DEVELOPMENT OF A PRACTICAL SOFTWARE TOOL FOR THE DESIGN OF ROLLS FOR NEAR NET SHAPE PROFILE ROLLING

A Thesis Presented to

The Faculty of the College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirement for the Degree

Master of Science

by

Christian E. Fischer

March, 1994

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Jay Gunasekera, for his advice and support during my work.

I would also like to thank Mr. Rich Diebolt of United Technologies Pratt & Whitney, the program manager of this project, for his input, suggestions, and comments.

I wish to acknowledge the assistance of Farid Masri, who, with my guidance, wrote a substantial portion of the actual LISP code. I also wish to acknowledge Anbu Rathinavel, who assisted me in code development and wrote the menu interfaces for AutoCAD.

Finally, I offer my greatest thanks to my parents, without whose support and encouragement I would have never made it this far.

TABLE OF CONTENTS

Table Of Contents (continued)

Table of Contents (continued)

LIST OF FIGURES

List of Figures (continued)

1.0 INTRODUCTION

The desire to reduce material loss and machining costs in manufacturing has led to increased use of near net shape forming processes. One of these processes is near net shape rolling, or profile rolling. In profile rolling, a flat strip or a ring is rolled between two rolls which have the desired resultant profile cut into them (see figure 1.1).

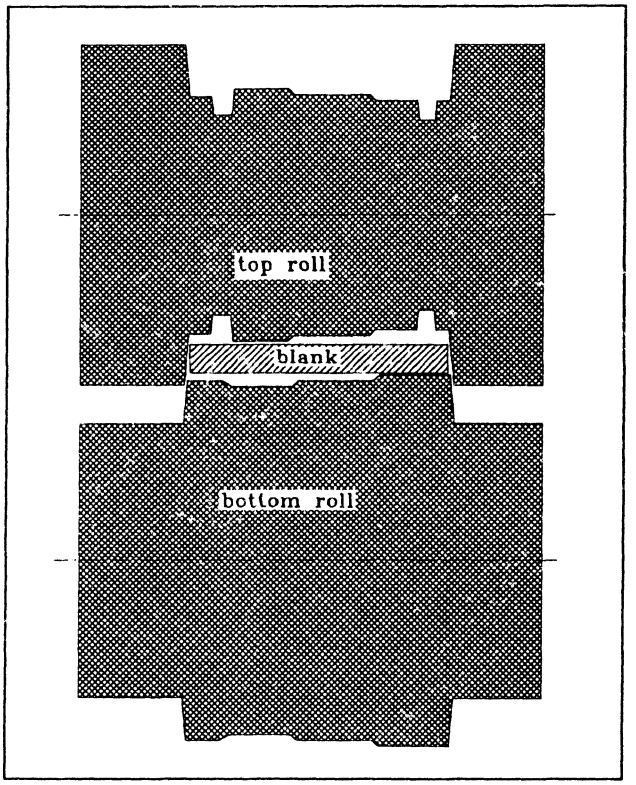


Figure 1.1: Profile rolling

Profile rolling or ring rolling of arbitrary shapes involves complex metal flow patterns which are difficult to predict. For various reasons, the profile of the product will not always match the profile of the rolls. In many cases, it is necessary to first roll a preformed product with a simpler geometry than the

desired shape, then roll the final shape from that preform. The design of preform geometries is a difficult task which relies on experience and trial and error. Several graphical heuristic techniques are used. Until now, these techniques have not been available in any computerized or automated form.

The purpose of this work was to develop a systematic methodology for the computer aided design of preforms for profile rolling. The software and application methods developed were part of a larger project for the design and analysis of rolls and the rolling process for profile rolling and profile ring rolling of aircraft engine compressor shrouds. The project was sponsored by United Technologies - Pratt & Whitney, under the United States Air Force Industrial Modernization Incentive Program, contract S33657-90-C0014,P00002. For a complete discussion of this project, refer to [27].

The product of the work was a software package written in AutoLISP and running in AutoCAD, and a systematic approach to designing and analyzing preform profile sequences. It forms the Process Design portion of the roll design software package ROLLCAD, which was developed under the project mentioned above. An emphasis was placed on development of user-friendly programs. AutoLISP programming allows the newly developed routines to be called from the AutoCAD editor exactly like any other AutoCAD command. The AutoCAD topline menu was also modified to include the functions (see figure 1.2). The software includes tools for geometric manipulations of the shape to either modify the shape in preparation for rolling or to generate a preform shape based on two existing shapes. It also

includes tools to analyze the shape and assist the user in determining its suitability for rolling.

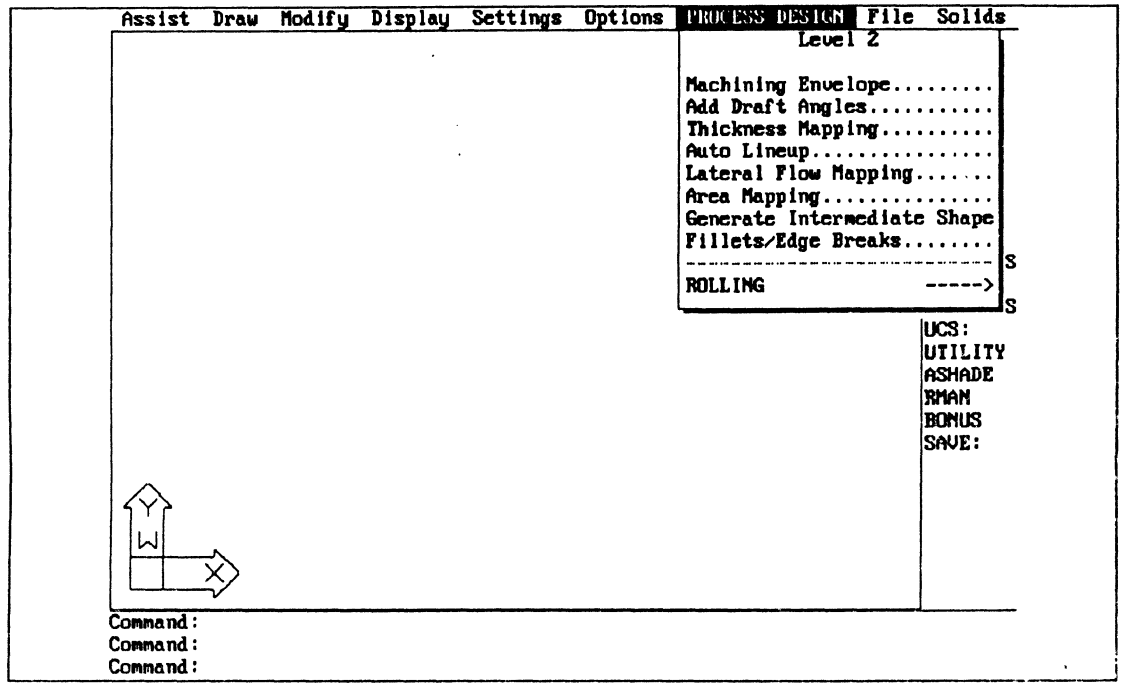


Figure 1.2 AutoCAD modified menu structure incorporating Process Design programs

When used by an experienced roll designer, these tools will considerably speed the design process. The analysis tools give the designer an indication of whether the design will work. Without these tools, experimentation or finite element analysis must be used. Both of these approaches are expensive and very time consuming. A single design iteration on part geometry may take several days with FEM, or considerably longer with experimentation. The tools discussed here allow several geometric iterations to be performed in a single design session. When used with the rest of ROLLCAD, it becomes an extremely powerful design tool.

1.1 ROLLCAD

As noted, the software discussed in this thesis is a module in a much larger software package (ROLLCAD). In order to have a full appreciation of the process design tools, it is necessary to have an understanding of the complete package. It runs on a PC in the AutoCAD environment using a modified menu structure. The primary features of the package are:

- Tools to assist in preform design, including addition of machining envelopes, draft angles, and fillets and edge breaks, thickness mapping, lateral flow mapping, and intermediate shape generation through area mapping.

- Simulation of profile strip rolling, including estimation of roll forces and torque, machine horsepower requirements, strain in the material, temperature rise, and length of the resultant part.

- Simulation of profile ring rolling, including ring growth rates, rolling times, centering forces, rpm of the inner roll to avoid slippage, as well as the machine and material parameters provided for strip rolling.

- Constant volume calculations for calculating linear growth in strip rolling, diametric growth in ring rolling, and required strip length for a given ring diameter.

- Utility to convert two dimensional part and roll geometries to PATRAN neutral file format for subsequent finite element mesh generation and analysis.

This thesis is concerned with items listed in the first point.

The AutoCAD user interaction format has been followed wherever possible. For simulation routines, which are calculation intensive, execution time would be unacceptably long in AutoCAD. These routines have been written in Fortran, but are still called from the AutoCAD menu system.

1.2 Thesis Organization

Chapter 2 provides a brief review of literature and other research in the field of profile rolling and profile ring rolling. The literature is extremely limited, and the bulk of the work has been experimental. The theory of preform and roll pass design is discussed in chapter 3. The systematic design methodology developed in this work is presented in this chapter, and the software tools are introduced. Chapters 4 and 5 describe the software. Chapter 4 contains an overview of AutoLISP and drawing entity handling in AutoCAD. Chapter 5 contains a detailed discussion of the software, including theory, development, implementation, and the role of each subroutine. Chapter 6 includes a discussion of results, conclusions, and recommendations for future work. There are two appendices. The first contains excerpts from the ROLLCAD users manual for the Process Design

software discussed in this thesis. The second contains the AutoLISP source code for the Process Design tools.

1.3 Acknowledgement of Contributors

The development of the ROLLCAD package was truly a team effort. Contributions were made by over a dozen people at Ohio University and Pratt & Whitney. The author is responsible for the theory behind all of the process design software tools, (with the exception of the "Add Machining Envelope" routine developed by Anbu Rathinavel) and for the methodology of their application. However, in order to meet time constraints on the software delivery, a significant amount of the actual AutoLISP code writing was done by Farid Masri [26]. Because the work is so closely integrated, it has all been reported in this thesis. Comments in the source code in Appendix 2 clearly state who was involved with the writing of each module.

2.0 LITERATURE SURVEY

Rolling of profiled sections is an established practice for common shapes such as structural shapes and railroad rail. Hot profile ring rolling is used for products as diverse as pipe weld flanges and aircraft engine casings. Recently, the use of near net shape rolling and ring rolling has become popular. As with any near net shape process, material flow and roll fill are important considerations. Predicting material flow and designing preform shapes to achieve proper fill is more an experienced based art than a science. However, numerous researchers have studied the topic, both experimentally and theoretically, in an attempt to better describe the phenomenon.

In his book *Fundamentals of Rolling* Wusatowski [1] offers empirical equations to predict elongation and spread in straight sections. These equations are complex and rely on empirical charts factoring roll diameter and part geometries. The first references to profile ring rolling in the literature was published by Mamalis, et. al. [2] in 1975. These authors studied cavity formation in the OD of T-shaped rings at large reductions of ring wall thickness. In 1979, Hawkyard and Ingham performed experiments using grooved rolls to produce a variety of internal and external profiles. This work, as well as much of the other early research into the general ring rolling process was performed in the late 1960s and 1970s at UMIST in Manchester, England. A review of this work and other research into ring rolling was published by Johnson and Mamalis in 1979 [5].

In 1976, Winship [3] published an article discussing near net shape cold ring rolling of aircraft engine compressor shrouds at General Electric Aircraft Engines. No details of the design process are given. In an associated interview article, Vern Fenci of Grotnes Machine Works, noted that roll geometry is the most important factor in the cold ring rolling process, because it determines how much metal is to be moved in various stages. However, he observes, there is no formula for determining this. Instead, it is a trial and error process.

The first analytic work for ring rolling non-rectangular sections was published in 1981 by Yang, et al. [6]. The upper bound analysis technique was applied to profile ring rolling of I sections. The roll torques were calculated for various process conditions and verified with experiments on aluminum alloy.

Hutcheon and Hawkyard investigated ring rolling of aluminum cycle wheel rims [7,11]. The cycle wheels have a complex profile shown in figure 2.1. The goal of the research was to produce a wheel rim from a rectangular section ring blank in as few forming stages as possible, while achieving the required cross section, diameter, and circularity. Figure 2.1a illustrates the profiles of preforms and the final shape. 2.1b illustrates the material distribution across the preform, and the lateral displacements required to transform a rectangular section ring blank to the profile in a single operation. Profile II in 2.1a is a "splayed-out" version of the final profile.

Moussa and Hawkyard [8] published the results of a detailed study of profile development and roll force in profile ring rolling in 1984. The

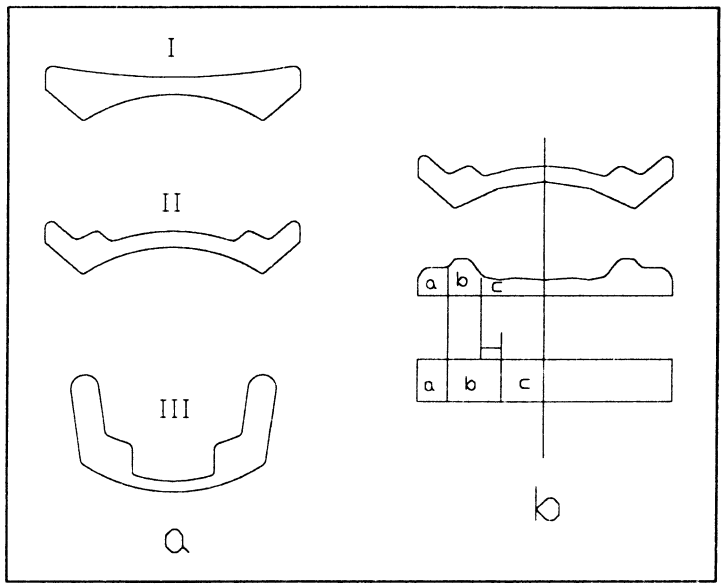


Figure 2.1 Bicycle rim profiles developed by Hutcheon and Hawkyard, showing material distribution and lateral flow (after [11])

effects of roll closure rate, lubrication, blank shape, and small variations in geometry were studied. Many profiles which could not be developed from rectangular profiles were successfully formed with preforms.

Hirai, et al. [10] described a study of profile fill in T-shaped rings with external profiles using layered plasticine rings. The rings were sectioned after rolling to allow the study of flow patterns in the cross section. Eulerian finite element simulation was also used for comparison. A novel method of resisting circumfrential growth in the ring by applying a braking force was investigated. Good results were obtained using plasticine rings in a small laboratory mill.

Recently, several authors have applied computer modeling techniques to ring rolling. In 1987 Doege and Aboutour [16] derived a general upper-bound model

for both plane and profile ring rolling. In 1991, Hahn and Yang [22] applied the Upper Bound Elemental technique to predict roll torque and geometry. Roll torque predictions were in good agreement with experimental results, but geometry predictions were not as satisfactory. Since that time, several authors [19,21,23] have developed finite element simulations of the ring rolling process. These simulations show reasonable results, but are extremely CPU intensive.

Kang and Kobayashi [21] proposed a process of preform design using finite element analysis. The process uses a backward tracing technique. This backward tracing scheme has also been applied to preform design axisymmetric and plane strain forging problems. This is the first extension to a three-dimensional problem. The theoretical results are good, but the paper does not address the problem of manufacturing the preform generated by the backward tracing.

Eruc and Shivpuri [24,25] have recently conducted an extensive literature survey into ring rolling technology, including machines, processes, production lines, process modeling, simulation, planning, and control. They observe that intricate profiles and near net shape forming demands make definition of rolling parameters and analysis of deformation patterns difficult.

In their discussion of profile filling, Eruc and Shivpuri note that a profile of a pass can be filled easier if:

- the local degrees of deformation do not deviate excessively from the mean degree of deformation.

- the thick walled sections that require less reduction constitute a greater percentage of the total section,
- the material flow is not hindered by sharp corners, narrow gaps, friction, etc.
- the blank profile is designed such that lateral flow of material in the cross section is minimized.

Very little research into preform design is discussed. Researchers in Aachen, Germany [12,13] are reported to have investigated backward tracing for rectangular cross sections, but no details are discussed. Rachakonda [20] also implemented a backward tracing scheme for rectangular profiles. Aside from the finite element approach noted earlier, the literature makes no mention of preform design for profiled sections.

3.0 PREFORM AND ROLL DESIGN THEORY

This chapter describes an approach for generating roll geometries using the Process Design tools in ROLLCAD as well as standard AutoCAD functions given the desired final shape in an AutoCAD drawing file. It is expected that the design will be completed by an experienced roll designer with a working knowledge of AutoCAD.

The steps of the design sequence are:

- Obtain AutoCAD drawing of net shape part profile
- Modify part profile, adding machining envelope, draft angles, fillets, and any other modifications necessary for rolling
- Analyze the shape, select starting stock size, and develop a preform sequence
- Split the part profiles and generate roll profiles
- Check the fit of each preform in the next roll in the sequence
- Continue with analysis with other ROLLCAD modules.

This chapter describes the theory of roll profile fill, then describes methods for using the Process Design tools and existing AutoCAD functions to perform each step.

3.1 Profile Fill

The design of compression rolls for profile rolling presents several challenges. The primary goal is to design a shape or sequence of shapes that will provide roll fill -- that will produce a part profile that conforms to the profile cut into the rolls. Once this shape sequence has been developed, it must be verified that a suitable grain flow pattern is achieved. A rolling strategy must then be developed considering force and horsepower limits on the rolling mill being used, and the total reduction between anneals for the material. The Process Design software assists with the first step -- designing a sequence of preform profiles which will allow the desired final profile to be reached.

Profile fill is a function of material flow in the roll gap. In the simplified model, there are two directions for material to be displaced when it undergoes compression (see Fig. 3.1) The first direction is longitudinal, or in the direction of rolling (A in figure 3.1). The second is lateral, or perpendicular to the direction of rolling (B in figure 3.1). In order for a profiled part to be formed from flat stock, sufficient lateral material flow must occur to fill the thicker sections. Opposing this lateral flow is a tendency for thicker sections to be stretched as thickness reduction and longitudinal extension occurs in thinner sections. This is illustrated in figure 3.2. Sections A and B are originally the same length. As section B undergoes greater reduction, its longitudinal extension is greater than section A. However, when the sections are joined, section A is stretched along with section B, causing incomplete fill in section A. This phenomenon is compensated by first rolling an easy to form section with a thickness distribution similar to the desired shape.

