you goes to Chi Un Kim for last minute proofreading help, and of course, I would like to thank my parents Deepak and Vijaya Muzumdar for their constant words of encouragement.

Contents

1 Introduction **9**

1.1 Motivation 9

1.2 General Approach 10

1.3 Success Criteria 14

1.4 Impact 15

1.5 Structure of this thesis 15

2 Previous Work **16**

2.1 Electronic Cocktail Napkin 16

2.2 GRANDMA 17

2.3 Drawing Analogies 18

2.4 Summary 18

3 Interface, Architecture, and Implementation **20**

3.1 Interface 20

3.2 Architecture and Implementation 21

3.2.1 rec.ui 22

3.2.2 rec.util 22

3.2.3 rec.geo 22

3.2.4 rec.sys 22

3.2.5 rec.core 25

3.2.6 rec.meche 29

3.3 Summary 41

4 Conclusion **42**

4.1 Future Work 42

4.2 Summary 44

Bibliography **45**

List of Figures

1-1 Sketch of a rivet setting tool 11
1-2 Alternative sketch of a rivet setting tool 12
1-3 A cam 12
1-4 A compression spring 13
1-5 A fixed pivot 13
1-6 A force 13
1-7 A frame 13
1-8 A screw 14
1-9 A washer 14

3-1 Rectangle recognition 26
3-2 RectangleRecognizer 28
3-3 Ball and socket recognition 30
3-4 Bar recognition 30
3-5 Belt recognition 31
3-6 Cam recognition 32
3-7 Chain recognition 33
3-8 Compression Spring recognition 33
3-9 Fixed Pivot recognition 34
3-10 Force recognition 34
3-11 Frame recognition 34
3-12 Gear recognition 35
3-13 Mass recognition 35

3-14 Motor recognition 36
3-15 Nut recognition 37
3-16 Piston recognition 37
3-17 Screw recognition 38
3-18 Solenoid recognition 38
3-19 Tension Spring recognition 39
3-20 Washer recognition 40

List of Tables

3.1 Important GShape methods 23
3.2 Important Recognizer methods 23
3.3 Important Widget methods 24
3.4 Important VisibleWidget methods 24
3.5 Important WPart methods 29

Chapter 1

Introduction

Engineers need an intelligent environment for developing their ideas. This thesis focuses on the design, implementation, and evaluation of ICEMENDR (Intelligent Capture Environment for Mechanical ENgineering DRawing), an intelligent environment for recognizing simple mechanical engineering design sketches.

1.1 Motivation

An intelligent environment for engineering should facilitate design development and design rationale capture. I envision an environment where engineers can sketch their ideas, and have a computer watch them and give them feedback. For engineers to find the environment desirable, it has to have an inobtrusive interface. Thus, it seems best to bring the computer to the engineer. As mechanical engineers are accustomed to sketching their ideas out with pencil and paper, a practical environment for engineering design should include a pencil-and-paper-like interface.

An environment consistent with this idea would be one in which the computer "watches" the engineer draw and communicates with him about what is happening. By speaking to the computer, the engineer could tell the computer what he is drawing, answer any questions, and correct mistakes the computer made. The computer could communicate to the engineer through speech synthesis. The engineer could continue talking to the computer without interrupting his work, his eyes on his paper and his

hand on his pen.

1.2 General Approach

ICEMENDR takes the first step toward creating an intelligent environment for mechanical engineering design. ICEMENDR is a Java program capable of recognizing a subset of mechanical engineering design sketches. The user uses a digital tablet and stylus to interact with ICEMENDR. Error correction is done through GUI interaction with buttons for various operations such as undo and redo.

ICEMENDR allows an engineer to quickly create simple design sketches. Mechanical engineering sketches are composed of simple parts which themselves are composed of geometric primitives. ICEMENDR is a scalable system that features different levels of recognition. As the engineer draws, ICEMENDR actively tries to recognize what he is trying to draw. Conceptually, the drawing paper is a Surface. The objects drawn on the Surface are called Widgets. The system's recognition knowledge is stored in Recognizers. Each Recognizer knows how to convert a set of lower-level Widgets to higher-level Widgets. It removes lower-level Widgets and adds higher-level Widgets to the Surface.

For example, a triangle Recognizer takes three line Widgets in a triangle configuration and replaces them with a single triangle Widget. The Recognizer encapsulates the knowledge to accomplish this task. Each Recognizer is interested in a set of Widgets on the Surface that relate to one another in a certain way; when it finds them it replaces them with other Widgets. These new Widgets, along with other Widgets already on the Surface, then trigger other Recognizers to act. By using this technique, ICEMENDR allows fairly complex parts to be recognized by applying this hierarchical recognition on each of the user's strokes. For example, figures 1-1 and 1-2 show two separate designs of a rivet setting tool drawn using ICEMENDR. Figures 1-3 through 1-9 represent the parts used in the rivet sketches. In addition, all non-part polygons are masses and all non-part lines are bars.