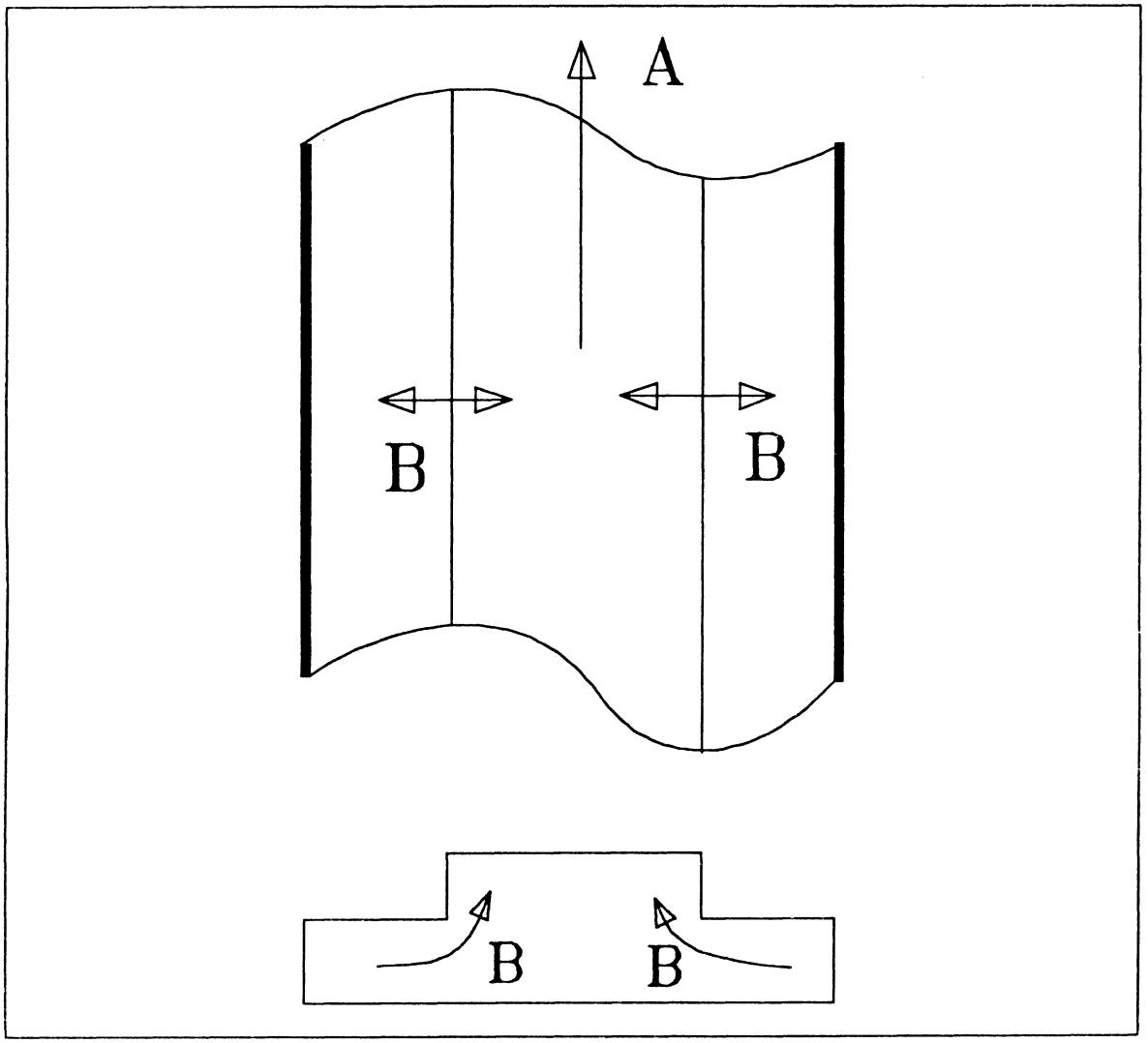


Figure 3.1 Primary flow directions in profile rolling.

Thus the lateral material flow required to attain roll fill is lower, and underfilling is less likely.

3.2 Software Tools to Aid in Predicting Roll Fill

ROLLCAD provides three tools for assisting in preform design and analysis. They are LATERAL FLOW MAPPING, THICKNESS DISTRIBUTION PLOT, and AREA MAPPING. The first two are analysis tools, and the third is a design tool

which generates an intermediate preform when presented with an initial and final shape.

3.2.1 Lateral Flow Mapping

LATERAL FLOW MAPPING plots a graph of the lateral material movement required for the indicated shape to fill from the desired preform. The height of the graph is proportional to the distance material at a particular section of the preform must move to fill the final shape. A sequence with less lateral flow is more likely to fill. This is illustrated in figures 3.3 and 3.4. Figure 3.3 shows the lateral material flow required to fill the target (lower) shape from the proposed preform (upper) shape. Figure 3.4 shows the lateral material flow for a modified version of the preform. Note that the peak material flow is much smaller, although there is still room for improvement. The flow is also much more localized.

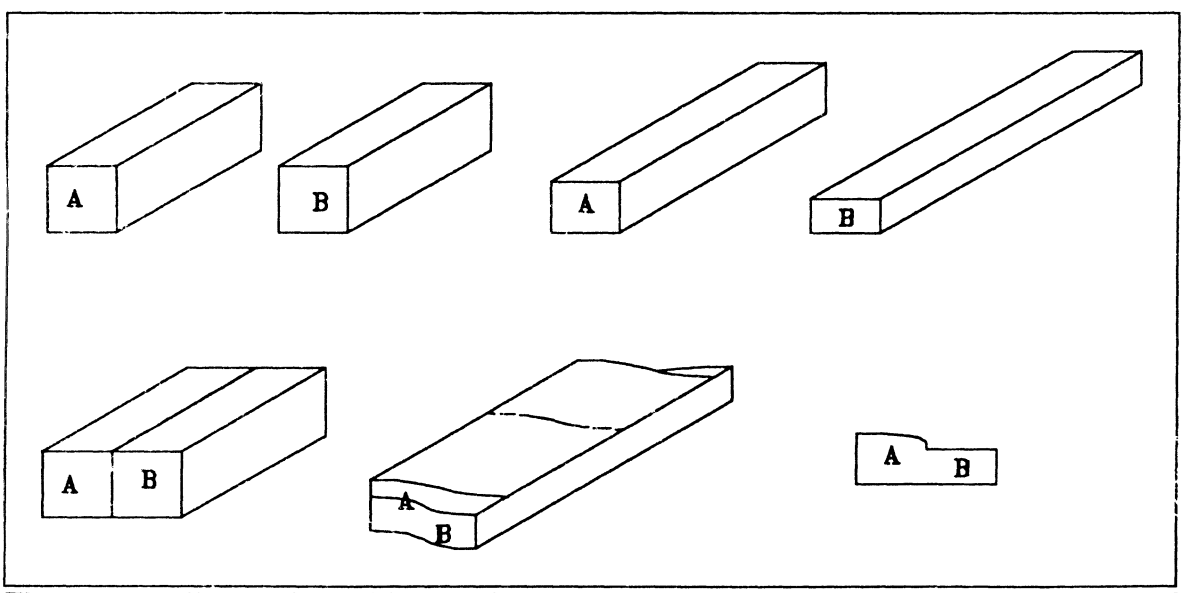


Figure 3.2 Illustration of incomplete profile fill in section A due to extension of thinner section B

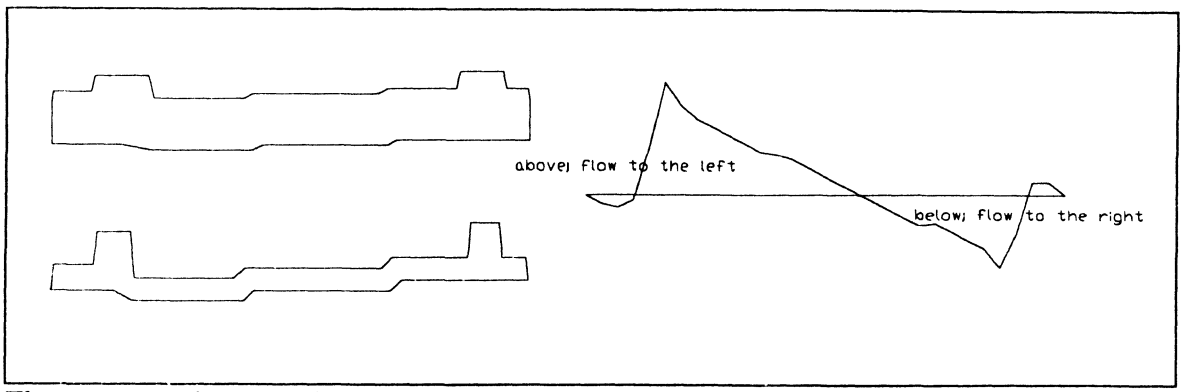


Figure 3.3 Lateral flow plot example 1

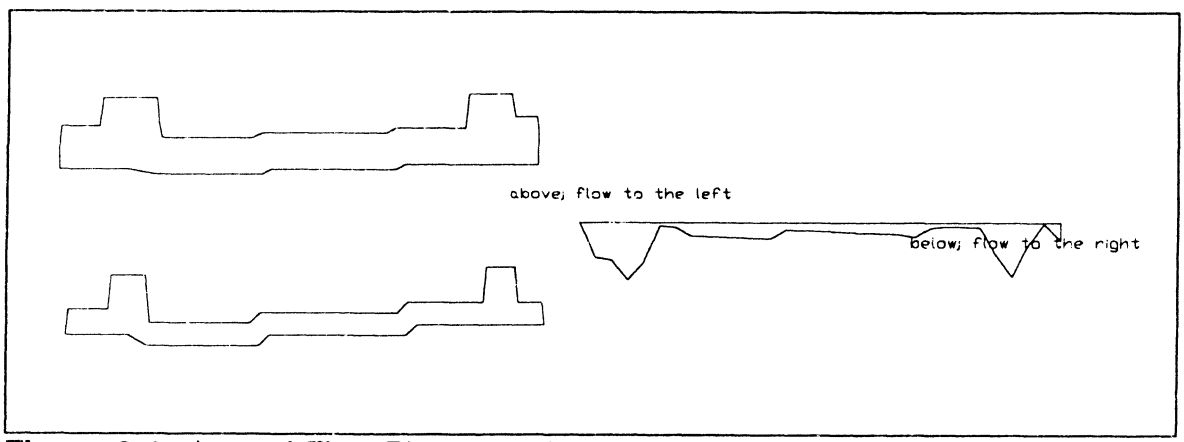


Figure 3.4: Lateral Flow Plot example 2

3.2.2 Thickness Distribution Plot

The THICKNESS DISTRIBUTION PLOT displays thickness distribution for complex shapes, allowing the designer to get a better idea of what thickness distribution is required in the preform. It may also be used to compare the material distribution

of two or more shapes in a sequence by overlapping the plots. This is illustrated in figures 3.5 and 3.6.

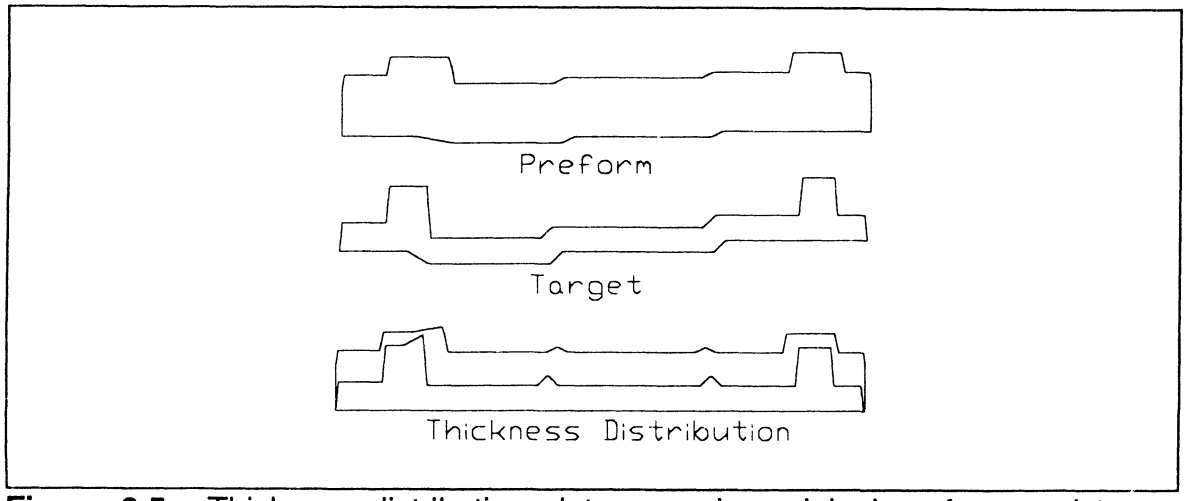


Figure 3.5: Thickness distribution plot comparing original preform and target shape.

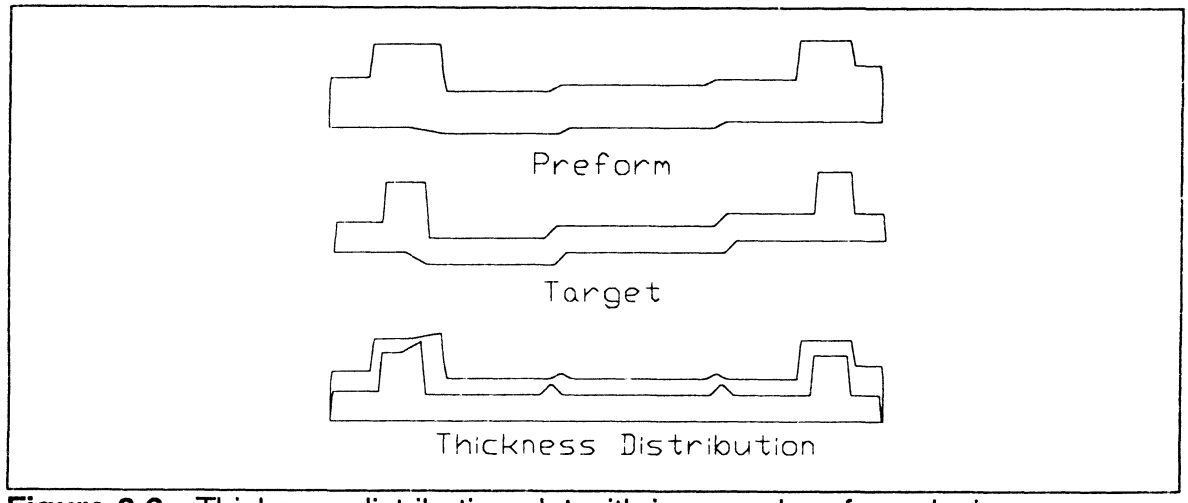


Figure 3.6: Thickness distribution plot with improved preform design

3.2.3 Area Mapping and Intermediate Shape Generation

In some complex situations, more than one preform step may be required to achieve fill. In this situation, AREA MAPPING and INTERMEDIATE SHAPE GENERATION may be useful in generating an intermediate preform shape. These functions generate a geometric average of two shapes as illustrated in figure 3.7.

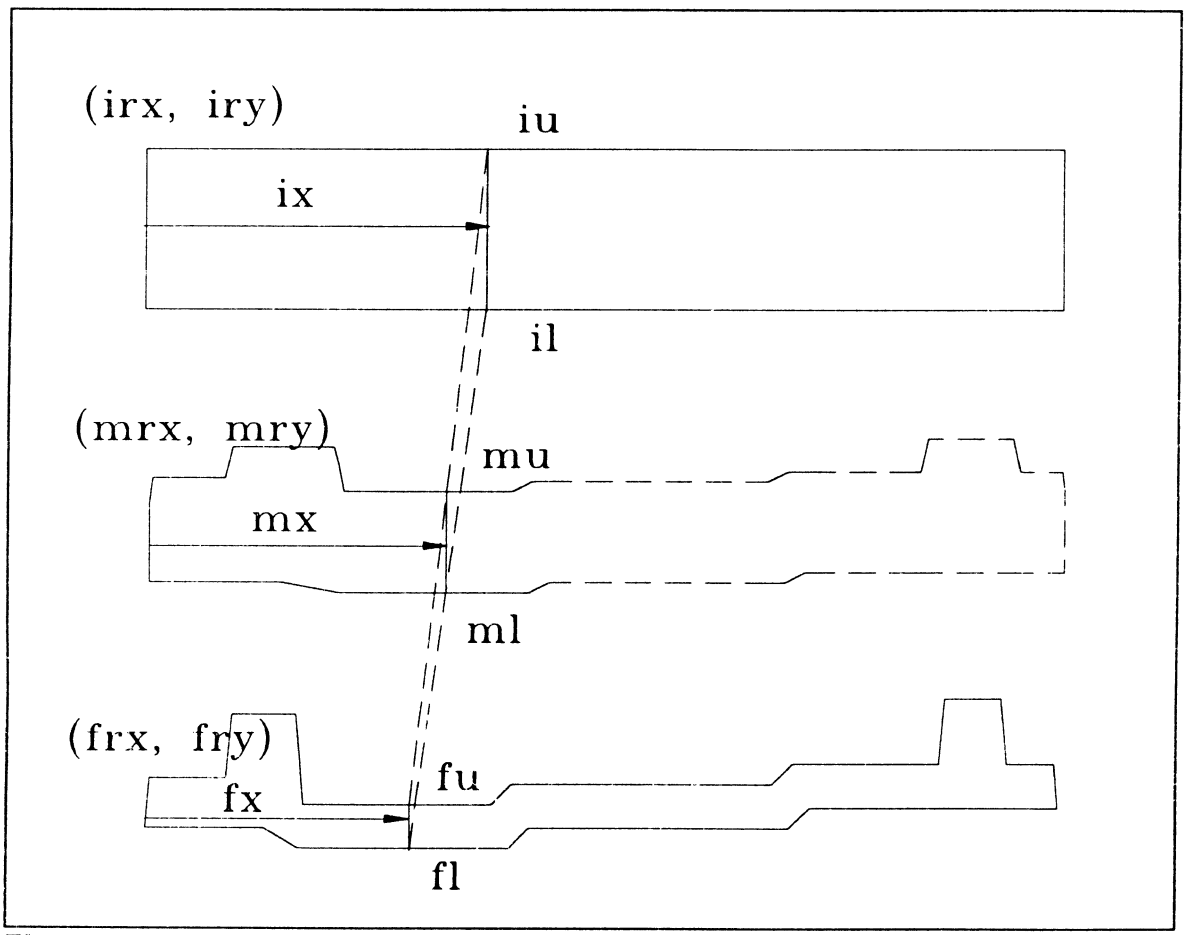


Figure 3.7: Generation of preform by mapping

3.3 Other Roll Design Features

ROLLCAD contains two other features which are useful for designing rolls. ADD DRAFT ANGLES allows you to select vertical edges in a part and adjust them to a specified angle. The FILLETS/EDGE BREAKS function adds fillets and edge breaks to the entire part or specified corners.

3.4 Generating Roll Profiles in AutoCAD

Once a desired part profile is obtained, the AutoCAD drawing editor can be used to convert the profile shape to roll profiles. It is assumed that rolling will be done with a closed pass. The COPY command is used to generate a duplicate part profile. The MOVE command is then used to separate the upper half of the shape from the lower half. The LINE command then can be used to draw in the rest of the upper and lower roll profiles. This is illustrated in figure 3.8.