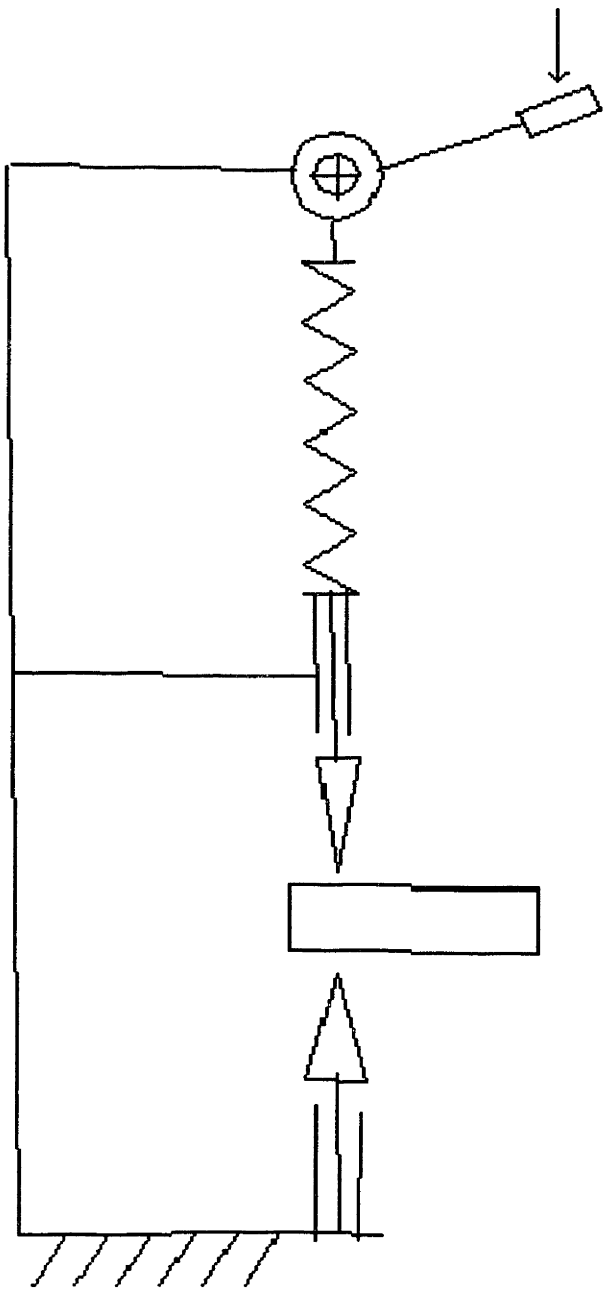

Figure 1-1: Sketch of a rivet setting tool

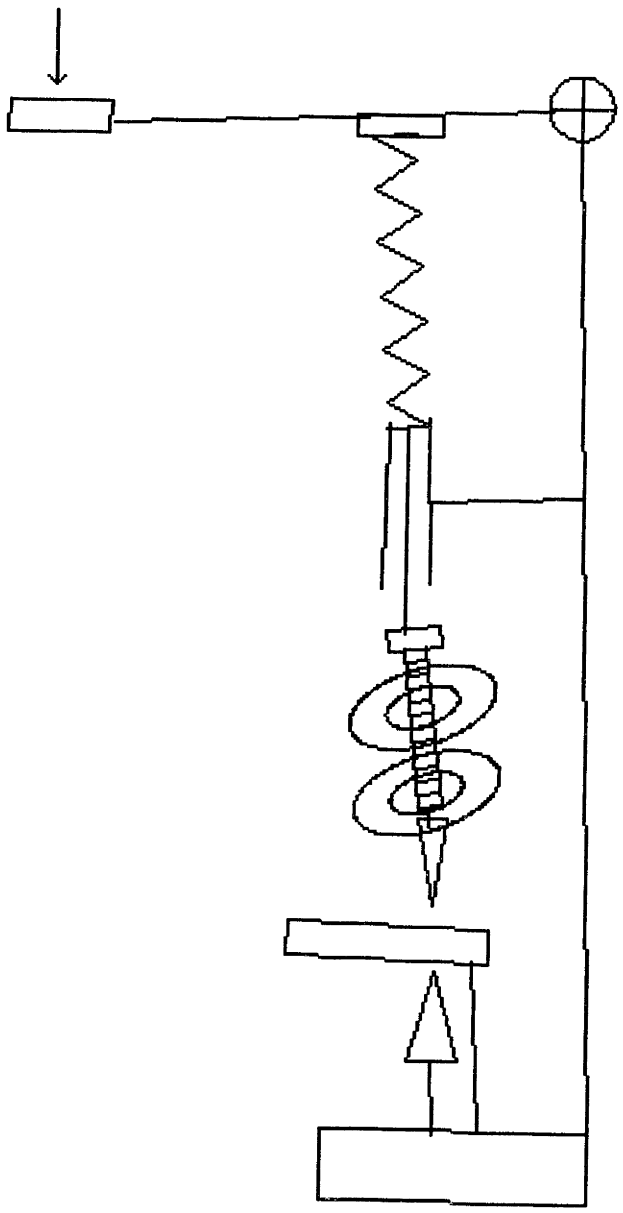

Figure 1-2: Alternative sketch of a rivet setting tool

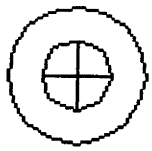

Figure 1-3: A cam

Figure 1-4: A compression spring

Figure 1-5: A fixed pivot

Figure 1-6: A force

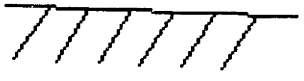

Figure 1-7: A frame

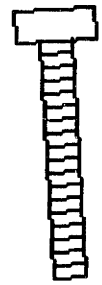

Figure 1-8: A screw

Figure 1-9: A washer

1.3 Success Criteria

ICEMENDR recognizes eighteen primitive mechanical engineering parts:

1. ball and socket
2. bar
3. belt
4. cam
5. chain
6. compression spring
7. fixed pivot
8. force
9. frame
10. gear
11. mass
12. motor
13. nut
14. piston
15. screw
16. solenoid
17. tension spring
18. washer

ICEMENDR also can be easily integrated with speech to be used with the MIT AI Laboratory's Intelligent Room and its underlying system Metaglue. The environment accomplishes part recognition by using a hierarchy of Recognizers to parse the mechanical engineering parts into simpler geometric primitives.

1.4 Impact

ICEMENDR's impact will come when it is integrated with other interfaces to create an intelligent drawing environment for mechanical engineering drawing. There has been extensive research on drawing recognition, voice interaction, and computer-aided design. There has also been some research in applying knowledge systems to drawing. However, these separate branches have not been integrated. As a result of this lack of integration, all products from these research endeavors have failed to produce a product that is easily accessible and useful. By creating ICEMENDR, I have taken one of the first steps toward creating a system where a user can work in an environment that is both familiar and helpful in the design process. Such an environment would be able to aid and record pertinent information generated both knowingly and unknowingly by the engineers who use it.

1.5 Structure of this thesis

Chapter 2 contains a summary of work that is related to this thesis. Chapter 3 describes the the interface and architecture of ICEMENDR, and presents the eighteen simple mechanical engineering parts. Finally, Chapter 4 discusses future work involving ICEMENDR and presents some conclusions that can be drawn from the results of this thesis.

Chapter 2

Previous Work

Three major projects lay the foundation for ICEMENDR. They are the Electronic Cocktail Napkin project, GRANDMA, and "Drawing Analogies." In this section, I will briefly discuss each project and which features relate to ICEMENDR.

2.1 Electronic Cocktail Napkin

The Electronic Cocktail Napkin (ECN) project has made much progress towards goals common to their work and mine. It is a computer-based environment for sketching and diagramming during conceptual design. The computer extracts knowledge about raw sketches and uses database queries to gather design images for architectural planning. ECN centers around methods for extracting knowledge from a sketch; it analyzes sketches based on bounding boxes, spatial relations, and diagram-based query and retrieval.

ECN uses a three-step recognition process. It recognizes pen-drawn "glyphs" on a table[1]. Next, it uses spatial relations among diagram elements to analyze the "glyphs." Finally, it searches across previously defined configurations to match elements on the diagram to the "glyphs." This final step is parsing the diagram with "glyphs."

ICEMENDR uses ECN's idea of parsing, but ICEMENDR's implementation dif-

[1] A glyph is ECN's terminology for a simple multi-stroke symbol.

fers in several important ways. First of all, ICEMENDR does not try to match multi-stroke symbols. It deals with one stroke at a time. ECN has two separate databases on which it searches: the one for the low-level "glyphs" and the one for the higher-level diagrams. ICEMENDR has no distinction between how lower- and higher-level recognition are done. All of this knowledge is put into a Recognizer. Thus, when another part or diagram needs to be added to ICEMENDR, another Recognizer is created. With ECN, the databases must be altered and the recognition of new parts must be done in the "glyph" and diagram databases. Hence, ICEMENDR differs from ECN in knowledge arrangement.

Both ICEMENDR and ECN use a paper-and-pen interface implemented with a tablet and stylus. Overall, the Electronic Cocktail Napkin project centers around knowledge extraction and not the interface [5].

2.2 GRANDMA

Dean Rubine implemented GRANDMA (Gesture Recognizers Automated in a Novel Direct Manipulation Architecture), a toolkit for reducing the effort involved in creating a gesture-based interface. GRANDMA creates a recognition interface from a set of examples of how to draw the gestures to be recognized. For example, a circle would be drawn five times into the system, and then GRANDMA, would construct a circle recognizer for the system. GRANDMA uses a mouse-driven interface to train the recognizer as well as for input. GRANDMA can create recognizers only for single-stroke gestures.

ICEMENDR differs from GRANDMA in both how recognition is done and in what interface is used. ICEMENDR does not train its Recognizers. They are hand-crafted for a particular interface. Although this does not allow for the automation provided by GRANDMA, it allows for recognition of multi-stroke gestures and gives future Recognizer creators leeway to create a particular Recognizer in the way best for its particular application field. Also, ICEMENDR's paper-and-pen interface is more natural than using a mouse [8].

2.3 Drawing Analogies

Ellen Yi-Luen Do and Mark Gross created "Drawing Analogies," a shape-based reminder program that uses freehand sketches to index and retrieve visual references for creative designing [2]. They present four recognition matching schemes: element count match, element type match, relations match, and element "type&relations" match. Element count match states that two drawings are identical if they have exactly the same number of elements. The element type match says that two drawings are identical if they have exactly the same element types regardless of the number of elements. The relations match states that two drawings are identical if they contain exactly the same spatial relations regardless of the elements involved. The element "type&relations" match says two drawings are identical if they are identical in the element count match, the element type match, and the relations match. The element "type&relations" match is important because it combines the previous matching schemes to analyze sketches as being composed of smaller drawing primitives in a particular spatial relationship. Each of ICEMENDR's Recognizers acts as an element "type&relations" matcher [1].

Mark Gross has argued for the importance of interacting with the computer through sketching. He believes that such an interface would free the creative minds of users and not limit them. He also maintains that the environment should be aesthetically appealing and easy to use. Gross's work provides defense and motivation for my work [4].

2.4 Summary

Previous work has produced a several mechanisms for recognizing, but they have not made the jump to an integrated environment for designing. ICEMENDR is one of the first steps in creating such an environment. It takes the some of the better recognition approaches from ECN, GRANDMA, and Drawing Analogies and combines them with scalability through the use of recognition units, Recognizers.

ICEMENDR provides a sketching interface which will easily integrate into a voice-enabled mechanical engineering design environment.

Chapter 3

Interface, Architecture, and Implementation

This chapter describes ICEMENDR's interface, architecture, and implementation. It discusses the foundation on which ICEMENDR's mechanical engineering recognition capabilities are built.

3.1 Interface

ICEMENDR is a Java application which presents one window when executed. The window has two parts: a large drawing area and a smaller panel on bottom of which contains a row of buttons. There are buttons for undoing and redoing operations as well as two buttons for indicating when the user is done drawing a portion of his sketch. The first button indicates when the user is done with an ambiguous part. When this button is pressed, the default mechanical part that matches the ambiguous part is drawn in place of the ambiguous part. The second button indicates when the user is done with the entire sketch. When this second button is pressed, the system replaces all ambiguous parts with their default parts, and indicates to the user via pop-up dialog boxes whether there is anything more to be recognized.

The user interacts with ICEMENDR using a digital tablet and stylus. The stylus is tracked across the tablet like a mouse. Pressing the buttons on the interface is

accomplished by tapping the button with the stylus. Drawing is accomplished by drawing on the tablet. The system recognizes each user stroke created by a pen down and then pen up. After each such stroke, ICEMENDR tries to recognize and, if it can, replace Widgets on the Surface until there is nothing left to recognize. When a Widget is recognized, its color changes on the Surface. Basic strokes and polygons are colored black. Ambiguous mechanical engineering parts are colored cyan, and recognized mechanical engineering parts are colored blue. The drawing is complete when all Widgets on the Surface are blue, because this indicates that all the Widgets on the Surface correspond to some mechanical engineering part.

ICEMENDR reads an initialization file that specifies the Recognizers to be loaded at start up. Each of these Recognizers is grouped into a Module. There are Modules for basic strokes, polygons, selectors, and mechanical engineering parts. Basic strokes consist of lines, arcs, and circles. Lines are used in polygon recognition. Basic stroke and polygon Recognizers are used in mechanical engineering part recognition. The selector Recognizers are used to select, move, delete, copy, and rotate Widgets on the Surface. Clicking on a Widget selects it. Once a Widget is selected it can be moved, deleted, copied, or rotated, and it can be deselected by clicking on the Surface away from the selected Widget. To move a selected Widget, the user draws a line from the Widget to the point to where the Widget should be moved. To delete a selected Widget, the user clicks on the Widget. To copy a selected Widget, the user draws a circle around the center point for the new Widget. Finally, to rotate a Widget, the user draws an arc around the Widget to indicate how many degrees to rotate the Widget and in which direction. A selected Widget is colored red. When a selected Widget is deselected, it returns to its original color.