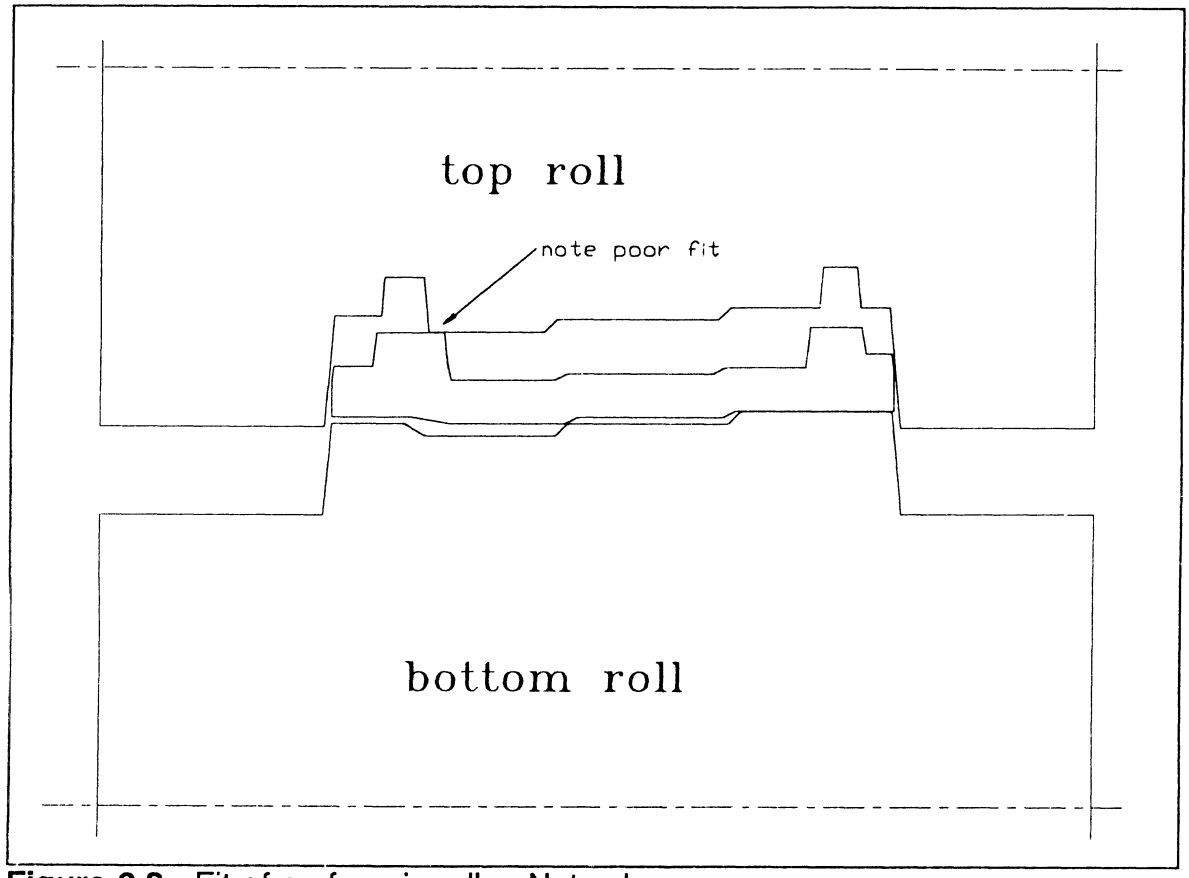


Figure 3.8: Fit of preform in rolls. Note sharp corner.

4.0 LISP IMPLEMENTATION OF PROCESS DESIGN TOOLS

All process design tools were developed in AutoCAD using AutoLISP. For a brief discussion of AutoLISP and entity handling in AutoCAD, refer to appendix 2.

4.1 Basic Shape Processing

All process design routines work on existing shapes. There are basic steps that all of the routines use to begin processing shapes. Figure A2.1 is repeated here as figure 4.1 for reference.

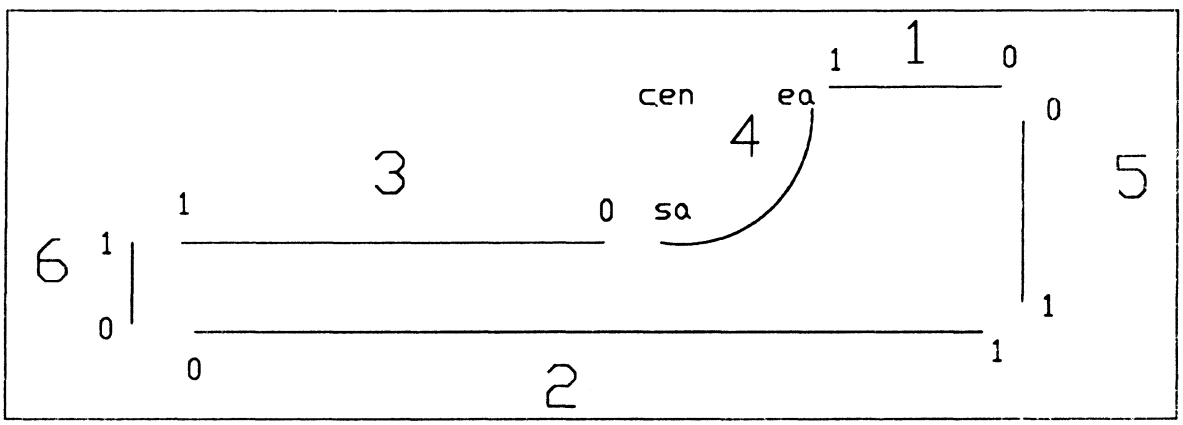


Figure 4.1 Typical AutoCAD drawing entities before shape processing

1. When a routine is called, all environment variables, such as current layer, current color, object snap mode, etc. are stored so they can be reset at the conclusion of the routine.

2. A shape selection routine (SHAPESEL) is called. This routine prompts the user to window a shape, then joins the shape into a polyline and re-explodes it. The new elements resulting from an

exploded polyline are arranged in consecutive order in the data file. The names of these new elements are passed back to the calling routine. Area and Lateral Flow Mapping call this routine twice (ie, they process two shapes). All other routines call it once.

3. The list returned from SHAPESEL containing entity names is passed to ARC2LINE. ARC2LINE converts all arcs to straight line approximations. Arcs are broken into segments of maximum 20° included angle. The chord of each of these line segments is drawn. This provides a reasonable approximation of the arcs, and greatly simplifies shape processing. ARC2LINE returns a modified entity name list to the calling routine.

Figure 4.2 shows the same shape as illustrated in figure 4.1 after it has undergone standard processing.

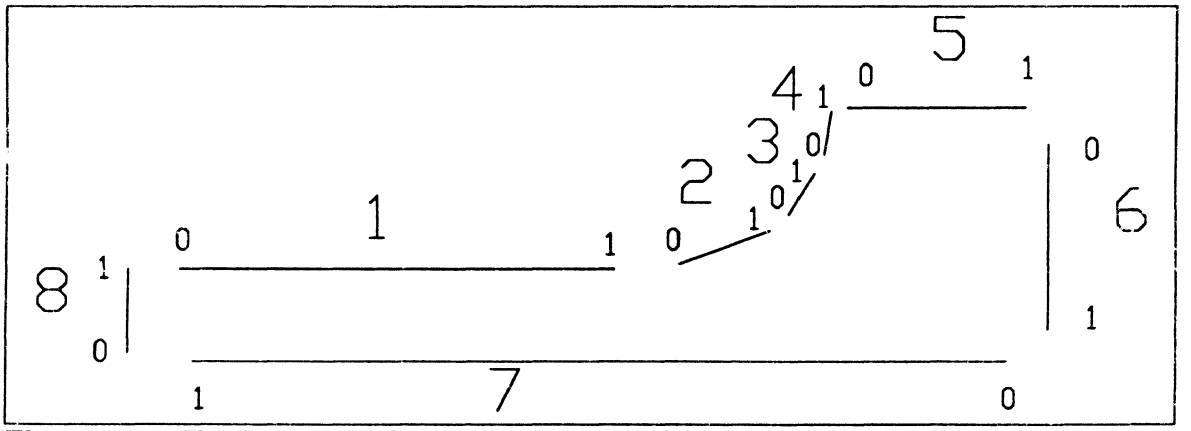


Figure 4.2 Typical AutoCAD drawing entities after shape processing

At this point, the main calling routine continues with its specified task.

4.2 Process Design Tools

All Process Design programs use hierarchical programming techniques, where utility subroutines are called by a main routine. The general theory behind each tool is discussed here. A detailed description is included in section 4.3.

4.2.1 Machining Envelope

Machining envelope utilizes the AutoCAD **offset** command. The desired shape is selected. The user is prompted for an offset distance. The shape and offset distance are then passed to the **offset** command using the AutoLISP **command** function to invoke AutoCAD commands.

4.2.2 Draft Angle

The calling order of routines is shown in figure 4.3. A shape is selected, and the arc to line conversion is performed as described above. A preliminary default draft angle of 5° is set. The program then enters a continuous loop which is exited when the user types <ENTER>.

Inside the loop, the user is prompted to select the entity to be modified, then the endpoint about which that entity is to be pivoted (the stationary endpoint.) The user is given the opportunity to modify the default draft angle. If this is done, the new angle becomes the default for the remainder of the executions. The user is

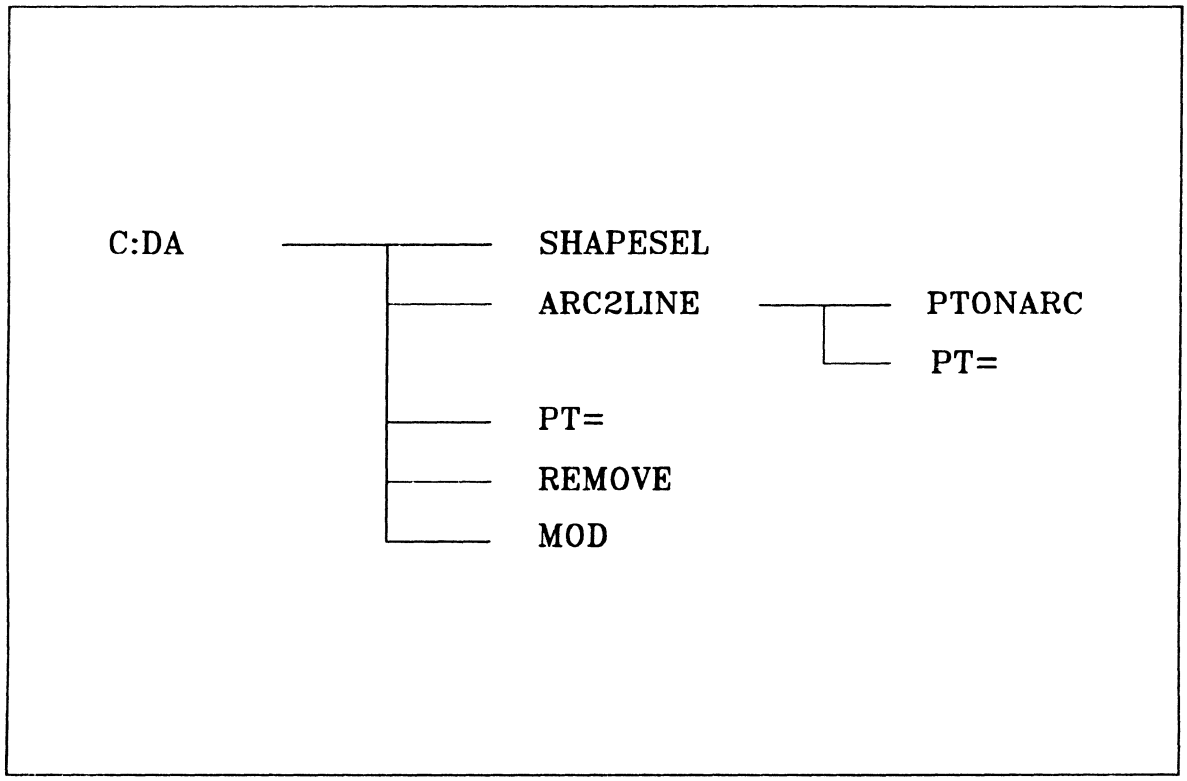


Figure 4.3 Hierarchy of LISP modules in the Draft Angle routine

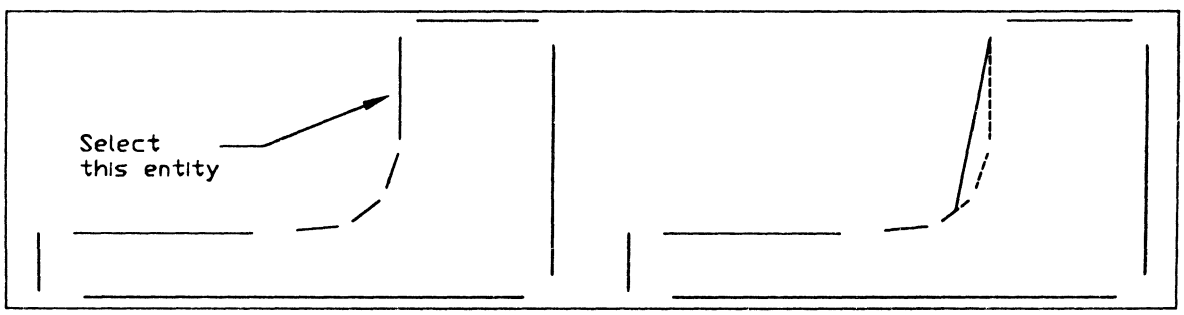


Figure 4.4 Modification of AutoCAD line entities by Draft Angle routine

then asked to indicate the offset direction with a point. Through a complex sequence of steps, the endpoint of selected entity is modified, the proper endpoint of the entity which it intersects is modified, and any entities which now lie entirely within the shape are erased.

The effect of adding draft angles is shown in figure 4.4.

4.2.3 Thickness Mapping

The calling order of thickness mapping routines is illustrated in figure 4.5. A single shape is selected, and standard processing is performed. The shape is then scanned left to right. Each time an entity endpoint is encountered, the thickness of the part is calculated at that x position. The thickness is then plotted against a horizontal line.

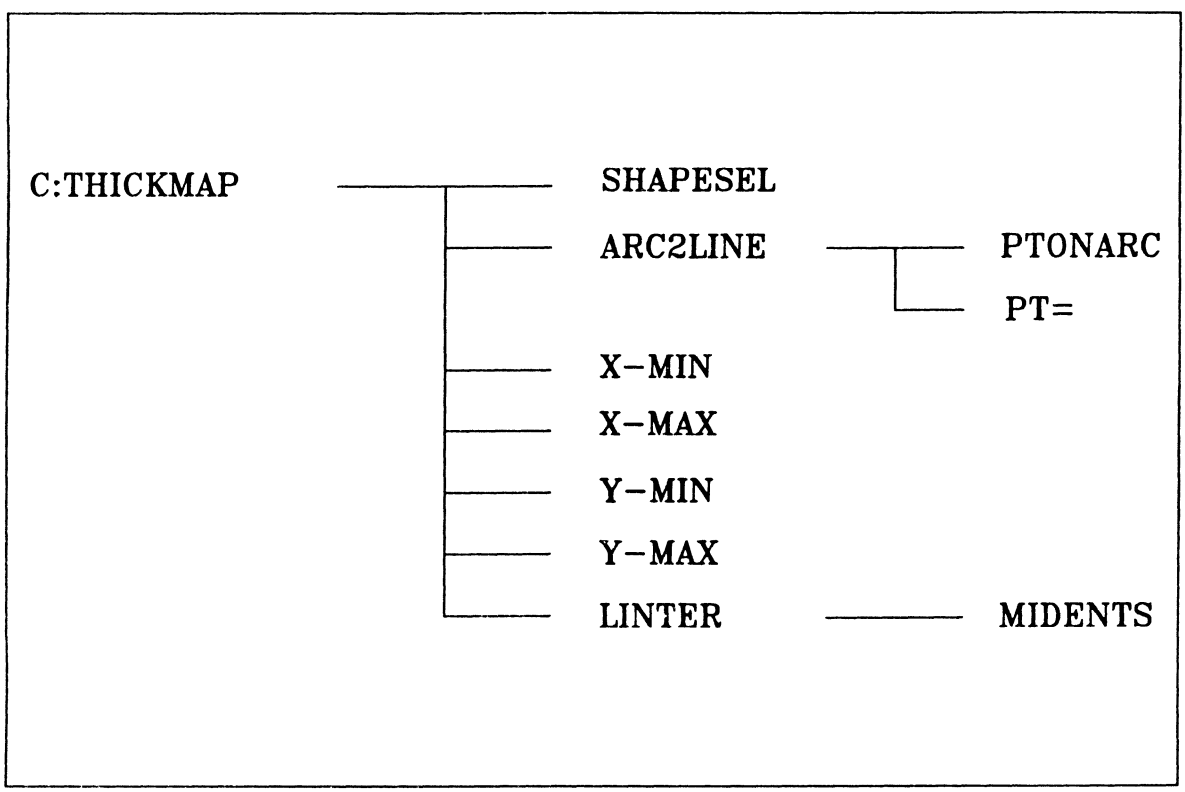


Figure 4.5 Hierarchy of LISP modules in Thickmap Routine

4.2.4 Lateral Flow Mapping

Lateral flow mapping calculates the theoretical lateral material flow required if the target shape is to be rolled from the preform shape. Axial flow is assumed to be uniform. Hence, all area calculations for the second (target) shape are scaled by the ratio of the initial area to the target area.

The equation for lateral flow is

$$Flow(x) = \int_0^x T_1(x)\,dx - \frac{A_1}{A_2}\int_0^x T_2(x)\,dx \qquad (4.1)$$

The plot of flow(x) vs. x is displayed by the function. It is assumed that the shapes are aligned properly in the drawing, thus x refers to the same absolute x measure on each shape, and is not relative to the left end of the shape being considered. The height of the plot is an indication of the relative material motion at that point. The points at which the plot crosses the horizontal axis indicate flow boundaries, where there is theoretically no movement in the material at that point other than uniaxial compression.

The calling order of lateral flow mapping routines is illustrated in figure 4.6. Two shapes are selected by the user. Both undergo standard processing. The area of each shape is obtained, and the area ratio is calculated. Referring to equation 1, the height of the lateral flow plot at a given x position is based on the area of the shape (the integral of the thickness distribution) to the left of the x position.

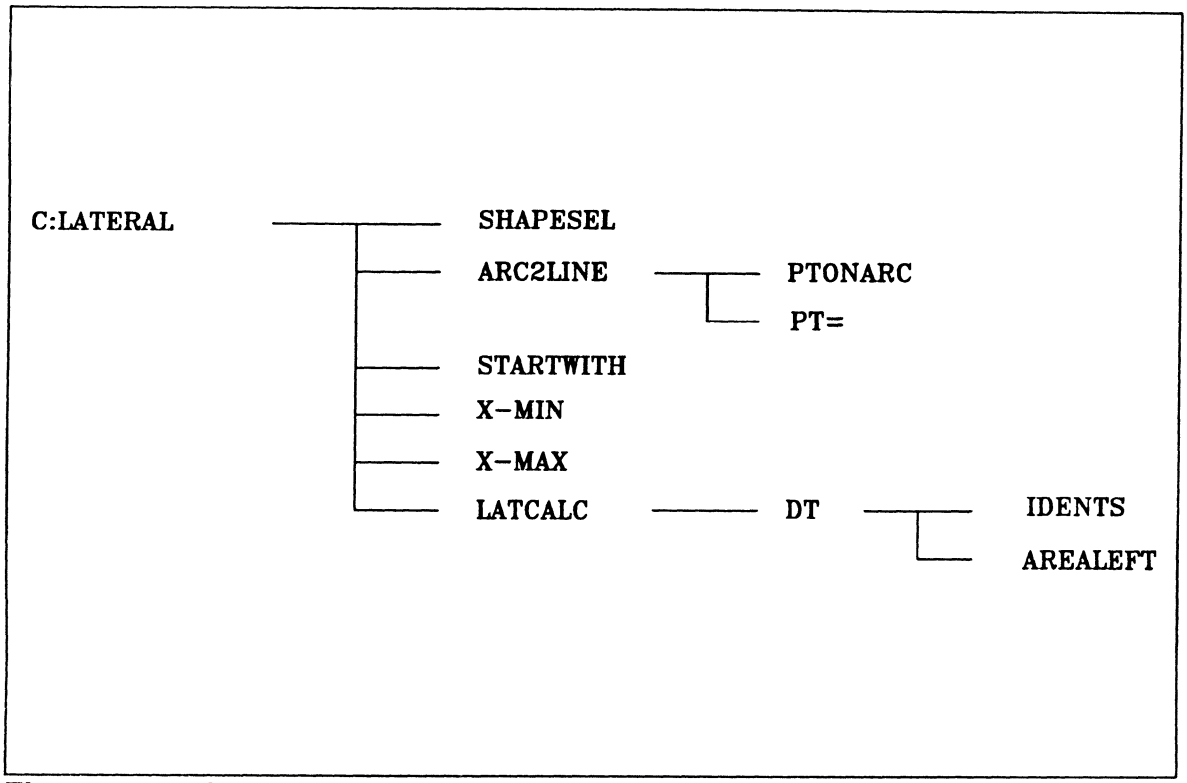


Figure 4.6 Hierarchy of LISP modules in Lateral Flow Mapping routine

The shapes are divided into a number of uniform intervals as specified by the user. The area to the left of each interval is obtained using the AREALEFT function, and the scaled difference in the areas is calculated.

4.2.5 Autolineup

Autolineup is used to align shapes before lateral flow mapping. The shapes are aligned based on their geometric centers (the midpoint between the maximum and minimum x values.)

4.2.6 Area Mapping and Generate Intermediate Shape

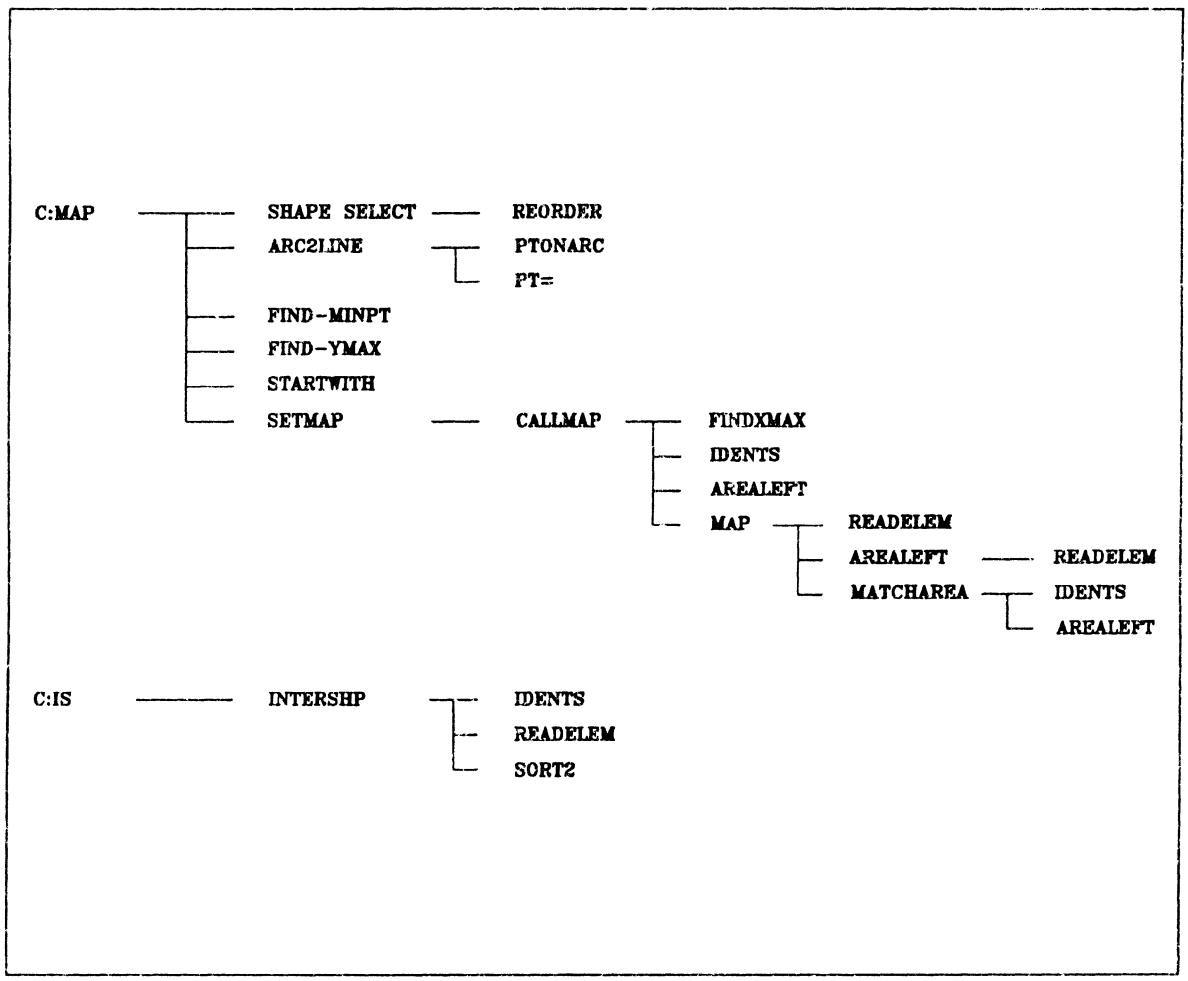


Figure 4.7 Hierarchy of LISP modules in Area Mapping and Intermediate Shape routines

Area mapping is used to generate intermediate preform shapes between an initial preform and a target shape. The area mapping process involves two separate user called functions. The first, C:MAP, generates a list of coordinates on each of the two shapes. The second, C:IS (generate Intermediate Shape), uses the list generated by area mapping, as well as the data from both shapes, to draw the intermediate shape. The calling order of both routines is illustrated in figure 4.7.

4.2.6.1 Area Mapping

Area Mapping prompts the user to window two shapes. Both shapes undergo standard processing. The shapes are then reordered so that the starting endpoint of the first entity in each shape is at the lower left corner of the shape. Control is then transferred to the CALLMAP routine, which first maps the initial shape onto the final shape, then maps the final shape to the initial shape. The actual mapping is performed by the routine MAP.

Map scans the first shape from left to right. Each time an entity endpoint is encountered, is encountered, the x coordinate of this endpoint is stored. The area to the left of a line through this point is calculated. This area is scaled by the area ratio of the two shapes, and a line is located on the second shape with a corresponding area to the left. The x position of this line is also stored (see figure 4.8).

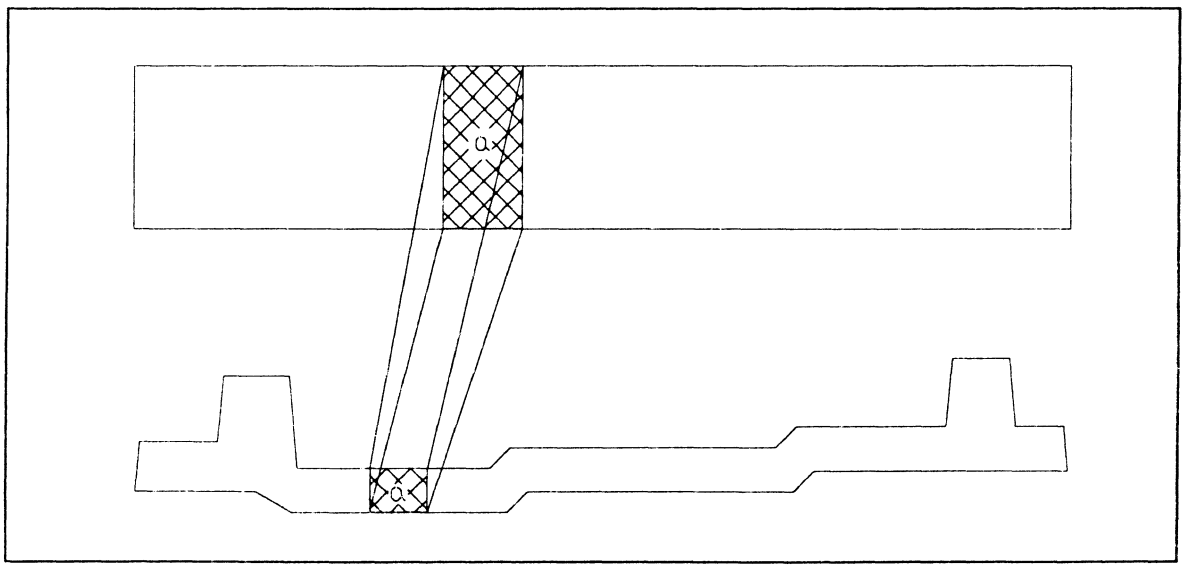


Figure 4.8 Mapping of segment of blank onto target shape.

After both shapes have been mapped onto each other, the x coordinate lists are combined into a single list with duplicate points removed. This data is stored in the variable MLIST, to be called by the intermediate shape routine.

4.2.6.2 Intermediate Shape

The routine INTERSHP is called after area mapping has been run. It uses the shape lists and mapping data generated by C:MAP. The vertical lines specified by the coordinates in MLIST are reconstructed. The points where the line intersect the top and bottom of each shape are identified. A weighted average of the corresponding coordinates is calculated using a user-supplied weighting factor. The calculated coordinates are linked with line segments, yielding the mapped intermediate shape. (see figure 4.9).

$$m_x = i_x + K(f_x - i_x) \quad (4.2)$$

$$m_u = i_u + K(f_u - f_l) \quad (4.3)$$

$$m_l = i_l + K(f_l - i_l) \quad (4.4)$$

4.2.7 Fillets/Edge Breaks

The AutoCAD fillet/edge break command is repeated in the Process Design menu for convenience.

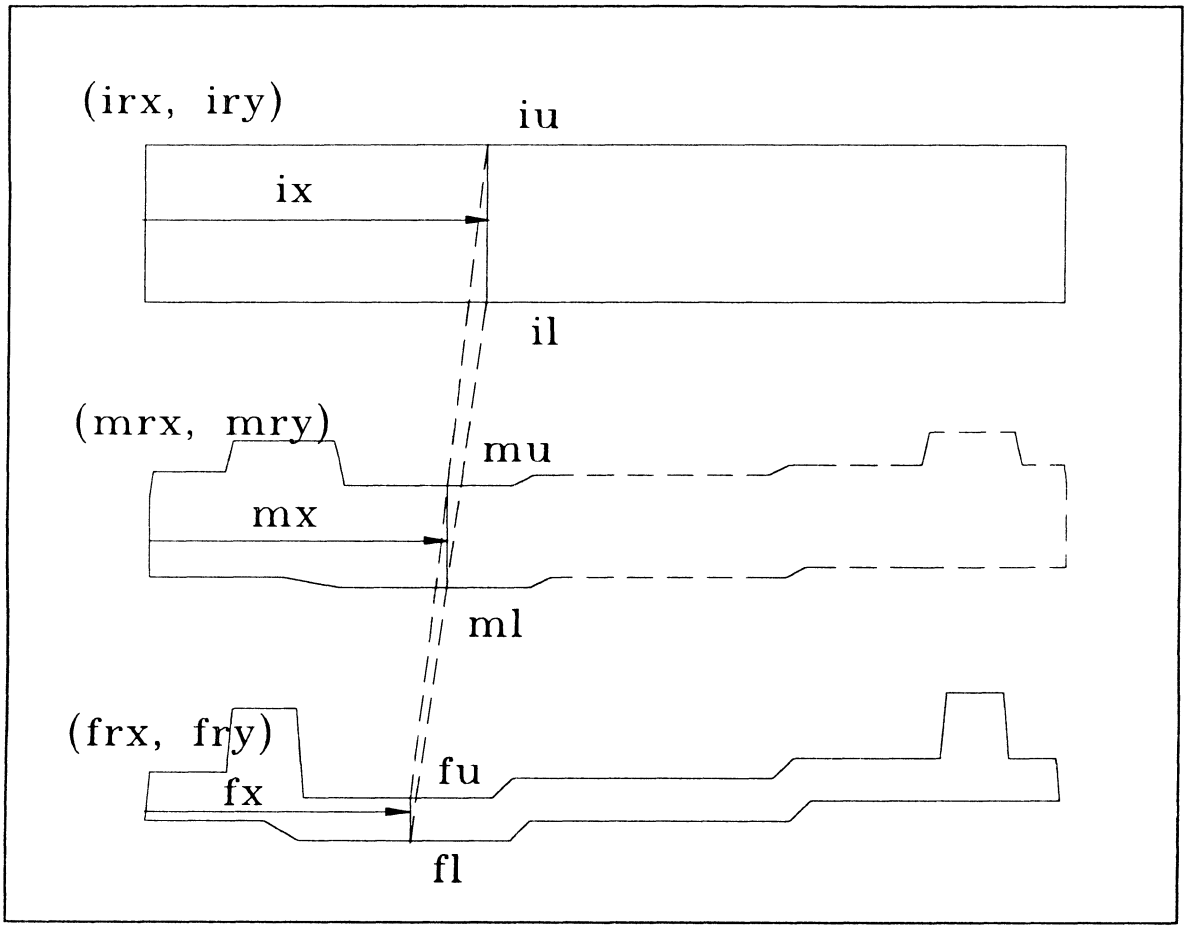


Figure 4.9 Generation of preform by mapping

4.3 Subroutines

As noted the LISP programs are all modular. Many subroutines are utilized by more than one program. For file management purposes, the subroutines are stored in six separate files. The file ALLUTIL.LSP contains utility subroutines that are used by all of the Process Design programs. The file LMUTIL.LSP contains utility subroutines used by Lateral Flow Mapping and Area Mapping. The other four files contain subroutines used only by the respective Process Design tools. All of the AutoLISP files are loaded when ROLLCAD is started. Table 4.1 lists the routines contained in each file. A

detailed description of each subroutine follows. The complete source code is in appendix 3.

MAP	ALLUTIL	LMUTIL	LATERAL	DRAFTANG	THICKMAP
CALLMAP	ARC2LINE	AREALEFT	LATCALC	C:DA	LINTER
FINDMAX	INT	FIND-MINPT	C:LATERAL	REMOVE	MIDENTS
C:IS	MOD	IDENTS	SHAPESEL2	SORT	C:THICKMAP
MAP	PT=	READELEM	DT	INSERT	
MATCHAREA	PTONARC	STARTWITH	C:AUTOLINEUP		
REORDER	SHAPESEL	FINDYMAX	LINEUP		
C:MAP	X-MAX		DI		
SETMAP	X-MIN				
SHAPE-SELECT	Y-MAX				
SORT2	Y-MIN				
INTER-SHP					

Table 4.1: Process Design Files and Programs

arealeft

File:	LMUTIL.LSP
Purpose:	Calculates the area of the shape defined by the entities in entlist to the left of the x coordinate specified in the parameter XPOINT.
Called By:	dt, callmap, matcharea
Calls:	readelem
Arguments:	entlist, iupperent, ilowerent, xpoint

Data returned: area (real number)

The shape is passed to the program in the parameter **entlist** which is a list of the entity names of the entities in the shape, sorted as described in section 4.1. **iupperent** and **ilowerent** contain the position index of the entities intersected by a vertical line through **xpoint**. These values were previously determined by the routine **idents**.

The endpoints of the upper and lower entities are extracted (**ulend, urend, llend, lrend** for upper left, upper right, lower left, and lower right endpoints, respectively.) The intersections between the entities and a vertical line through **xpoint** are identified (**uipt, lipt** for upper intersection point, lower intersection point.) Lines are constructed from **ulend** to **uipt**, from **uipt** to **lipt**, and from **lipt** to **llend**. These entities, and all entities with indices less than **iupperent** or greater than **ilowerent** are made into a polyline. The AutoCAD **area** command is used to find the area of this region. The process is then undone using **undo**, and the area value **larea** is returned.

arc2line

File:	ALLUTIL.LSP
Purpose:	Approximates arcs as a series of line segments.
Called by:	c:da, c:thickmap, c:lateral, c:map
Calls:	pt=, ptonarc
Arguments:	entlist

Data Returned: new list of entity names, included line segment entities created to replace arcs.

The basic principle of **arc2line** is simple. The arc angle **dalpha** is divided into a number of segments **nsegs** by the command

```
(setq nsegs (int (+ (/ dalpha maxdel) 1)))
```

which insures that the angle of each of the segments will be less than the maximum defined included angle **maxdel**. Each segment angle **del** is then defined by the statement

```
(setq del (/ dalpha nsegs))
```

which breaks the angle into uniform increments. The endpoints of each of these arc segments is connected by a straight line, and the arc is erased.

Implementation becomes more difficult, because the endpoints of the adjacent segments must remain connected. Much of the program is devoted to insuring proper connectivity. The arc is always defined counterclockwise from the start angle to the end angle. The endpoint of the segments connecting the arc are used as the endpoints for the first and last line segments drawn.

c:autolineup

File:	LATERAL.LSP
Purpose:	Lines up two shapes vertically according to their vertical centers before running lateral flow mapping.
Called by:	Process Design Menu, AutoCAD command line
Calls:	shapesel, arc2line, x-max, x-min, y-max, y-min
Arguments:	*none*
Data Returned:	*none*

C:autolineup is defined as an AutoCAD command. It prompts the user to select two shapes, then finds the point midway between the largest and smallest x coordinate of each of these shapes. The AutoCAD **move** command is then used to align the midpoint of the second shape with the midpoint of the first.

callmap

File:	MAP.LSP
Purpose:	Calls the mapping routine to map the initial shape to the final shape, then the final shape to the initial shape.
Called By:	setmap
Calls:	findxmax, idents, arealeft, map

Arguments:	initshp, finlshp, initarea, finlarea
Data Returned:	a list of the form ((m1 n1) (m2 n2) (m3 n3) ...) where m*i* and n*i* are x coordinates of vertical lines through the first and second shape.

This subroutine compiles the list of x coordinates which is used to generate the intermediate shape mapped from two shapes. The lists of entity names in the first and second shape are passed to **callmap** by **setmap**. They undergo standard processing, then the **map** routine is called. **Map** returns a list of x coordinate pairs defining vertical lines through the first and second shape, respectively. **Map** is called a second time, with the second shape given first, and the first shape given second. This generates a second coordinate list. **Callmap** assembles the two lists into a single list, and eliminates duplicate data points, and returns this list to the calling program.

c:da

File:	DRAFTANG.LSP
Purpose:	Change the angle of drawing entities and modify or erase adjacent entities to create draft angles on part profiles.
Called by:	Process Design Menu, AutoCAD command line.
Calls:	shapesel, arc2line, pt=, remove, mod
Arguments:	*none*

Data Returned: *none*

C:da is defined as an AutoCAD command. It may be invoked from the AutoCAD command line or from the Process Design menu. The user is prompted to select a shape. The subroutine **shapesel** obtains the shape data from the screen and returns it to the variable **entlist**, which then undergoes standard processing. The draft angle variable **dangle** is set to a default value of 5°. The user is then prompted to select the first drawing entity to be modified (**vertent**), and then select one of the endpoints on that entity (**endpoint**). The user is given the opportunity to modify the default draft angle, or leave it at its previous value. Then a direction is indicated with a point on either side of the entity (**side**). The program identifies whether the starting or ending endpoint of the entity was selected. It then identifies the side of the entity on which the point **side** was selected. A long vector is created starting at **endpoint** at an angle **dangle** to the same side of the existing entity as **side**. The point **pt2** is identified 100 drawing units in this direction. No line entity is created, since the AutoCAD intersection command uses only points.