3.2 Architecture and Implementation

ICEMENDR is object-oriented. It is organized into six Java packages[1] as follows:

[1] A package in Java is a library of related classes.

- rec.ui
- rec.util
- rec.geo
- rec.sys
- rec.core
- rec.meche

In this section, I will briefly describe each of the packages and their constituents. I will then explain how they fit together.

3.2.1 rec.ui

Rec.ui consists of the actual ICEMENDR program as well as classes associated with loading in initialization files. It interfaces with rec.util and rec.sys to set up the ICEMENDR program and lets these packages handle the actual recognition.

3.2.2 rec.util

Rec.util includes all of ICEMENDR's shared utility classes. It contains classes for data structures, sorting, and file parsing. It also contains the Displayable interface that all Widgets are required to implement so that the system can draw them properly on the Surface.

3.2.3 rec.geo

Rec.geo is comprised of the basic geometry classes used throughout ICEMENDR. It contains the shapes that all the Widgets are composed of. It also has a class called Geometry with useful methods for calculating distances, angles, bisectors, and other geometric concepts. The shapes that compose the Widgets all derive from the base class GShape, ensuring that all Widgets have the properties of GShapes. Table 3.1 describes GShape's important methods.

3.2.4 rec.sys

Rec.sys contains the main classes that form ICEMENDR. The basic classes are Module, Recognizer, Squiggle, Surface, Transition, Widget, and VisibleWidget.

Method	Description
copy	Returns a copy of the GShape.
reflect	Reflects the GShape by swapping the x and y axis.
translate	Moves the GShape a relative distance.
rotate	Rotates the GShape a relative angle.
size	Returns the relative size of the GShape.
getBounds	Returns the rectangular bounding box of the GShape.
includes	Returns whether a point is within a certain distance of the GShape.

Table 3.1: Important GShape methods

Method	Description
recognize	Is called by Surface when the Surface gets a Widget that interests the Recognizer.
getInputTypes	Returns the Widgets that interests the Recognizer.
getOutputTypes	Returns the Widgets that the Recognizer creates.

Table 3.2: Important Recognizer methods

Surface represents the paper that is drawn on. It keeps track of all Widgets on itself as well as Recognizers associated with it and changes that have been made to it via Transition objects. Recognizers are grouped into Modules as described in section 3.1. They operate on Widgets that they are interested in by replacing them with Widgets that they create. Table 3.2 shows the major Recognizer methods. A Widget represents a object on the surface. Table 3.3 displays the important Widget methods. A VisibleWidget is a visible object on the Surface. An example of a VisibleWidget is a line, and an example of an invisible widget is a click. The simplest VisibleWidget is the Squiggle which is the immediate result of a user's drawn stroke. Table 3.4 illustrates the major methods that a visible widget has in addition to the methods derived from Widget.

Conceptually, the user draws on the Surface. This creates a Squiggle. The Surface then cycles through each Module until a Module says that it recognizes the Squiggle. A Module does this by cycling through its Recognizers one at a time and when one

Method	Description
copy	Returns a copy of the Widget.
move	Moves the Widget to an absolute point.
rotate	Rotates the Widget a relative angle.
size	Returns the relative size of the Widget.
getLocation	Returns location information about the Widget.
getBounds	Returns the rectangular bounding box of the Widget.
removed	Is called by Surface when the Widget is removed off of the Surface.
replaced	Is called by Surface when the Widget is replaced by another Widget on the Surface.
altered	Is called by Surface when the Widget has some property altered.

Table 3.3: Important Widget methods

Method	Description
draw	Is called by Surface to draw the Widget.
includes	Returns whether a point is within a certain distance of the Widget.

Table 3.4: Important VisibleWidget methods

of its Recognizers recognizes the Squiggle it signals the Surface. The Recognizer then removes the Squiggle from the Surface, adds the Widgets it creates to the Surface, and gives the Surface a Transition that indicates the changes it has made to the Surface. When a Widget is added to the Surface, it is actually put on a queue that is reprocessed by the Surface's Modules until the queue is empty. Also, as soon as the Surface is signaled to remove, replace, or alter a Widget, it calls the appropriate callback method on the Widget in question. Widget's callback methods corresponding to removing, replacing, and altering are called removed, replaced, and altered, respectively.

3.2.5 rec.core

Rec.core contains the basic Recognizers and Widgets that are built on rec.sys. It includes Recognizers and Widgets for selection, rotation, moving, copying, and deleting as well as Recognizers for creating the primitive Widgets that are used by rec.meche to recognize mechanical engineering parts. The basic Recognizers and the Widgets they create are:

- ArcRecognizer creates WArc.
- LineRecognizer creates WLine.
- OvalRecognizer creates WOval.
- TriangleRecognizer creates WTriangle.
- QuadrilateralRecognizer creates WQuadrilateral.
- PentagonRecognizer creates WPentagon.
- HexagonRecognizer creates WHexagon.
- PolygonRecognizer creates WPolygon.
- EquilateralTriangleRecognizer creates WEquilateralTriangle.
- RectangleRecognizer creates WRectangle.
- SquareRecognizer creates WSquare.

WArcs and WLines are created from Squiggles. A WOval is formed by two WArcs. The polygons are formed from the appropriate number of WLines. The PolygonRecognizer recognizes polygons with greater than six sides. WEquilateralTriangle, WRectangle, and WSquare are formed from their associated polygons. Also all polygons are derived from WPolygon. For example, a WRectangle would be created by the following sequence of events:

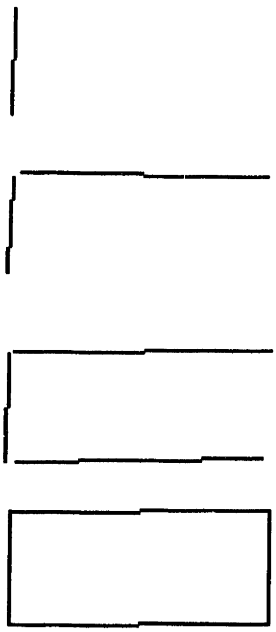

Figure 3-1: Rectangle recognition

1. Squiggle is drawn by the user on the Surface.
2. LineRecognizer converts a Squiggle that "approximates" a line to a WLine.
3. Squiggle is drawn by the user on the Surface.
4. LineRecognizer converts a Squiggle that "approximates" a line to a WLine.
5. Squiggle is drawn by the user on the Surface.
6. LineRecognizer converts a Squiggle that "approximates" a line to a WLine.
7. Squiggle is drawn by the user on the Surface.
8. LineRecognizer converts a Squiggle that "approximates" a line to a WLine.
9. QuadrilateralRecognizer converts four WLines in "approximate" quadrilateral formation into a WQuadrilateral.
10. RectangleRecognizer converts WQuadrilateral which "approximates" a rectangle into a WRectangle.