The program steps through **entlist** (forward or backward, depending on whether the point selected was the starting or ending endpoint of the entity) until an intersecting entity is found. If no intersecting entity (other than the immediately prior entity in **entlist**) is found, then the intersection with the next entity in **entlist** is calculated. The intersection point becomes the

new endpoint for both entities. If any entities are between **vertent** and the intersected entity, they are deleted by the function **remove**.

The program repeats until terminated by the user.

dl

File:	LATERAL.LSP
Purpose:	Prompts the user for two points and returns the distance between them.
Called by:	c:lineup
Calls:	*none*
Arguments:	*none*
Returns:	list containing (x,y,z) coordinates of second point relative to first.

The program simply uses the AutoLISP **getpoint** command twice. The second time it is called, the first point is given as an argument, and the value returned is a relative value.

dt

File:	LATERAL.LSP
Purpose:	Calculates the lateral flow function at the given x position.

Called by:	latcalc
Calls:	idents, arealeft
Arguments:	entlist1, entlist2, xpos, xmax-1, xmax-2
Returns	real value of lateral flow graph at given x position.

The value of the lateral flow function is given by the equation

$$Flow(x) = \int_0^x T_1(x)\,dx - \frac{A_1}{A_2}\int_0^x T_2(x)\,dx \qquad (4.5)$$

In this program, the first integral is calculated by the function

(arealeft entlist1 u l xpos)

and the second integral is calculated by the function

(arealeft entlist2 u l xpos)

where **entlist1** and **entlist2** are lists of entity names of the first and second shape, respectively. The area ratio **aratio**, $\frac{A_1}{A_2}$, is a global variable.

findxmax

File:	MAP.LSP
Purpose:	Returns the largest x coordinate in the shape
Called by:	callmap
Calls:	*none*
Arguments:	entlist

Returns: real x coordinate

Scans the x coordinate of the starting endpoint of every entity in **entlist** and returns the largest value found.

find-minpt

File:	LMUTIL.LSP
Purpose:	Identifies the entity with the smallest x coordinate of its starting endpoint.
Called by:	c:map
Calls:	*none*
Arguments:	entlist
Data Returned:	list of entities with smallest x coordinate.

Scans the x coordinate of the starting endpoint of every entity in **entlist** and returns the name of the entity with the smallest coordinate. If more than one entity is found, all are returned.

findymax

File:	LMUTIL.LSP
Purpose:	Identifies the entity in a list of entities with the largest y coordinate as a starting endpoint.
Called by:	c:map
Calls:	*none*
Arguments:	entlist
Data returned:	entity with largest y coordinate.

Scans the y coordinates of the entities in the list returned by find-minpt, and returns the entity with the largest value. When used with find-minpt this function identifies the left top corner of the shape, with the horizontal position taking precedence over the vertical position.

idents

File:	LMUTIL.LSP
Purpose:	Identifies the lower and upper entities intersecting a vertical line through a point of a given x coordinate.
Called by:	dt, callmap, matcharea, intershp
Calls:	*none*
Arguments:	entlist, xpoint

Data Returned: list of 2 entity names identifying the first and last entity in entlist which intersect a vertical line through xpoint. *nil* if no intersection is found.

Starting with the first entity in entlist, the x coordinates are checked to ensure that they are not close together but non-vertical

(and (< (abs (- x1 x2) 5.0E-4) (/= x1 x2)).

Lines that are close to vertical will cause problems with the AutoLISP intersection function. If any entities which meet this criteria are found, the x coordinate of the first endpoint is replaced with the x coordinate of the second endpoint, and the second endpoint of the connecting endpoint is adjusted similarly.

The entity is not vertical, it is tested for intersection with a vertical line through **xpoint**. If an intersection is found, the entity is added to the list **ents**. After all entities in **entlist** have been tested, the first and last entities in **ents** are returned to the calling program.

inter-shp

File:	MAP.LSP
Purpose:	Generates the intermediate shape from a list of corresponding points on the initial and final shapes.
Called By:	c:is
Calls:	idents, readelem, sort2
Arguments:	initshp, finlshp, maplist
Data returned:	*none*

The function uses the data returned by callmap to generate the intermediate shape. Reference points are identified on the left end of the initial and final shapes, and the user supplies a reference point for the left end of the mapped shape. The data in maplist is the corresponding x coordinates of vertical lines through the initial and final shapes. The coordinates are global. The coordinates of the shapes are all converted to local with the reference point of each shape as the origin. The corresponding coordinates for the initial and final shapes are identified. A weighted average of the corresponding coordinates is calculated using a user-supplied weighting factor. (equations 4.7, 4.8, and 4.9) The calculated coordinates are linked with line segments, yielding the mapped intermediate shape. (see figure 4.10).

$$m_x = i_x + K(f_x - i_x) \quad (4.7)$$

$$m_u = i_u + K(f_u - f_l) \quad (4.8)$$

$$m_l = i_l + K(f_l - i_l) \quad (4.9)$$

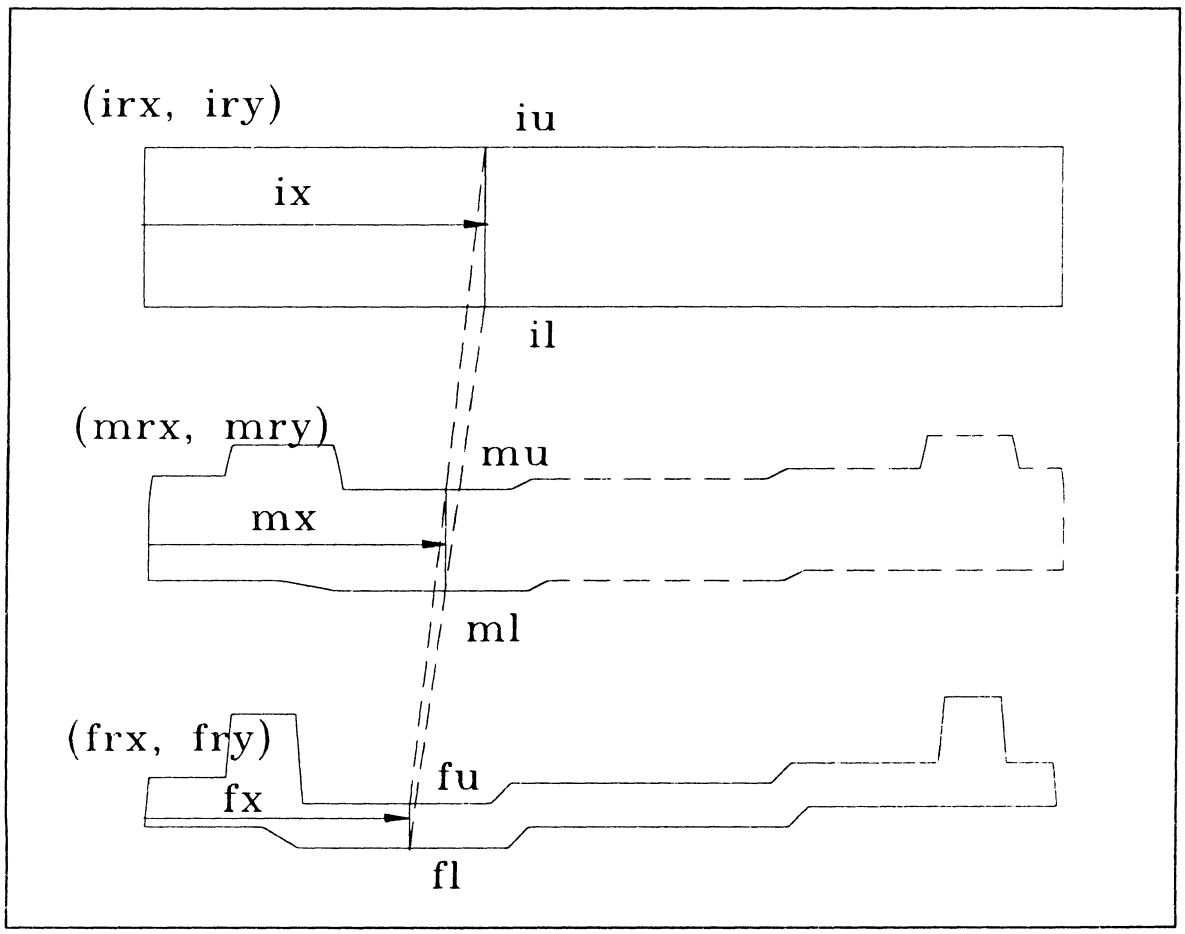


Figure 4.10: Generation of preform by mapping

c:is

File:	MAP.LSP
Purpose:	Call the intermediate shape function
Called by:	Process Design Menu, AutoCAD command line.

Calls:	inter-shp
Arguments:	*none*
Data Returned:	*none*

This function is defined as an AutoCAD command. Its sole purpose is to invoke the AutoLISP defined function **intershp**.

latcalc

File:	LATERAL.LSP
Purpose:	Calls dt to calculate the lateral flow, and plots the lateral flow map.
Called by:	lateral
Calls:	dt
Arguments:	l1, l2, scaley, res
Data Returned:	*none*

This function calculates the increments at which lateral flow is to be calculated. It then calls **dt** top perform the lateral flow calculations at each of these points, and draws the lateral flow graph based on these results.

c:lateral

File:	LATERAL.LSP
Purpose:	AutoCAD command to initiate lateral flow mapping and control user interaction.
Called By:	Process Design Menu, AutoCAD command line.
Calls:	shapesel, are2line, startwith, x-min x-max, latcalc
Arguments:	*none*
Data returned:	*none*

This AutoCAD command prompts the user to select two shapes. The shapes undergo standard processing. The width of the two shapes is calculated, and the width of the lateral flow plot is set to the wider shape. The ordinate line of the graph is drawn, then **latcalc** is called to draw the graph. The text is scaled and drawn in.

lineup

File:	LATERAL.LSP
Purpose:	Lines up one shape with respect to another shape.
Called By:	user
Calls:	di
Arguments:	*none*
Data returned:	*none*

The user selects two shapes. The function **di** is called to obtain the offset distance, and the AutoCAD **move** command is used to shift the shape.

linter

File:	THICKMAP.LSP
Purpose:	Returns the intersection points of a vertical line at a given x position and the entities that it intersects.
Called By:	c:thickmap
Calls:	midents
Arguments:	xp
Data returned:	intersection points

The function **midents** is called to identify the entities intersected by a vertical line through **xp**. The intersection points are then identified using the AutoLISP **inters** function.

map

File:	MAP.LSP
Purpose:	Given an initial and final shape, map scans the starting shape from left to right. Each time an entity endpoint is encountered, the area to the left of a vertical line through that point is calculated.

	The x position of a vertical line through the final shape is then calculated so the area ratios to the left of the lines are equal to the area ratio of the total shapes.
Called by:	callmap
Calls:	readelem, arealeft, matcharea.
Arguments:	startshp, endshp, startarea, endarea
Data returned:	list containing two lists. The first is a list of x coordinates of vertical lines. The second is a list of x coordinates of corresponding vertical lines in the final shape.

The arguments startshp and endshp are a list of entity names of the lines in the initial and final shapes, respectively. The area ratio of the final area to the initial area is calculated (**ratio**). The lists have already been sorted so the starting endpoint of the first entity is at the left end of the shape. The name of the first entity in the list is assigned to **entlo** and the name of the last entity in the list is assigned to **enthi**. The rightmost endpoint of each of these entities is found, and the smaller one is used for area the area calculation. The area calculation in performed by **arealeft.** The second area is calculated from the first, and **matcharea** is called with this area as a parameter to find the position of a vertical line through the second shape with this area. The x values are appended to their respective lists, and the process repeats with the next entity. The process is illustrated in figure 4.11.

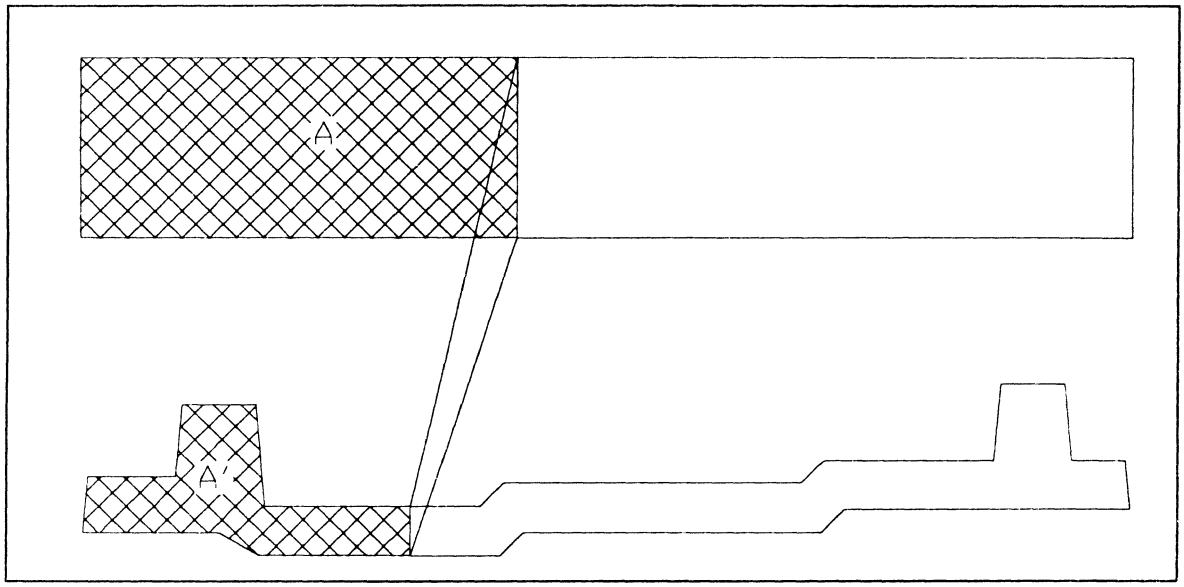


Figure 4.11 Illustration of matching area ration as calculated in map

The entity endpoints are used as calculation points because area variation between these points is necessarily linear.

c:map

File:	MAP.LSP
Purpose:	AutoCAD command to initiate mapping. Performs user interaction, then transfers control to setmap.
Called by:	Process Design Menu, AutoCAD command line.
Calls:	shape-select, arc2line, find-minpt, find-ymax, startwith, setmap
Arguments:	*none*
Data returned:	*none*

The user is prompted to select an initial and final shape for mapping. The shapes undergo standard processing and the setmap function is called. This program calls callmap, which controls the mapping procedure.

matcharea

File:	MAP.LSP
Purpose:	Finds the x position of a vertical line with the specified area to the left of it.
Called by:	map
Calls:	idents, arealeft
Arguments:	shape, targetarea, arealeft
Data returned:	real x position of vertical line through shape.

The function utilizes an iterative approach to find the position of the line in the shape. The position of **x** is estimated from three global variables which are set by the function, but are not lost between consecutive calls of the function. **oldx** stores the previous x value calculated by the function. **oldarea** stores the previous area which was associated with **oldx**. **dxda** is the incremental increase in **x** divided by the incremental increase in **targetarea** over the last two steps. Default values are set for the first execution of this function. The new **x** position is estimated by the formula

$$x_{new} = x_{old} + \frac{dx}{da}(area_{old} - area_{target}) \qquad (4.10)$$

The area to the left of this **x** is calculated using **arealeft**. If the areas are not within an acceptable tolerance, a new **dxda** is calculated by the formula

$$\frac{dx}{da} = \frac{(x_{new} - x_{old})}{(area_{new} - area_{old})} \qquad (4.11)$$

The process is repeated until an two consecutive steps yield results equal within an acceptable tolerance.

midents

File:	THICKMAP.LSP
Purpose:	Identify the indices of lower and upper entities intersected by a vertical line through xpoint.
Called by:	thickmap, linter
Calls:	*none*
Arguments:	xp
Data returned:	list of two entities intersected by a vertical line through xp.

Midents is an improved version of **idents.** The entities are scanned both from the top and bottom of the shape. The left most entities intersected by the line are returned if more than two are identified.

pt=

File:	ALLUTIL.LSP
Purpose:	Tests two points for equality within 0.0005"
Called By:	c:da, arc2line
Calls:	*none*
Arguments:	point1, point2
Data returned:	logical True or nil

The function tests the equality of the x and y coordinates of the points. Both the x and y coordinates are equal within 0.0005, the function returns true.

ptonarc

File:	ALLUTIL.LSP
Purpose:	Finds the coordinates of a point on an arc in the xy plane given the center, radius, and angle on the arc from the positive x axis.

Called by:	arc2line
Calls:	*none*
Arguments:	center, radius, angle
Data returned:	list containing x and y coordinates of the point on the arc with the specified polar coordinates.

The coordinates are calculated using the sine and cosine functions.

readelem

File:	LMUTIL.LSP
Purpose:	Reads an element from a list "l" which is the "index" member of the list.
Called by:	map, arealeft
Calls:	*none*
Arguments:	l, index
Data returned:	value of the specified element in the list.

The function sets a counter, and steps through the list until the given index is reached.

reorder

File:	MAP.LSP

Purpose:	Reorders the entities in shape into consecutive order.
Called By:	Shape-Select
Calls:	*none*
Arguments:	shape, layername, layn
Data returned:	sorted list, and area of the shape.

The shape is made into a polyline and then exploded. The entities resulting are assembled into the list to be passed to the next routine.

remove

File:	DRAFTANG.LSP
Purpose:	removes the specified element from the list, and reassembles the list.
Called by:	c:da
Calls:	*none*
Arguments:	x, entlist
Data returned:	list with the specified element removed.

The function steps through the list. At each step, the index of the entity is checked. If the index does not match **x**, the entity is removed from **entlist** and appended to **templist**. If the index does match, the entity is ignored, and the process continues to the next entity.

setmap

File:	MAP.LSP
Purpose:	calls the function callmap, which controls the mapping process.
Called by:	c:map
Calls:	callmap
Arguments:	*none*
Data returned:	*none*

This is simply a calling routine. It was implemented to bypass a minor glitch in the software which would not allow the other functions to be called directly.

shapesel

File:	ALLUTIL.LSP
Purpose:	Prompt the user to window the shape, then return and ordered list of the windowed shape.
Called by:	c:da, c:thickmap, c:lateral
Calls:	*none*
Arguments:	*none*
Data returned:	list of entity names of the entities in the shape, the name of the selection set chosen.

After the calling program prompts the user to select the shape, this function performs the window operations and returns a selection set. The program stores information regarding the color, layer, and other attributes of the shape. It then checks to ensure that the entity selected is not a circle or a polyline. The shape is moved to a temporary layer for easier manipulation. The shape is converted to a polyline, then exploded. The new elements are appended to **pshape**. The original attributes are reset and **pshape** is returned.

sort2

File:	MAP.LSP
Purpose:	sort two values in ascending order
Called by:	intershp
Calls:	*none*
Arguments:	i,j
Data returned:	i and j sorted in ascending order.

The function checks for **j < i**. If this is true, they are swapped and returned to the calling program. Otherwise they are returned directly to the calling program.

startwith

File:	LMUTIL.LSP
Purpose:	Receives a list of entity names in a shape sorted into consecutive order, and the name of the entity to start the output list. Shifts the list to start with the specified entity.
Called by:	c:map, c:lateral
Calls:	*none*
Arguments:	entlist, entname
Data returned:	list containing the same elements in entlist, starting with entname.

The function takes elements one by one from the beginning of **entlist**, checks it against **entname**, and if it does not match, appends it to the temporary **listb**. When **entname** is found, **listb** is appended to the end of the remainder of **entlist**.

c:thickmap

File:	THICKMAP.LSP
Purpose:	AutoCAD command to plot the thickness profile of a shape.
Called by:	Process Design Menu, AutoCAD command line.

Calls:	shapesel, arc2line, x-min, x-max, y-min, y-max, linter
Arguments:	*none*
Data returned:	*none*

The function prompts the user to window a shape, which undergoes standard processing. the minimum and maximum coordinates are identified. The user is prompted for a starting point of the thickness map, and a baseline is drawn from this point.

The shape is scanned from left to right. At each endpoint, the vertical distance from that endpoint to the line segment directly above or below it is calculated. This distance is then used to plot the thickness curve above the baseline.

x-max, x-min, y-max, y-min

File: ALLUTIL.LSP

Purpose: Determine the extreme x and y coordinates of a shape

Called by: c:thickmap, c:lateral

Calls: *none*

Arguments: entlist

Data returned: real number x or y coordinate of the maximum or minimum point, as described by the function name.

The operation of these functions is identical except for the coordinate tested and the "greater than" or "less than" test condition. The starting endpoint of the first entity in the list is initially assigned to the variable **xmax, xmin, ymax** or **ymin**. The starting endpoint of each successive entity is tested against this value. If it is larger (for max functions) or smaller (for min functions) than the previous value, the value is replaced. After all entities are tested, the extreme value is returned.

5.0 CONCLUSION

5.1 Results

The goal of this work was to develop software tools to assist the roll designer in designing and evaluating a rolling sequence for complex profiles. Tools were developed to automatically add a machining envelope and draft angles, and to generate an intermediate preform from a starting and final shape. Evaluation tools developed in this project will display the thickness distribution of a part and map the lateral material movement required to roll a target shape from a given profile.

The software developed is currently in use in industry. There has been a good correlation between results obtained from finite element analysis and experimentation, and the results expected from using the evaluation tools. In most cases, it was possible to attain satisfactory roll fill in the first run. In almost all cases where fill was not achieved, the region which did not fill was identified as questionable in the preliminary design stage.

5.2 Discussion

The software models, while useful, still require a skilled and experienced roll designer to evaluate their output. The information presented by thickness mapping and lateral flow mapping is accurate, but its effect on

roll fill is strictly qualitative. There is no known way to quantify the effect of increased lateral flow or the amount of thickness variation. These effects are relative, and simply provide another tool which, with experience, will become increasingly useful to the designer.

5.3 Future Work

As noted, at this time there is no quantifiable relationship between lateral flow or thickness distribution and profile fill. A study relating various shape factors with the evaluation tools would be useful. Clearly, the relationships exist. We now have the tools to observe the relationships, but cannot yet predict them.

An improved quantitative understanding of the relationship between profile fill and such characteristics as lateral flow and thickness mapping may lead to further automation of the design process by defining limits of lateral flow, using these limits to develop new shapes.

REFERENCES

1. Z. Wusatowski, *Fundamentals of Rolling* Pergamon Press, Oxford. 1969.
2. A.G. Mamalis, J.B. Hawkyard and W. Johnson, "Cavity Formation in Profiled Ring Rolling", *International Journal of Mechanical Sciences*, 1975 vol 17. pp. 669-672.
3. J.T. Winship, "Cold Ring Rolling Warms Up", *American Machinist*, January, 1976.
4. J.B. Hawkyard and P.M. Ingham, "An Investigation into Profile Ring Rolling", *Proceedings of the 1st International Conference on Rotary Metal-working Processes*, London, UK, November 20-22, 1979, pp. 309-320.
5. W. Johnson and A.G. Mamalis, "Rolling of Rings", *International Metals Review*, 1976, Vol.4 pp. 137-148.
6. D.Y. Yang, J.S. Ryoo, J.C. Choi, and W. Johnson, "Analysis of Roll Torque in Profile Ring Rolling of L-Sections", *Proceedings of the 21st International Machine Tool Design and Research Conference*, 1981, pp. 69-74.
7. K.F. Hutcheon and J.B. Hawkyard, "Production of Jointless Light Aluminum Cycle Wheel Rims by Ring Rolling", *Proceedings of the 24th International Machine Toll Design and Research Conference*, 1983, pp. 13-18.

8. G. Moussa and J.B. Hawkyard, "Studies of Profile Development and Roll Force in Profile Ring Rolling", *Proceedings of the 3rd International Conference on Metal Working Processes,* Kyoto 1984.

9. M.M. Vyas, C.J. Romberger, "Development of Roll Passes for Special Structural Sections Using Computer-Aided Design Techniques. *2nd International Conference on Steel Rolling.* Duesseldorf, Germany. 1984.

10. T. Hirai, T. Katayama, S. Kusada and J.B. Hawkyard, "Design Concept of Projection Forming in Profile Ring Rolling." *original source unknown.*

11. G. Moussa and J.B. Hawkyard, "Investigation into the Multi-stage Ring Rolling of Alloy Bicycle Wheel Rims", *International Journal of Mechanical Sciences,* 1986, vol 28, pp. 841-851.

12. U. Koppers, H. Wiegels, P. Dreinhoff, J. Henkel, and R. Kopp, "Methods Applied to Reduce Material and Energy Expenditures in Ring Rolling", *Stahl U. Eisen,* 106, July 1986, pp. 789-795.

13. P. Drienhoff and R. Kopp, "Methods for Economic Use of Material and Energy in Ring Rolling", *Institut fuer Bildsame Formgeburg,* RWTH, Aachen, Germany.

14. P. Boucly, J. Oudin and Y. Ravalard "New Approach for Predicting Ring Rolling Procedure", *Proceedings of the 2nd International Conference on Technology of Plasticity, Advanced Technology of Plasticity,* 1987, vol. 11.

15. F. Burnett, "Ring Rolled Technology for Aircraft Engine Parts", *Proceedings of the 1st International Conference on Ring Rolling*, Ohio University, 1987, Athens, Ohio.

16. E. Doege and M. Aboutour, "Simulation of Ring Rolling Process", *Proceedings of the 2nd International Conference on Technology of Plasticity, Advanced Technology of Plasticity*, 1987, vol. 11.

17. D.W. Kim and H.Y. Kim, "Preform Design for Axisymmetric Closed-Die Forging by the Upper Bound Elemental Technique (UBET)", *Computer Modeling and Simulation of Manufacturing Processes.* Winter Annual Meeting of the American Society of Mechanical Engineers. pp. 155-164, 1990.

18. Y-K. Hu and W.K. Liu, "Simulation of Ring rolling Process By Arbitrary Lagrangian Eulerian Finite Element Method." *Computer Modeling and Simulation of Manufacturing Processes.* Winter Annual Meeting of the American Society of Mechanical Engineers. pp. 225-240. 1990

19. N. Kim, S Machida, S. Kobayashi, "Ring Rolling Process Simulation by the Three Dimensional Finite Element Method" *International Journal of Machine Tools and Manufacture* v30 n4 pp569-577, 1990.

20. V.B.S. Rachakonda, "Computer Aided Design of Ring Rolling Process", Masters Thesis, Ohio University, Athens, OH. 1990.

21. B.S. Kang, S. Kobayashi, "Preform design in Ring Rolling Process by the Three Dimensional Finite Element Method." *International Journal of Machine Tools and Manufacturing* v.31 n.1 pp 139-151, 1991.

22. Y.H. Hahn and D.Y. Yang, "UBET Analysis of Roll Torque and Profile Formation During the Profile Ring-Rolling of Rings Having Rectangular Protrusions." *Journal of Materials Processing Technology* v26, n3 pp267-280, 1991.

23. S.G. Xu, J.C. Lian, J.B. Hawkyard, "Simulation of Ring Rolling using a Rigid-Plastic Finite Element Model." *International Journal of Mechanical Sciences* v33 n5 1991 pp 393-401. 1991.

24. E. Eruk and R. Shivpuri, "A Summary of Ring Rolling Technology--I. Recent Trends in Machines, Processes, and Production Lines, *International Journal of Machine Tools and Manufacturing.* v32 n3 p379-398, 1992.

25. E. Eruk and R. Shivpuri, "A Summary of Ring Rolling Technology--I. Recent trends in Process Modeling, Simulation, Planning, and Control, *International Journal of Machine Tools and Manufacturing.* v32 n3 p 399-413, 1992.

26. F. Masri, *Using AutoCAD in the Analysis and Design of Rolls.* Masters Project Report. Ohio University, Athens, OH, 1992.

27. J.S. Gunasekera, A.F. Ali, A. Rathinavel, C.E. Fischer, *The Development of a Computer Model to Assist in the Design of Compression Rolls.* Final Report to United Technologies, Pratt & Whitney. Ohio University. 1993

APPENDIX 1: PROCESS DESIGN SECTION FROM ROLLCAD USERS MANUAL

The following pages are taken directly from the ROLLCAD version 3.01 user's manual. They describe the procedure for running all of the process design programs described in this thesis. Execution is within the AutoCAD drawing editor.

PROCESS DESIGN

Selecting **PROCESS DESIGN** from the **ROLLING** menu accesses the level 2 **PROCESS DESIGN** menu. The options in this file are geometric manipulation and analysis tools provided for assisting in the development of roll designs. The options in this file are:

- Machining Envelope
- Add Draft Angles
- Thickness Mapping
- Auto Lineup
- Lateral Flow Mapping
- Area Mapping
- Generate Intermediate Shape
- Fillet / Edge Break

The purpose and use of each of these options is described in the following sections. They should be used after the desired part profile is drawn.

The recommended procedure for designing rolls with ROLLCAD is:

1 - Draw the desired final part profile in AutoCAD

2 - Add the machining envelope to the part

3 - Add draft angles to the part

4 - If the profile is complex, run **thickness mapping** to get a better idea of the required thickness distribution in the part.

5 - Develop a plan for rolling the part. This may include one or more preforms, bending, or a welded assembly as discussed in the introduction. For simple profiles, it may be possible to form the part directly from rectangular stock

6 - Using the thickness map as a guide, draw a preform shape with a similar thickness distribution, but with smooth transitions which will fill more readily than sharp corners.

7 - Use **lateral flow mapping** to gauge the quality of the preform. You may wish to experiment with several different preforms to minimize the peaks in the lateral flow graph.

8 - If one preform does not prove adequate, a second preform may be generated using **area mapping** and **generate intermediate shape**. After this shape has been generated, **lateral flow mapping** can once again be used to gauge preform quality. The intermediate shape can be modified using the **stretch** command from the AutoCAD **Modify** menu.

9 - After suitable shapes have been drawn, add fillets and edge breaks to all the shapes.

10 - Draw the roll profiles using the shapes generated.

11 - Create a **PATRAN** input file and analyze the roll sequence using a finite element analysis package such as ANTARES.

Notes on Program Execution

For these options to run properly, the shape must be drawn in the XY plane (Z = 0 at all points). All line elements must be connected. (Segments whose endpoints are close but not identical may cause problem with some modules.) The best way to achieve this is to draw line segments one after another as a chain, or to use endpoint snaps when constructing the profile.

In all of these options, arcs are approximated as a series of line segments determined by chords of the arc. The arc length of the chord is less than 20°.

If a program does not run the first time on a selected shape, try again making sure to select the shape properly - i.e. make sure to enclose all the elements of the desired shape within the window.

Some of the programs do not run properly if the shape selected is a polyline. If this happens, the program will stop, and you will see this message:

Close/Join/Width/Edit vertex/Fit curve/Spline curve/Decurve/Undo/eXit<x>:

Press <enter> to eXit the polyline option, then **explode** both shapes and try the command again.

If a program does not seem to do anything for a few minutes, chances are that it hung-up. To interrupt the program, just type <ctrl-c> and start over. Note that if you press any key other than <ctrl-c>, the computer hangs-up and the only way to regain control is to reboot the computer.

If you suspect a bug in the program, immediately save the current drawing file, and send it, with a detailed description of the problem encountered and the circumstances under which it arose.

Any shape generated by a process design program will be drawn on the current layer. If you want to place the generated drawing on a separate layer, make that layer the current layer before running the particular program.

Throughout ROLLCAD several layers are created for temporary use. All layers begin with the letters OU. You should avoid drawing anything on these layers.

Machining Envelope

Rolling does not necessarily offer the degree of precision and tolerance control necessary for the final product. Thus, the goal of profile rolling is to form a near net shape part which will undergo machining operations after rolling is completed. To ensure that enough material exists outside the net shape part profile, a machining envelope is added around the part. The **Machining Envelope** command utilizes the AutoCAD "OFFSET" command to ease modification of the previously drawn part profile. **Machining Envelope** is selected from the **PROCESS DESIGN** menu. Execution proceeds as follows:

Pick a point on the object: AN ARBITRARY POINT ON THE PROFILE TO IDENTIFY THE SHAPE

Enclose the object with a window: WINDOW MODE IS SELECTED AUTOMATICALLY

Enter the desired machining envelope in inches: THE MODIFIED PROFILE WILL BE DRAWN.

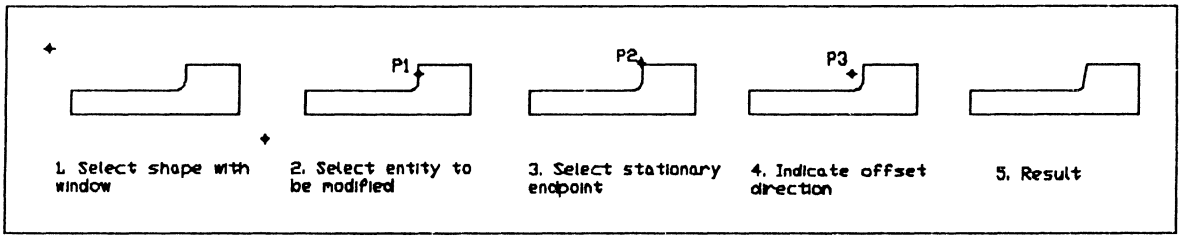
IF IT IS ACCEPTABLE, YOU SHOULD ERASE THE INITIAL PROFILE. IF NOT, ERASE THE NEW PROFILE AND TRY AGAIN WITH A NEW ENVELOPE DIMENSION.

Add Draft Angles

In any roll design, vertical surfaces in the roll gap must have a slight relief angle to keep the part being rolled from jamming in the roll. These relief angles are referred to as draft angles. The **Add Draft Angles** utility has been developed to ease the modification of part profile drawings to include draft angles. Since draft angles may significantly increase the cross sectional area of a profile, they should be added before any analysis routines are run on the shape.

Add Draft Angles may be selected from the **PROCESS DESIGN** menu, or by typing

Command: **da**



Select the desired shape with a window: WINDOW THE SHAPE TO BE MODIFIED. WINDOW MODE IS SELECTED AUTOMATICALLY

Select the entity to be modified, or ENTER to quit: PICK A POINT ON THE VERTICAL EDGE TO BE MOVED. PRESS ENTER OR THE RIGHT MOUSE BUTTON TO EXIT THE UTILITY

Select the stationary endpoint: SELECT ONE OF THE ENDPOINTS OF THE ENTITY JUST SELECTED ENDPOINT SNAP MODE IS AUTOMATICALLY SELECTED. IF THE POINT SELECTED IS NOT ON THE ENTITY, AN ERROR MESSAGE WILL RESULT. BE SURE THAT YOU ARE SELECTING A SINGLE LINE AND NOT TWO CONNECTED LINES.

Enter the angle (degrees), or press ENTER or the right mouse button to accept default <default angle> : THE DRAFT ANGLE IS SPECIFIED FROM VERTICAL, NOT FROM THE CURRENT ENTITY ANGLE. THE DEFAULT ANGLE IS THE LAST ANGLE USED, OR 5.0 FOR THE FIRST RUN.

Indicate the offset direction with a point left or right of the line :

DRAWING WILL BE
MODIFIED. AND THE SEQUENCE REPEATS.

Note: Due to the way arcs are handled internally, attempts to modify a line which intersects an arc may produce unexpected results. If this occurs, exit the utility and use the **U** (undo) command to step back through the modification.

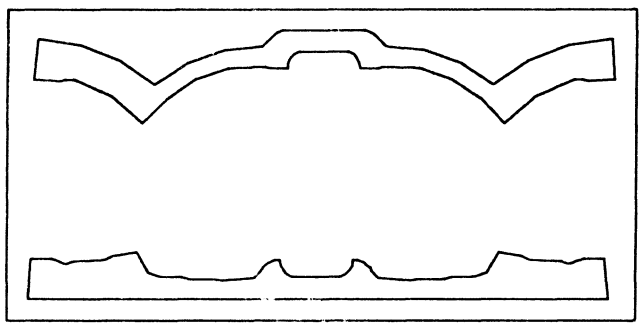
Thickness Mapping

As noted above, a profile has the best chance of filling if the material distribution in the preform closely matches the material distribution in the final shape. In some parts with complex profiles, the thickness distribution may not immediately be apparent. For these situations, a thickness mapping utility has been developed to plot the material distribution across the part profile.

Thickness Mapping is selected from the **PROCESS DESIGN** menu. Execution then proceeds as follows:

Select the desired shape with a window: WINDOW THE SHAPE TO BE MODIFIED. WINDOW MODE IS SELECTED AUTOMATICALLY

Select starting point of thickness map: SHAPE WILL BE DRAWN TO THE RIGHT OF THE INDICATED POINT. THE MAPPED SECTION WILL BE THE SAME WIDTH AS THE PART.



Lateral Flow Mapping and **Autolineup**

While thickness mapping provides a tool for analyzing the material distribution of a single part, Lateral Flow Mapping provides a tool for comparing the material distribution of consecutive shapes. Specifically, the graph produced gives a relative indication of the lateral flow required to form the second shape from the first shape. The height of the graph at each section indicates how far material from that section in the first shape must move laterally to form the second shape. Thus, a sequence of shapes which produces a flatter curve is more likely to fill than one with higher peaks in the graph.

It is important to note that this is only a preliminary analysis tool. It must be weighted against other factors affecting lateral flow, which must be determined by experience. It is most useful in fine tuning preforms to optimize thickness and height of certain sections. This is illustrated in figure 14 where it may be seen that preform 2 requires less lateral flow than preform 1 to fill the desired profile.