Figure 3-1 shows a rectangle being drawn from top to bottom. "Approximate" is where the knowledge lies. Each Recognizer knows how to calculate the appropriate "approximate" for whichever Widgets it recognizes. To illustrate this better and to understand how one would go about adding additional Recognizers to the system let us analyze the code for the RectangleRecognizer (see figure 3-2).

The RectangleRecognizer imports several packages so it has access to the classes in them. Vector is located in package java.util. Surface, Widget, and Recognizer are located in package rec.sys. The RectangleRecognizer derives from class Recognizer just like all Recognizers[2]. The RectangleRecognizer is interested in WQuadrilaterals as indicated in the getInputTypes() method and creates WRectangles as indicated in the getOutputTypes() method. RectangleRecognizer's work occurs in the recognize() method. Here the Recognizer checks to see if the WQuadrilateral has equal angles by using the WQuadrilateral's hasEqualAngles() method. This method returns true if the angles are within ten degrees of ninety degrees. The RectangleRecognizer then replaces a WQuadrilateral with a WRectangle if the WQuadrilateral's angles are approximately equal.

The rec.core, rec.sys, rec.geo, and rec.util packages form the basis for the rec.meche package. The next chapter will discuss the rec.meche package, the heart of ICE-MENDR's mechanical engineering part recognition capabilities.

[2]Extending a class is how Java derives classes.

```
package rec.core;

import java.util.*;
import rec.sys.*;

public class RectangleRecognizer extends Recognizer
{
  public RectangleRecognizer()
  {
  }

  public boolean recognize( Surface s, Widget w )
  {
    WQuadrilateral q = (WQuadrilateral)w;

    if( q.hasEqualAngles() ) {
      WRectangle r = new WRectangle( q );
      s.replaceWidget( q, r );
      return( true );
    }

    return( false );
  }

  public Enumeration getInputTypes()
  {
    Vector v = new Vector();
    v.addElement( "rec.core.WQuadrilateral" );
    return( v.elements() );
  }

  public Enumeration getOutputTypes()
  {
    Vector v = new Vector();
    v.addElement( "rec.core.WRectangle" );
    return( v.elements() );
  }
}
```

Figure 3-2: RectangleRecognizer

Method	Description
create	Overriden by subclasses to return the array of GShapes that make up the WPart.
getComponents	Returns the array of GShapes that make up the WPart.
getCenter	Returns the center point of the WPart.
getAngle	Returns the orientation angle of the WPart in radians.
size	Returns the relative size of the WPart.
move	Moves the WPart to an absolute point.
rotate	Rotates the WPart a relative angle.
getLocation	Returns location information about the WPart.
getBounds	Returns the rectangular bounding box of the WPart.
draw	Is called by Surface to draw the WPart.
includes	Returns whether a point is within a certain distance of the WPart.

Table 3.5: Important WPart methods

3.2.6 rec.meche

The rec.meche package consists of the Recognizers, Widgets, and utility classes used to create the mechanical engineering part recognizers. This subsection will present all the mechanical engineering parts and analyze how they are recognized.

All the mechanical engineering part Widgets are derived from the rec.meche utility class WPart. A WPart is derived from VisibleWidget. Table 3.5 describes WPart's major methods. Creating a new mechanical engineering Widget is accomplished by deriving from WPart and overriding the create method to return an array of GShapes that make up the WPart. For example, a screw Widget consists of an array of WLines which correspond to the screws sides and its threads.

A ball and socket is represented by a WMBallAndSocket Widget (See figure 3-3.). A WMBallAndSocket is formed by two intermediate parts, the WMSocket Widget and the WMBall Widget. A WLine and a WArc form a WMSocket and a WLine and a WCircle form a WMBall. If the ends of the WMSocket and WMBall Widgets are close to one another a WMBallAndSocket Widget is formed.

A bar is represented by a WMBar Widget (See figure 3-4.). A WMBar is formed by a WLine. Once the user presses a button on ICEMENDR's interface to indicate

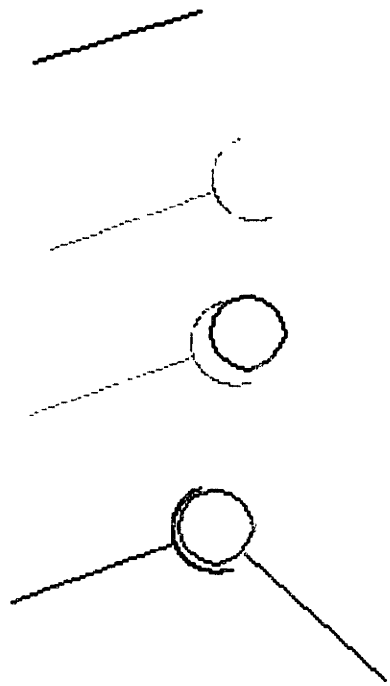

Figure 3-3: Ball and socket recognition

Figure 3-4: Bar recognition

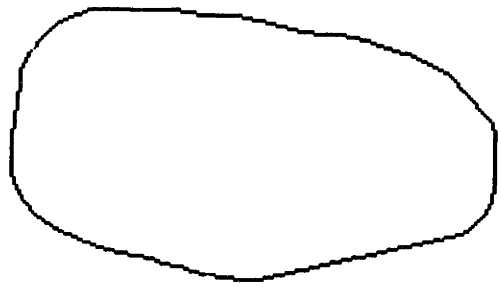

Figure 3-5: Belt recognition

that the Surface should be cleaned up, all WLines are replaced with WMBars.