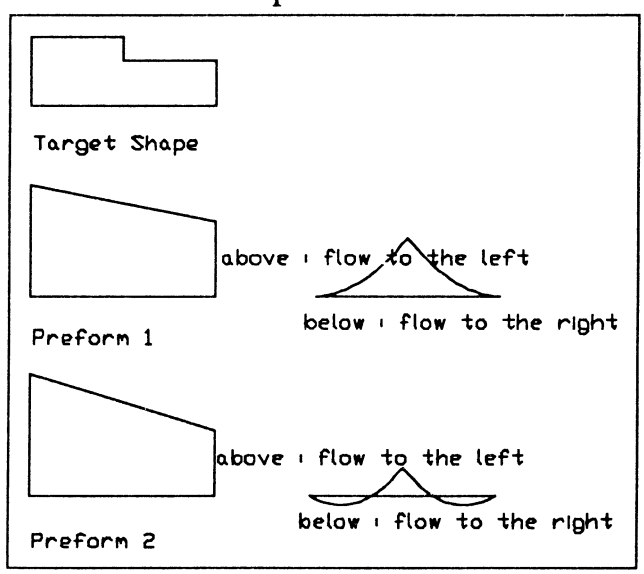


Figure 4.14 Using Lateral Flow Mapping to compare preform shapes.

Since the material flow in the roll gap varies with the position of the preform as it is inserted, **Lateral flow mapping** must account for this positioning. Hence, the positioning of the two shapes relative to each other

is important in this utility. For symmetric shapes, a utility named **Autolineup** has been provided to insure this positioning is correct. For non-symmetric shapes, you must carefully locate the shapes above and below each other.

If you are using symmetric shapes, you should run **Autolineup** before **Lateral flow mapping**. It is called from the **PROCESS DESIGN** menu, and runs as follows:

Select the shape to be moved with a window: THE CENTER OF THIS SHAPE WILL BE LINED UP WITH THE CENTER OF THE NEXT SHAPE SELECTED (THE REFERENCE SHAPE).

Select the reference shape with a window: AFTER THIS SHAPE IS SELECTED, THE FIRST SHAPE IS ALIGNED, AND THE UTILITY IS EXITED. YOU MAY NOW PROCEED TO **LATERAL FLOW MAPPING**

Lateral Flow Mapping is selected from the **PROCESS DESIGN** menu, or by typing

Command: **lateral**

Execution proceeds as follows:

Select the initial shape with a window: WINDOW THE DESIRED PREFORM SHAPE. WINDOW MODE IS SELECTED AUTOMATICALLY

Select the final shape with a window: WINDOW THE DESIRED TARGET SHAPE

Type in the y coordinate scaling factor of the graph <10>: FOR MOST PLOTS, THE GRAPH IS VERY FLAT. IT IS NECESSARY TO MAGNIFY THE Y SCALE OF THE GRAPH TO OBTAIN USEABLE RESULTS

Type in the graph resolution (10 - 30) <20 default>: CALCULATIONS ARE MADE AT UNIFORM INTERVALS ALONG THE SHAPES. MORE INTERVALS PROVIDE INCREASED ACCURACY BUT INCREASE EXECUTION TIME. 20 IS SUITABLE FOR MOST SHAPES. PLOTS WHICH APPEAR AS JAGGED LINES INSTEAD OF SMOOTH CURVES MAY BENEFIT FROM MORE CALCULATION INCREMENTS.

Area Mapping
and
Generate Intermediate Shape

In some situations involving complex shapes, the difference between the preform and the final shape may be too great to obtain fill even with proper material distribution in the preform. For these cases, you may find it helpful to first form a shape which is between the preform and final shape. **Area Mapping** performs a mathematical analysis of the preform and final shape you select and stores the data. **Generate Intermediate Shape** uses the data generated in **Area Mapping** to produce an average of the two shapes. You specify the weighing of each shape as the mapping ratio factor, with 0 being the initial shape and 1 being the final shape. Thus 0.5 is half way between the two shapes.

Area Mapping must be executed once before **Generate Intermediate Shape**. After that, **Generate Intermediate Shape** may be called repeatedly to generate shapes with different mapping ratios. To work with different preform or final shapes, execute **Area Mapping** again.

IMPORTANT NOTE : **Area Mapping** processes a large number of elements, and execution time may be very slow. Due to the nature of the AutoCAD database, it is necessary to delete all "undo" information while running **Area Mapping**. This means that changes to the drawing made before **Area Mapping** runs cannot be changed using the **undo** command afterwards. Also, execution time may be significantly enhanced by **end**ing the drawing and re-entering it immediately before executing **Area Mapping**. This is especially true if the computer you are using begins near continuous hard disk access during execution. If this occurs, execution time may well jump from a matter of minutes to a matter of hours. The only practical solution is to press <ctrl-c>, **end** the drawing, and start over.

Both **Area Mapping** and **Generate Intermediate Shape** are accessed from the **PROCESS DESIGN** menu. Execution Proceeds as follows:

SELECT **AREA MAPPING**

Select initial shape : WINDOW THE PREFORM SHAPE. WINDOW MODE IS SELECTED AUTOMATICALLY

Select final shape : WINDOW THE FINAL SHAPE. AGAIN, WINDOW MODE IS SELECTED AUTOMATICALLY.

THE PROGRAM WILL NOW RUN FOR ANYWHERE FROM A FEW SECONDS TO AROUND 10 OR 15 MINUTES. DEPENDING ON THE COMPLEXITY OF THE SHAPE AND THE SPEED OF THE COMPUTER BEING USED. IT MAY BE INTERRUPTED AT ANY TIME BY PRESSING <CTRL-C>. IF THE DRAWING IS ALTERED, IT MAY BE CORRECTED BY USING **UNDO BACK**.

SELECT **GENERATE INTERMEDIATE SHAPE**

Enter the mapping ratio factor (0 to 1) AS DISCUSSED ABOVE. NUMBERS OUTSIDE THE RANGE ARE NOT ACCEPTED.

Fillet / Edge Breaks

The **Fillet / Edge Breaks** command is identical to the AutoCAD **Fillet** command found under the **Modify** menu. It is included here for convenience. You may refer to the AutoCAD reference manual for a detailed description of the function. The command is invoked by selecting **Fillet / Edge Breaks** from the **PROCESS DESIGN** menu, or **Fillet** from the **Modify** menu, or by typing **fillet** at the Command: prompt. AutoCAD first responds with three options:

Polyline/Radius/<Select two objects>:

If no other option is specified, AutoCAD expects you to point to two lines or arcs, which it will connect with a smoothly fitted arc of the specified radius. To specify this radius,
type "r" at the Polyline/Radius/<Select two objects>: prompt. AutoCAD then prompts:

Enter fillet radius <current>:

You may press <enter> to maintain the current radius, or specify a new one.

You also have the option of filleting an entire polyline at once. First, join the desired shape using the **EDIT POLYLINE** command. Then execute the following sequence:

Command: **fillet**
Polyline/Radius/<Select two objects>: **p**
Select 2D polyline: PICK A POINT ON THE POLYLINE. THE SHAPE WILL BE FILLETED.

Appendix 2: AutoCAD ENTITY HANDLING AND AutoLISP

The ROLLCAD process design routines were written in AutoLISP, which is AutoCAD's implementation of the LISP programming language. AutoLISP is an integral part of the AutoCAD package, and includes commands to manipulate the AutoCAD drawing database. This chapter provides a synopsis of how AutoCAD stores entity data in the drawing database, the fundamentals of LISP programming, and techniques for manipulating the AutoCAD drawing database with AutoLISP. For a complete discussion of AutoLISP, refer to the *AutoLISP® Programmer's Reference Manual.*

A2.1 Entity Data Handling

As drawing entities are created in AutoCAD, they are stored in the drawing database file. Each drawing entity entry in the database has associated data which defines the entity. The data is sorted by group codes which indicates the type of data in the group (line type data, coordinates, layer data, etc). The codes are the same as those used by the DXF (ascii Drawing eXchange File) format. For more information, refer to the *AutoCAD Customization Manual.* The entity data groups and associated group codes for groups used in the process design software are:

0	Entity Type
5	Entity Handle
6	Line Type
8	Layer Name
10	Primary Coordinate (start point of line, center of arc.)
11	End Point of Line
40	Arc or Circle Radius
50	Start Angle of Arc
51	End Angle of Arc

Table A2.1 Entity Data Group Codes

These values may be accessed and modified using AutoLISP functions which will be described later in this chapter.

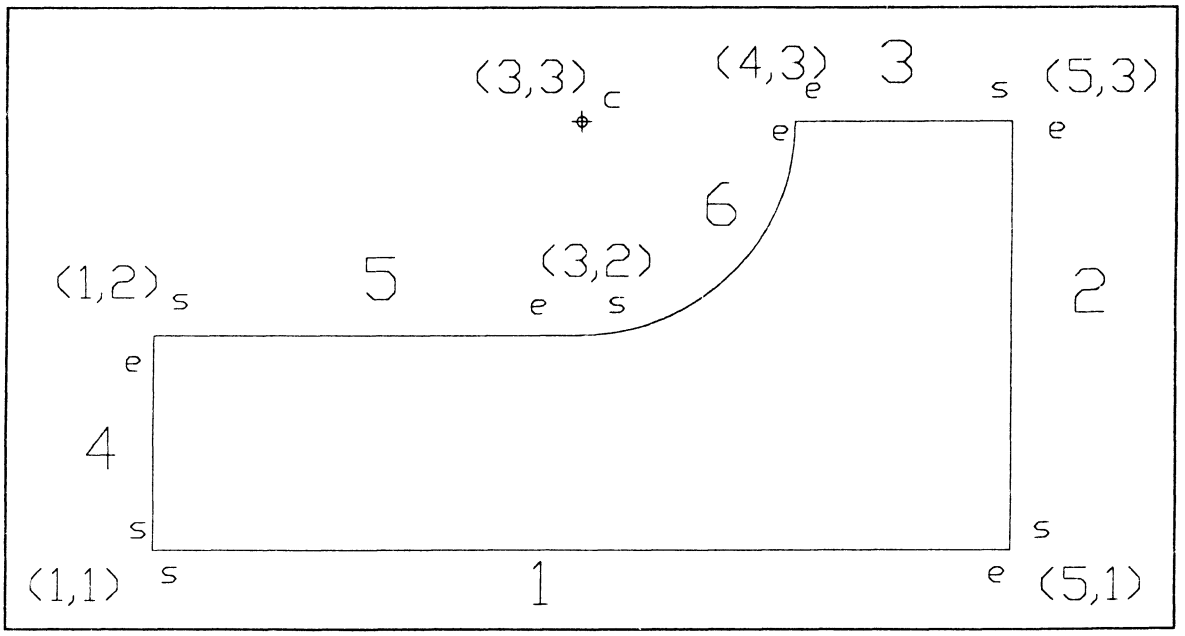


Figure A2.1 Sample drawing illustrating entity data

A simple drawing is shown in figure A2.1 to illustrate these codes. The drawing data will be represented as follows:

Entity 1:

-1	<6000001>	8 Digit entity name assigned by AutoCAD
0	LINE	Entity type
6		Blank line type indicates default for line type
8	PREFORM	Layer name
10	1.0, 1.0, 0.0	x,y,z coordinates of start point (marked 's')
11	5.0, 1.0, 0.0	x,y,z coordinates of end point (marked 'e')

Entity 6:

-1	<6000024>	8 Digit Hex entity name assigned by AutoCAD
0	ARC	Entity type
8	PREFORM	Layer name
10	3.0, 3.0, 0.0	x,y,z coordinate of center
40	1.0	radius of arc
50	270	start angle of arc (cw from positive x axis)
51	0	end angle of arc

This drawing will be used later to illustrate entity manipulations in AutoLISP and ROLLCAD.

A2.2 Fundamentals of AutoLISP

LISP (for LISt Processor) is a programming language developed for artificial intelligence programming. A LISP program is made up of a sequence of expressions. Each expression is a list with the form

(function-name [argument] [argument] ...).

Each expression begins with a left parenthesis and consists of a function name and an optional list of arguments to that function. Every expression returns a value that can be used by a surrounding expression. Hence LISP functions can be (and frequently are) nested.

A2.2.1 LISP Data Types

Arguments which are not functions must be data of some type. The data types are

- Symbols or Variables
- Lists
- Strings
- Integers
- Real Numbers
- File descriptors
- AutoCAD entity names
- AutoCAD selection sets (described later).

Symbols or Variables

Like most programming languages, variables (or symbols, the terms are used interchangeably) may be assigned values using an assignment statement. This value can be any other valid AutoLISP data type.

Lists

Lists are the fundamental data form in LISP. A list is a set of data enclosed in parenthesis. The elements of a list can be any combination of valid AutoLISP data types, including integers, strings, entity names, or other lists.

Strings, Integers, Real Numbers

Strings, Integers, and Real Numbers are common to all programming languages. No description is necessary here.

File Descriptors

File descriptors are alphanumeric labels assigned to files opened by AutoLISP. This descriptor is assigned when the file is opened, and referenced any time the file is accessed (for reading or writing.)

Entity Names

An entity name is a numeric label assigned to entities in a drawing. This label is referenced by AutoLISP to allow selection of entities for processing.

Selection Sets

Selection sets are groups of one or more entities. They are returned by any AutoCAD entity selection function (pick point, window, crossing, etc.) or by

AutoLISP filters (all entities on a certain layer, all entities with a given color, all arcs, etc).

A2.2.2 Evaluation of LISP Functions

The AutoLISP interpreter is an integral part of AutoCAD. The interpreter evaluates LISP expressions in sequential order. Nested expressions are evaluated from the inside out. As noted, all LISP expressions are lists with the form

(function-name [argument] [argument] ...).

where the arguments are any valid AutoLISP data type or expression, and are separated by spaces. The specified function acts on the arguments. For example, a simple addition would take the form

(+ 3 4)

which would return the value 7. In cases where the order of the arguments is important, the arguments are treated in the order in which they appear. For example, the subtraction statement

(- 3 4)

returns the value -1, while the statement

(- 4 3)

returns the value 1.

Nested expressions are evaluated from the inside out. The expression

(+ 5 (* 4 3))

returns the value 17.

Variables are assigned values using the **setq** command, which has the form

(setq *var1 expr1 [var2 expr2] ...*).

The statement

(setq a 5.0)

assigns the value 5.0 to the variable *a*. The statement

(setq b (/ 20 5))

assigns the value 4 to the variable *b*.

Any variable which has not explicitly been assigned a value has the value *nil.* This is the equivalent of a logical *FALSE.* A variable may also be explicitly assigned the value *nil.* LISP contains logical functions (**and, or, not**, etc) which evaluate to *T (logical TRUE).* However, any expression with a non-*nil* value evaluates to *T* for purposes of logical testing. For example, using the **if** function on the following case.

```
(setq a 3)
(setq b 4)
(if (+ a b) ....
```

would evaluate to true, and the **if** condition would be executed. However

```
(if c ....
```

where **c** has not been previously defined, would evaluate false, and the **if** condition would not be executed.

A2.2.3 Some Common AutoLISP Functions

The operation of many LISP functions (**if, open, read, print**) should be apparent to anyone with programming experience. However, LISP also contains many functions which are unique to LISP or AutoLISP. A brief introduction to these functions is included here. For a complete discussion, refer to a text on LISP.

(append *expr ...)*

This takes any number of lists and combines them into one list. For example:

(append '(a b) '(c d)) returns (a b c d).

(assoc *item assoc-list)*

The function searches the association list **assoc-list** for *item* and returns the entry. For example, assuming the association list **arc1** is defined as

((name arc) (radius 1.5) (center (3.2 2.5 0.0)) (start_ang 23) (end_ang 56))

then

(assoc 'center arc1) returns (center (3.2 2.5 0.0)).

(cons *new-first-element list*)

This adds a new first element to the specified list. For example:

(cons 'a '(b c d)) returns (a b c d).

(list *expr ...*)

This functions assembles any number of expressions (*expr*) and assembles them into a list. For example:

(list 'a 'b 'c)	returns	(a b c)
(list 'a '(b c) 'd)	returns	(a (b c) d).

(car *list*)

This function returns the first element of *list.* If *list* is empty, it returns *nil.* For example:

(car '(a b c))	returns	a
(car '((a b) c))	returns	(a b)

(cdr *list*)

This functions returns a list containing all but the first element of *list.* If *list* is empty, it returns *nil.*

(cadr *list*), (cddr *list*), (caddr *list*), etc.

car and **cdr** may be chained up to 4 levels. For example, given the list x with the value

((a b) c d)

(caar x)	is equivalent to	(car (car x))	and returns	a
(cdar x)	is equivalent to	(cdr (car x))	and returns	(b).

In AutoLISP, **car, cadr,** and **caddr** return the first, second, and third element of a list, respectively. Thus, they may be used to obtain the X, Y, and Z coordinates of a point. For example:

```
(setq pt1 '(3.2  4.2  0.0))
```

then

(car pt1)	returns	3.2	which is the X coordinate
(cadr pt1)	returns	4.2	which is the Y coordinate
(caddr pt1)	returns	0.0	which is the Z coordinate.

(nth *n list*)

This function returns the *nth* element of *list*, where *n* is the number of the element to return (zero is the first element). For example:

```
(nth 3 '(a b c d e))        returns      d.
```

(defun *function-name argument-list expression ....*)

The **defun** function defines a function with the name *function-name.* Following the function name is a list of arguments (possibly empty), and the sequence of expressions to be evaluated. For example:

```
(defun add10 (x)
        (+ 10 x)
        )
```

then

(add10 5) returns 15.

Adding **c:** to the beginning of the function name defines the function as an AutoCAD command. Thus the expression

```
(defun c:pmake() ....)
```

defines an AutoCAD command **pmake** with no arguments.

A2.3 AutoLISP Entity Handling

AutoLISP provides access to the entity data in the drawing database. Functions are included for processing both single entities and selection sets of several entities. A discussion of entity processing functions follows.

(ssget *[mode] [pt1 [pt2]]*)

The *ssget* function returns a selection set, which is a group of several entities selected by the user or through an attribute filter. If no arguments are present, *ssget* prompts the user through standard object selection to select objects. The user may then use any standard selection method (ie window, crossing, pick point, with or without snap.)

Several mode options are available. Two which are significant for ROLLCAD are "w" and "x."

(ssget "w" pt1 pt2) selects all entities inside the window from pt1 to pt2.

(ssget "w" (setq crnr (getpoint)) (getcorner crnr)) accepts two input points from the user and draws an "elastic" window box around the selected elements.

(ssget "x" *filter-list*) returns all entities matching the condition specified in the filter list.

A *filter-list* is an association list which specifies which property (or properties) of the entities are to be checked, and what values constitute a match. For example:

returns a selection-set containing all circles in the drawing (recall group code 0 is the entity type.)

(entget *ename*)

This function retrieves the entity whose name is *ename* from the database, and returns a list containing the entity definition data. The data is coded as a LISP association list, from which items may be extracted by means of the **assoc** function. Thus, referring to figure A2.1, the following statement retrieves the entity data for entity 1.

```
(entget entity1)    returns        ((-1 . <Entity name: 60000001>)
                                    (0 . "LINE")
                                    (8 . "PREFORM)
                                    (10 1.0 1.0 0.0)
                                    (11 5.0 1.0 1.0)
                                    )
```

The statement

```
(setq pt3 (cdr (assoc 10 (entget entity1))))
```

assigns the coordinates of the starting endpoint of entity 1 to the variable pt3.