A belt is represented by a WMBelt Widget (See figure 3-5.). A WMBelt is formed by a Squiggle that is closed and does not interesect itself.

A cam is represented by a WMCam Widget (See figure 3-6.). A WMCam is formed by two concentric WCircles with two WLines crossed approximately perpendicularly in the inner WCircle.

A chain is represented by a WMChain Widget (See figure 3-7.). A WMChain is formed by a WBelt with two WCircles anywhere on the WBelt.

A compression spring is represented by a WMCompressionSpring Widget (See figure 3-8.). A WMCompressionSpring is formed by a WMSpring with two WLines connected perpendicularly to the WMSpring, one at each end. A WMSpring is an intermediate mechanical engineering part that is formed by a Squiggle that is wide and pleated from one end to the other.

A fixed pivot is represented by a WMFixedPivot Widget (See figure 3-9.). A WMFixedPivot is formed by a small WCircle. Once the user presses a button on ICEMENDR's interface to indicate that the Surface should be cleaned up, all WCircles with small radii are replaced with WMFixedPivots.

A force is represented by a WMForce Widget (See figure 3-10.). A WMForce is formed by three WLines. The two "arrowhead" WLines are approximately at a forty-five degree angle from the "base" WLine.

A frame is represented by a WMFrame Widget (See figure 3-11.). A WMFrame is formed by five WLines. The four WLines coming off of the "base" WLine are

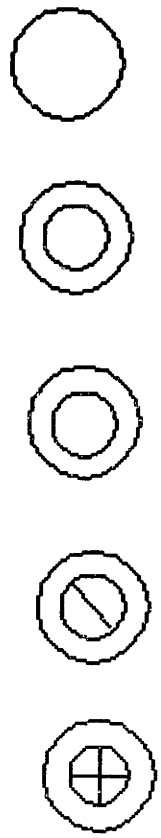

Figure 3-6: Cam recognition

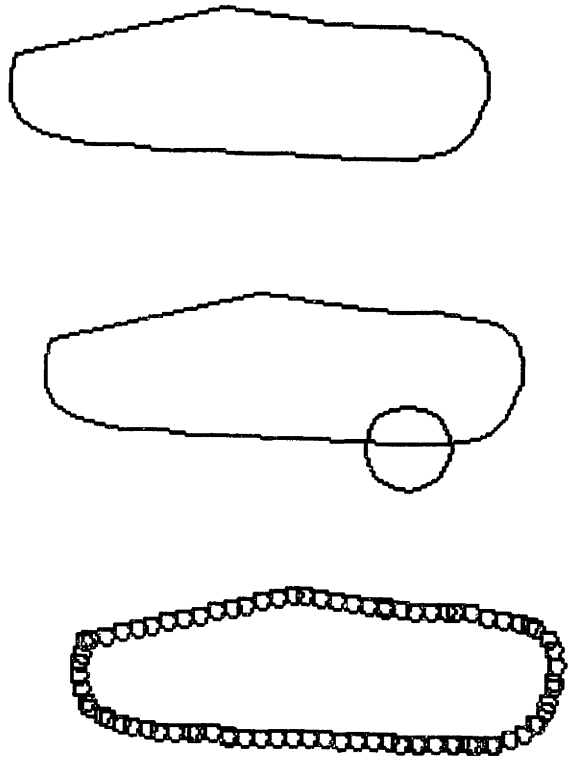

Figure 3-7: Chain recognition

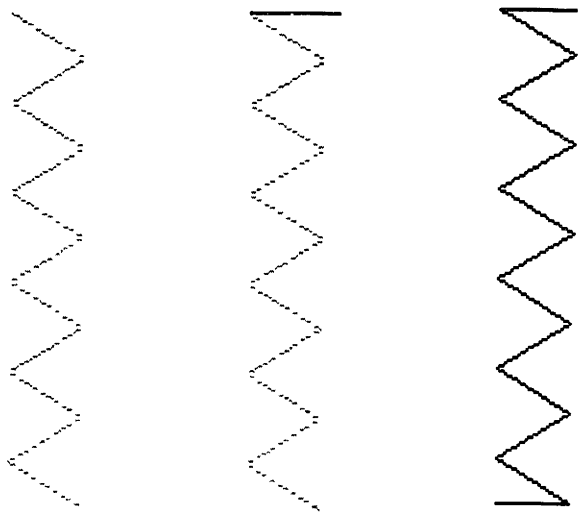

Figure 3-8: Compression Spring recognition

Figure 3-9: Fixed Pivot recognition

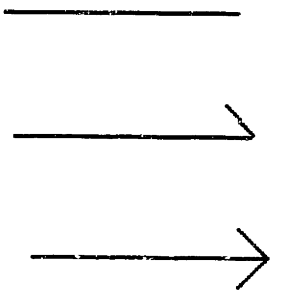

Figure 3-10: Force recognition

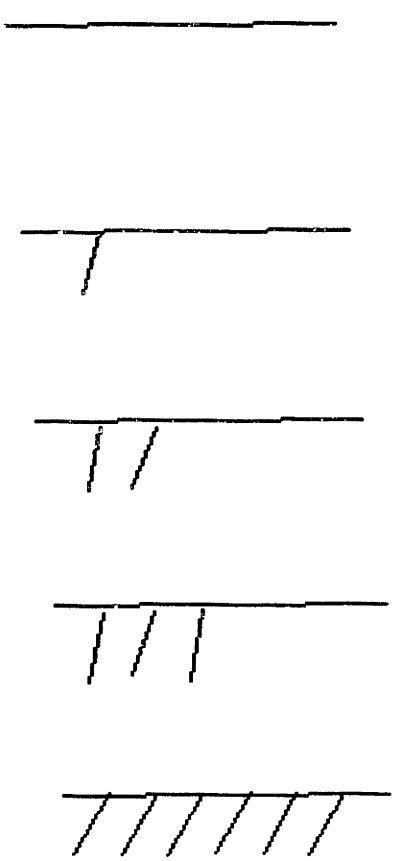

Figure 3-11: Frame recognition

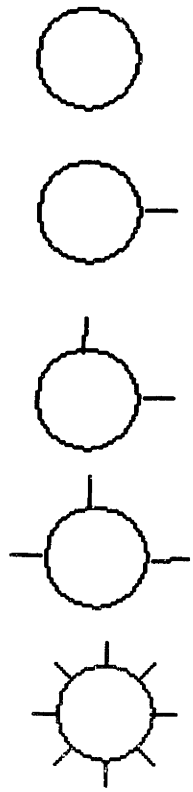

Figure 3-12: Gear recognition

approximately parallel and have approximately the same length.

A gear is represented by a WMGear Widget (See figure 3-12.). A WMGear is formed by a WCircle surrounded by four WLines of approximately the same length which are just out of the WCircle's radius and which angle toward the center of the WCircle.

A mass is represented by a WMMass Widget (See figure 3-13.). A WMMass is formed by a WPolygon. Once the user presses a button on ICEMENDR's interface

Figure 3-13: Mass recognition

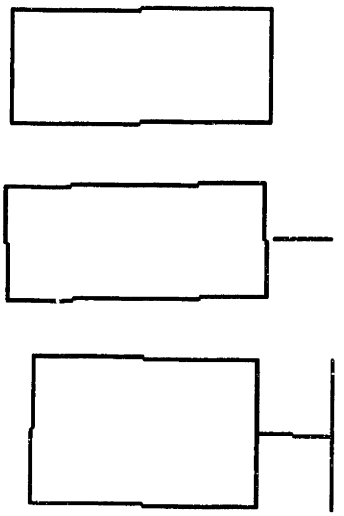

Figure 3-14: Motor recognition

to indicate that the Surface should be cleaned up, all WPolygons are replaced with WMMasses.

A motor is represented by a WMMotor Widget (See figure 3-14.). A WMMotor is formed by a WRectangle and two WLines in an approximate "T" formation on either "short" side of the WRectangle. The WLine approximately parallel to the "short" side must be approximately as long as that side, and the other WLine must be approximately half as long as the side.

A nut is represented by a WMNut Widget (See figure 3-15.). A WMNut is formed by a WRectangle and two WLines that are approximately parallel to the "short" side of the WRectangle. The WLines are also approximately evenly spaced between the two "short" side of the WRectangle.

A piston is represented by a WMPiston Widget (See figure 3-16.). A WMPiston is formed by a WRectangle and two WLines that are in an approximate "T" formation with one WLine in the middle and approximately parallel to the "short" side of the WRectangle and the other one projecting out from the WRectangle.

A screw is represented by a WMScrew Widget (See figure 3-17.). A WMScrew is formed by two WRectangles in an approximate "T" formation.

A solenoid is represented by a WMSolenoid Widget (See figure 3-18.). A WM-

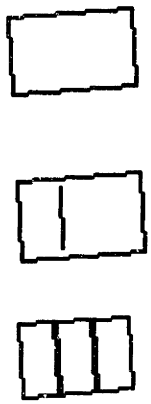

Figure 3-15: Nut recognition

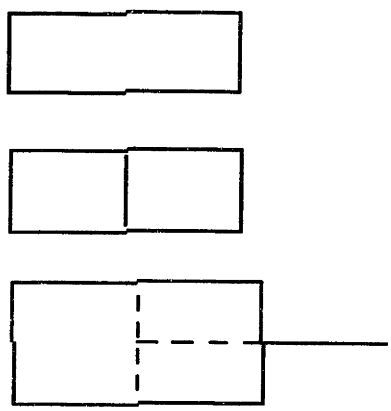

Figure 3-16: Piston recognition

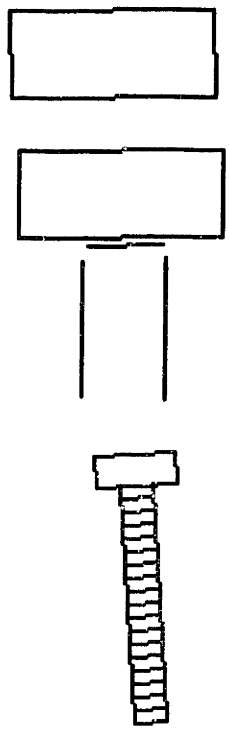

Figure 3-17: Screw recognition

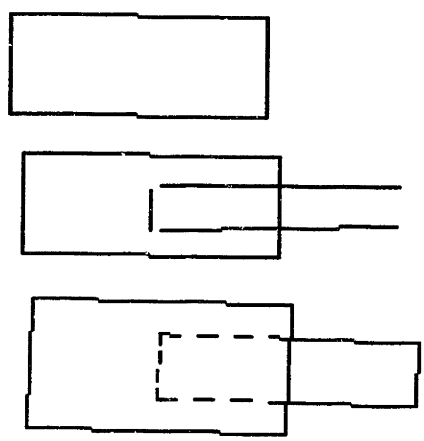

Figure 3-18: Solenoid recognition

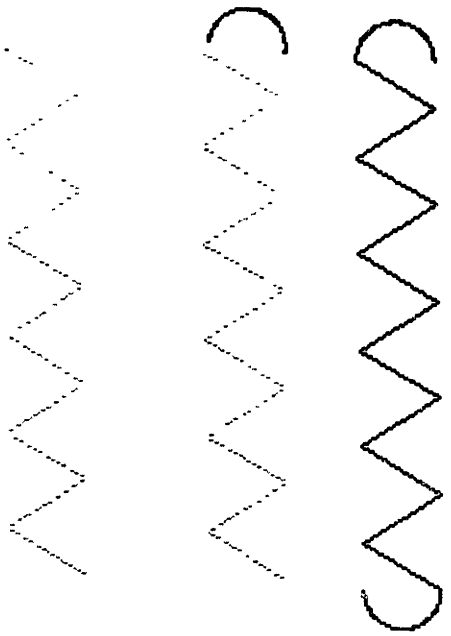

Figure 3-19: Tension Spring recognition

Solenoid is formed by two WMRectangles. One of the WRectangles is inside and projects out of the other WRectangle.

A tension spring is represented by a WMTensionSpring Widget (See figure 3-19.). A WMTensionSpring is formed by a WMSpring with two WArcs connected to the end of the WSpring, one at each end, drawn outwards like hooks.

A washer is represented by a WMWasher Widget (See figure 3-20.). A WMWasher is formed by two concentric WOvals whose major axes have approximately the same angles.

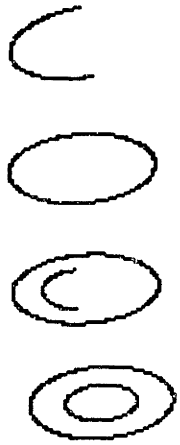

Figure 3-20: Washer recognition

3.3 Summary

ICEMENDR is written in a highly modular, object-oriented fashion. Several Java packages encompass the graphical interface, the underlying recognition system, and the mechanical-engineering-specific recognition knowledge.

Chapter 4

Conclusion

This chapter discusses what the system can do and what future work can be done, and makes some conclusions about ICEMENDR.

4.1 Future Work

ICEMENDR is able to do simple recognition of eighteen mechanical engineering parts. It implements error correction through buttons on a graphical user interface. Although ICEMENDR forms a solid basis for mechanical engineering sketch recognition, there is a variety of areas for future work.

ICEMENDR's use of stylized recognition can be improved through Recognizer redundancy. Although the drawing order of the Widgets does not matter when drawing a mechanical engineering part under ICEMENDR, the number and type do matter. In order to make ICEMENDR even more useful, multiple Recognizers for each mechanical engineering part should be created to account for different drawing preferences. For example, it takes eight properly arranged WLines to create a WMScrew. It is possible for a user to sketch a WMScrew using one continuous stroke for the top rectangle and another continuous stroke for the lower rectangle. To allow for this to work with the current WMScrew recognizer, the RectangleRecognizer must be expanded to recognize a single Squiggle as a WRectangle if it approximates a rectangle. It is also possible for the user to draw a "T" and add some threads to it. The user might

expect this to be recognized as a WMScrew because he is used to sketching screws this way. In order to make ICEMENDR easier to use, it must be able to recognize many common ways that a sketch could be drawn.

ICEMENDR can be augmented with part interaction. ICEMENDR currently does not account for part interaction. Figure 1-2 illustrates that when a WMWasher is put over a WMScrew they do not interact in any way. This interaction could be accounted for by adding Recognizers which watch for specific part interactions, or a special mechanism could be put into the system specifically for part interactions that works on the surface level. Such a mechanism could require each Widget to define multiple draw and includes methods. This would account for the different "views" of the Widget that the user might see on the Surface depending on which other Widgets it interacts with. For example, WMWasher and WMNut can be specified to interact with WMScrew. When Surface draws all the VisibleWidgets, it notes where WMWashers and WMNuts are with respect to WMScrews. If WMWashers and WMNuts are within a certain distance of a WMScrew, the Surface calls a draw method and passes in the adjacent WMWashers and WMNuts as arguments. The Surface then skips calling the draw method for the adjacent WMWashers and WMNuts. The includes methods of WMWasher, WMNut, and WMScrew could be modified to check the Surface to see if there are any interactions between these parts on the Surface. If there are, then Surface calls the WMScrew's includes method with the adjacent WMWashers and WMNuts as arguments, and the result would be returned as the result of the initial includes call.

Several improvements can be made with ICEMENDR's user interface. Instead of using buttons, the Surface correction commands can be activated through voice recognition. Additionally, facilities for adding and removing modules on the fly could be added so that the system can be kept constantly running. ICEMENDR could interpret stylus button clicks as actions and could have methods for serializing Widgets so that they can be saved and accessed on different drawing sessions.

Finally, ICEMENDR could use many more Recognizers. Recognizers for more specific parts, for resizing Widgets, and annotating parts could make ICEMENDR

much more useful. Currently, the Recognizers use relative sizing to do recognition. A single relative sizing Recognizer could be converted to several absolute sizing Recognizers to account for more specific parts. For examples, there are different types of screws, such as flat and Phillips, with different lengths.

4.2 Summary

ICEMENDR is an intelligent environment for recognizing simple mechanical engineering design sketches. Its power lies in its mechanism for hierarchical recognition which allows for parsing of complex mechanical engineering parts with simple geometric primitives. Although ICEMENDR provides a solid base for recognizing mechanical engineering design sketches, it is only a first step towards the ultimate goal of having an intelligent environment for mechanical engineering design that is both easy to use and helpful.

Bibliography

[1] E. Yi-Luen Do and M.D. Gross. Shape based reminding as an aid to creative design. *The Global Design Studio, Proc. International Conference on Computer Aided Architectural Design Futures*, 1995.

[2] E. Yi-Luen Do and M.D. Gross. Drawing analogies, supporting creative architectural design with visual references. *Preprints Computational Models of Creative Design*, pages 37–58, 1996.

[3] E. Yi-Luen Do and M.D. Gross. Drawing as a means to design reasoning. *Artificial Intelligence and Design*, 1996.

[4] M. D. Gross. Recognizing and interpreting diagrams in design. *International Conference on Image Processing*, pages 308–311, 1995.

[5] M. D. Gross. The electronic cocktail napkin-a computational environment for working with design diagrams. *Design Studies*, 17(1), 1996.

[6] Vladimir Hubka, M. Myrup Andreasen, and W Ernst Eder. *Practical Studies in Systematic Design.* Buttersworths, Boston, Massachusetts, 1988.

[7] Jerome F. Mueller. *Standard Application of Mechanical Details.* McGraw-Hill, New York, New York, 1985.

[8] D. Rubine. Specifying gestures by example. *Computer Graphics*, 25(4), 1991.

[9] E. Saund and T.P. Moran. Perceptual organization in an interactive sketch editing application. *Proc. International Conference on Computer Vision*, 1995.

[10] E. Tjalve, M. M. Andreasen, and F. Frackmann Schmidt. *Engineering Graphic Modelling, A Workbook for Design Engineers.* Newnes-Buttersworths, Boston, Massachusetts, 1979.

THESIS PROCESSING SLIP

FIXED FIELD: ill. _____ name _____

index _____ biblio _____

▶ COPIES: Archives Aero Dewey Eng Hum
Lindgren Music Rotch Science

TITLE VARIES: ▶ ☐ _____

NAME VARIES: ▶ ☑ Deepak _____

IMPRINT: (COPYRIGHT) _____

▶ COLLATION: 46 P

▶ ADD: DEGREE: _____ ▶ DEPT.: _____

SUPERVISORS: _____

NOTES:

cat'r: _____ date _____

page:

▶ DEPT: E.E.

▶ YEAR: 1999 ▶ DEGREE: M.Eng.

▶ NAME: MUZUMDAR, Manoj D.

TJ5044