(entmod *elist*)

entmod updates the drawing database by replacing the entity whose name is specified in the -1 group in *elist*. This allows LISP to modify a drawing entity directly. For example:

```
(setq new-ent1 (subst '(10 1.0 1.5 0.0)
                      (assoc 10 (entget entity1))
                      entity1)
                      )
(entmod new-ent1)
```

changes the Y coordinate of the starting endpoint of entity1 to 1.5, and then replaces the old entity data with the new data in the drawing database.

(entnext *[ename]*)

If called with no arguments, *entnext* returns the entity name of the first nondeleted entity in the database. If *entnext* is called with an entity name argument, it returns the entity name of the first nondeleted entity following *ename* in the database. If there is no next entity, *nil* is returned.

(entlast)

This function returns the name of the last entry in the drawing database. This function can be used to mark the endpoint of the database before new entities are created so that the new entities can be accessed using *entnext.*

APPENDIX 3: LISP SOURCE CODE

The LISP source code for the process design routines is given on the following pages. The programs are grouped by the file in which they are contained. There are six program files:

- ALLUTIL.LSP
- DRAFTANG.LSP
- LATERAL.LSP
- LMUTIL.LSP
- MAP.LSP
- THICKMAP.LSP

The programs are discussed in chapter 5.

FILE ALLUTIL.LSP

```
;*************************************************************************
;        function arc2line                              by Chris Fischer
;*************************************************************************
;  Modified by Farid Masri
;
;     approximates arcs as a series of line segments.  the arc is
;  divided into equal segments of maximum angle maxdel.  the endpoints of
;  each of these segments is joined with straight line.
;
;     alpha0 : starting arc angle w.r.t. x axis
;     alphaf : end arc angle w.r.t. x axis
;     dalpha : included angle of arc
;     del : segment angle
;     nsegs : number of segments
;                              ;
(defun arc2line (entlist  / retlist)
;   (setq rotdir rd)
   (setq ccolor (getvar "cecolor")); get current default color
   (setq retlist nil)
   (setq maxdel 0.3490658)   ;  radians, or 20.0 degrees
   (while (setq current (car entlist))
      (setq entlist (cdr entlist))
      (setq enttype (read (cdr (assoc 0 (entget current)))))
      (if (= enttype 'line)
         (setq retlist (append retlist (list current)))
         ) ; end if {enttype = line}
      (if (= enttype 'arc) (progn
         ; find the original color of the current arc
         (setq arccol (cdr (assoc 62 (entget current))))
         (setq arclayer (cdr (assoc 8 (entget current))))
         (if arccol
            (command "color" arccol)
         ;else
            (progn
            (setq layinfo (tblsearch "layer" arclayer)); if entity color does
            (setq arccol (cdr (assoc 62 layinfo))); not exist, get layer color
            (command "color" arccol)
            );end progn
         ); end if {arcol}


     ;find the center, radius, start and end angles, and endpoints
     ;of the  arc.
         (setq cen (cdr (assoc 10 (entget current))))
         (setq rad (cdr (assoc 40 (entget current))))
         (setq alpha0 (cdr (assoc 50 (entget current))))
         (setq alpha1 (cdr (assoc 51 (entget current))))

         (if (> alpha0 alpha1)
            (setq alpha1 (+ alpha1 (* 2 pi)))
            )  ; endif {> alpha0 alpha1}

         (setq apt0 (ptonarc cen rad alpha0))
         (setq apt1 (ptonarc cen rad alpha1))

      ;identify the endpoints of the arc from the endpoints of the
      ;adjoining entities

         (if retlist
            (setq lastendpt (cdr (assoc 11 (entget (last retlist)))))
         ;else
            (progn
   ;****************begin copy block
            (setq lastent (last entlist))
            (setq lasttype (read (cdr (assoc 0 (entget lastent)))))

            (if (= lasttype 'arc)(progn
```

```
            ;identify the endpoints
            (setq ncenter (cdr (assoc 10 (entget lastent))))
            (setq nradius (cdr (assoc 40 (entget lastent))))
            (setq nalpha0 (cdr (assoc 50 (entget lastent))))
            (setq nalpha1 (cdr (assoc 51 (entget lastent))))
            (setq lastendpt0 (ptonarc ncenter nradius nalpha0))
            (setq lastendpt1 (ptonarc ncenter nradius nalpha1))

            (if (or (pt= lastendpt0 apt0) (pt= lastendpt1 apt0)) ; apt0
               (if (pt= lastendpt0 apt0)                    ;coincides with
                  (setq lastendpt lastendpt0)               ;lastendpt
               ;else {pt= lastendpt1 apt0}
                  (setq lastendpt lastendpt1)
                  )  ; endif {pt= lastendpt0 apt0}
            ;else
               (if (pt= lastendpt0 apt1)
                  (setq lastendpt lastendpt0)
               ;else {pt= lastendpt1 apt1}
                  (setq lastendpt lastendpt1)
                  ) ;endif {pt= lastendpt0 apt1}
               ) ;endif {or (pt= lastendpt0 ...}
            )  ; end progn
         ;else {= lasttype line}
               (setq lastendpt (cdr (assoc 11 (entget lastent))))
               )  ; end if {= lasttype 'arc}



;****************end copy block

         )) ; end progn, endif {retlist}
      (setq nextent (car entlist))

      (if (not nextent)      ;if current is the last entity, select the
         (setq nextent (car retlist))   ;first entity in the shape
         )  ;  endif {not nextent}

      (setq nexttype (read (cdr (assoc 0 (entget nextent)))))
      (if (= nexttype 'line)
         (setq nextendpt (cdr (assoc 10 (entget nextent))))
         )  ;  endif {nexttype = line}

         ;  if the entity is an arc, find the first endpoint

      (if (= nexttype 'arc)(progn
         ;identify the endpoints
         (setq ncenter (cdr (assoc 10 (entget nextent))))
         (setq nradius (cdr (assoc 40 (entget nextent))))
         (setq nalpha0 (cdr (assoc 50 (entget nextent))))
         (setq nalpha1 (cdr (assoc 51 (entget nextent))))
         (setq nextendpt0 (ptonarc ncenter nradius nalpha0))
         (setq nextendpt1 (ptonarc ncenter nradius nalpha1))

         (if (pt= lastendpt apt0)  ;identifies apt1 of current arc as
                                   ;second endpoint
            (if (pt= nextendpt0 apt1)
               (setq nextendpt nextendpt0)
            ;else
               (setq nextendpt nextendpt1)
               )  ;endif {pt= nextendpt0 apt1}
        ;else {pt= lastendpt apt1}
            (if (pt= nextendpt0 apt0)
               (setq nextendpt nextendpt0)
            ;else
               (setq nextendpt nextendpt1)
               )  ;endif {pt= nextendpt0 apt0}
            )  ;endif {pt= lastendpt apt0}
         ))  ; end progn end if {= nexttype 'arc}
```

```
;  find all the points on the arc, and draw the line segments

        (setq dalpha (- alpha1 alpha0))
        (setq nsegs (int (+ (/ dalpha maxdel) 1))) ; int (dalpha/maxdel) + 1
        (setq del (/ dalpha nsegs))
        (if (pt= apt0 lastendpt) (progn    ;pt= compares x & y coords
                                           ;returns t if within 0.0005"
           (setq alpha (+ alpha0 del))
           (while  (< alpha (- alpha1 del))
                 ;insure that angle is not close to but less than alpha1
                 ;due to roundoff errors
              (setq endpt (ptonarc cen rad alpha))
              (command "line" lastendpt endpt "")
              (setq retlist (append retlist (list (entlast))))
              (setq lastendpt endpt)
              (setq alpha (+ alpha del))
              )  ;  end while
           (setq endpt (ptonarc cen rad alpha))
           (setq tlength (distance endpt nextendpt))
               (if (> tlength 0.0005)(progn
                   (command "line" lastendpt endpt "")
                   (setq retlist (append retlist (list (entlast))))
                   (command "line" endpt nextendpt ""));end progn
                   ; else
                   (command "line" lastendpt nextendpt "")
               ); end if
           (setq retlist (append retlist (list (entlast))))
           )  ;  end progn
        ;  else {= rd cw}
           (progn
           (setq alpha (- alpha1 del))
           (while  (> alpha (+  alpha0 del))
              (setq endpt (ptonarc cen rad alpha))
              (command "line" lastendpt endpt "")
              (setq retlist (append retlist (list(entlast))))
              (setq lastendpt endpt)
              (setq alpha (- alpha del))
              )  ;  end while
           (setq endpt (ptonarc cen rad alpha))
           (setq tlength (distance endpt  nextendpt))
               (if (> tlength 0.0005)(progn
                   (command "line" lastendpt endpt "")
                   (setq retlist (append retlist (list (entlast))))
                   (command "line" endpt nextendpt ""));end progn
                   ; else
                   (command "line" lastendpt nextendpt "")
               ); end if
           (setq retlist (append retlist (list (entlast))))
           )) ;  end progn, end if {= rd ccw}
           ;erase the original arc
           (command "erase" current "")
        ))  ; end progn, end if {= enttype arc}

     (if (= enttype 'circle) (progn
        ; find the original color of the current arc
        (setq arccol (cdr (assoc 62 (entget current))))
        (setq arclayer (cdr (assoc 8 (entget current))))
        (if arccol
           (command "color" arccol)
        ;else
           (progn
           (setq layinfo (tblsearch "layer" arclayer)); if entity color does
           (setq arccol (cdr (assoc 62 layinfo))); not exist, get layer color
           (command "color" arccol)
           );end progn
        ); end if {arcol}

        (setq cen (cdr (assoc 10 (entget current))))
        (setq rad (cdr (assoc 40 (entget current))))
```

```
          (setq del maxdel)
          (setq alpha del)
          (setq lastendpt (ptonarc cen rad 0.0))
          (setq firstendpt lastendpt)
          (while  (< alpha (- (* 2 pi) del ))
             (setq endpt (ptonarc cen rad alpha))
             (command "line" lastendpt endpt "")
             (setq retlist (append retlist (list(entlast))))
             (setq lastendpt endpt)
             (setq alpha (+ alpha del))
             )  ;  end while

         (command "line" lastendpt firstendpt "")
         (setq retlist (append retlist (list(entlast))))
          (command "erase" current "")
          )) ; end progn, end if { = enttype circle}
       )  ;end while
   (redraw)

   (command "color" ccolor); restor default color
   (setq return retlist)

   )  ;end function


;*************************************************************************
;     FUNCTIONS INT and MOD
;*************************************************************************

(defun int(x / ans um)
   (setq um (getvar "unitmode"))
   (setq ans (read (rtos x 5 2)))
   (setvar "unitmode" um)
   (setq return ans)
   )

(defun mod(x y / a)
;  returns the remainder of x / y
   (setq a (int (/ x y)))
   (setq b (* y a))
   (setq return (- x b))
   )

;*************************************************************************
;     FUNCTION PTONARC
;*************************************************************************
;
;  Finds the coordinates of a point on an arc in the xy plane given
;  the center, radius, and angle on the arc from the positive x axis.
;
(defun ptonarc (center radius angle / xc yc xp yp)
   (setq xc (car center))
   (setq yc (cadr center))
   (setq xp (+ xc (* radius (cos angle))))
   (setq yp (+ yc (* radius (sin angle))))
   (setq return (list xp yp))
   )

;*************************************************************************
;     FUNCTION PT=
;*************************************************************************
;
;     Tests two points for equality within 0.0005"
;
(defun pt= (point1 point2 / x1 x2 y1 y2)
   (setq x1 (car point1))
   (setq x2 (car point2))
   (setq y1 (cadr point1))
   (setq y2 (cadr point2))
```

```
   (setq return (and (< (abs (- x1 x2)) 0.0005) (< (abs (- y1 y2)) 0.0005)))
   )  ; end function

;*************************************************************************
;*************************************************************************
; the following 4 functions obtain xmax, xmin, ymax, and ymin
; points of the original shape. this porgram requires an entity
; list of the the shape.                                    by Farid Masri
;*************************************************************************
   (defun x-min (entlist)
   (setq temp1 (car entlist))
   (setq xmin (cadr (assoc 10 (entget temp1))))
   (setq templist (cdr entlist))
      (while (car templist)
         (setq temp2 (car templist))
         (setq xtemp (cadr (assoc 10 (entget temp2))))
            (if (< xtemp xmin)
               (progn
                  (setq templist (cdr templist))
                  (setq xmin xtemp)
                  (setq temp1 temp2)
               );end progn
               (setq templist (cdr templist))
            ); end if
      );end while
  ; (setq entminx temp1)
  ; (setq yminx (caddr (assoc 10 (entget entminx))))
   (setq return xmin)
   ); end function
;*************************************************************************
;*************************************************************************
   (defun x-max (entlist)
   (setq temp1 (car entlist))
   (setq xmax (cadr (assoc 10 (entget temp1))))
   (setq templist (cdr entlist))
      (while (car templist)
         (setq temp2 (car templist))
         (setq xtemp (cadr (assoc 10 (entget temp2))))
            (if (> xtemp xmax)
               (progn
                  (setq templist (cdr templist))
                  (setq xmax xtemp)
                  (setq temp1 temp2)
               );end progn
               (setq templist (cdr templist))
            ); end if
      );end while
  ; (setq entmaxx temp1)
   (setq return xmax)
   ); end function
;*************************************************************************
; function y-min
; this function returns the entity name with the min y-coordinate
;
   (defun y-min (entlist)
   (setq temp1 (car entlist))
   (setq ymin (caddr (assoc 10 (entget temp1))))
   (setq templist (cdr entlist))
      (while (car templist)
         (setq temp2 (car templist))
         (setq ytemp (caddr (assoc 10 (entget temp2))))
            (if (< ytemp ymin)
               (progn
                  (setq templist (cdr templist))
                  (setq ymin ytemp)
                  (setq temp1 temp2)
               );end progn
               (setq templist (cdr templist))
            ); end if
```

```
      );end while
   (setq entminy temp1)
   (setq return ymin)
   ); end function
;*************************************************************************
; function y-max
; this function returns the entity name with the max y coordinate
;
   (defun y-max (entlist)
   (setq temp1 (car entlist))
   (setq ymax (caddr (assoc 10 (entget temp1))))
   (setq templist (cdr entlist))
      (while (car templist)
         (setq temp2 (car templist))
         (setq ytemp (caddr (assoc 10 (entget temp2))))
            (if (> ytemp ymax)
               (progn
                  (setq templist (cdr templist))
                  (setq ymax ytemp)
                  (setq temp1 temp2)
               );end progn
               (setq templist (cdr templist))
            ); end if
      );end while
   (setq entmaxy temp1)
   (setq return ymax)
   ); end function
;
;*************************************************************************
;*************************************************************************
;*************************************************************************
;*************************************************************************
;     porgram shape select
;
;     by Farid Masri
;     modified version of the porgram shape-select originally
;     written by chris fischer and anbu rathinavel
:
:     this program prompts the user to select the required shape
;     with a window. program will return an orderd list of
;     the shape components in consecutive order.
;
;
;
   (defun shapesel ()
   ;
   ; (prompt "\nselect the desired shape with a window")
   ; (princ "\n \n")(princ)
   (setq shape (ssget "w" (setq crnr (getpoint))(getcorner crnr)))
   (setq nshape shape)
   (command "undo" "control" "all")
   (command "undo" "mark")
   (command "undo" "group")
   (setq first  (entget (ssname shape 0))); get first entity of selection
   (setq currentl (cdr (assoc 8 first))); get layer of selected
                                        ;shape.
        ; find the original color of the current shape
         (setq shapecol (cdr (assoc 62 first)))
         (setq shapelayer (cdr (assoc 8 first)))
         (if shapecol
            (command "color" shapecol)
         ;else
            (progn
            (setq layinfo (tblsearch "layer" shapelayer)); if entity color does
            (setq shapecol (cdr (assoc 62 layinfo))); not exist, get layer color
            (command "color" shapecol)
            );end progn
         ); end if {shapecol}
```

```
;
(setq index 0)
(setq pline nil)
(setq circl nil)
(while (and (< index (sslength shape)) (/= pline 1))
   (setq etype (cdr (assoc 0 (entget (ssname shape index)))))
     (if (= etype "CIRCLE" )
        (setq circl 'T)
     ); end {if = etype circle}
     (if (= etype "POLYLINE")
        (setq pline 1)
     ); end if
   (setq index ( + 1  index))
); end while

   (setq org-layer (getvar "clayer")); store current layer and
   ;move selected object to new layer for easier manipulation.
   ;
   (command "layer" "make" "ounlayer" "")
   (command "layer" "set" org-layer  "")
   (command "change" shape "" "prop" "layer" "ounlayer" "")
   ;
   ;reorder the elements so that they are in consecutive order
   (setq elast (entlast))
      (if circl
          (progn
          (setq shape (ssget "x" (list (cons 8 "ounlayer"))))
          (setq pshape nil)
          (SETQ INDEX 0)
          (setq temp nil)
            (WHILE (< INDEX (SSLENGTH SHAPE))
                (setq etype (cdr (assoc 0 (entget (ssname shape index)))))
                (if (= etype "CIRCLE" )
                    (PROGN
                    (SETQ TEMP (SSNAME SHAPE INDEX))
                    (SETQ PSHAPE (LIST TEMP))
                    (SETQ INDEX (1+ INDEX))
                    ); END PROGN
                ); end {if = etype circle}
            ); END WHILE

          ; (setq temp elast)
           ;  (while (setq temp (entnext temp))
           ;     (setq pshape (append pshape (list temp)))
           ;  );  end while
           ); end progn
           ;else
           (progn
           (if (> (sslength shape) 1)
           ( progn
              ( if (= pline 1)
                 (command "pedit" shape "j" shape "" "x")
                 ;else
                 (command "pedit" shape "y" "j" shape "" "x")
              ); end if
           ); end progn
           ); end if { > sslength..}
           (setq pshape (ssget "x" (list (cons 8 "ounlayer"))))
           ;(command "area" "entity" pshape)
           ;(setq shparea (getvar "area"))
           (command "explode" pshape)
           (setq shape (ssget "x" (list (cons 8 "ounlayer"))))
           (setq pshape nil)
           (setq temp elast)
             (while (setq temp (entnext temp))
                (setq pshape (append pshape (list temp)))
             );  end while
             );end progn
      );end if {circl}
```

```
      (command "change" shape "" "prop" "layer" currentl ""); return
                                        ; shape to its original layer

   (setq return (list pshape nshape))

); end program
```

FILE DRAFTANG.LSP

```
(defun c:lda() (load "draftang"))

;*************************************************************************
;        FUNCTION DRAFTANGLE
;*************************************************************************
;
;     THIS FUNCTION MODIFIES A PART DRAWING, ADDING DRAFT ANGLES TO VERTICAL
;  SURFACES SPECIFIED BY THE USER.  THE USER IS PROMPTED FOR THE SHAPE,
;  THE ENTITY IN THE SHAPE, THE ENDPOINT OF THE ENTITY WHICH IS TO REMAIN
;  FIXED, AND THE DIRECTION TO ROTATE THE ENTITY
;
(defun c:da()
   (setq oldcmdecho (getvar "cmdecho"))
   (setvar "cmdecho" 0)
;   (select1shape)
   (prompt "\nSelect the desired shape with a window:")
   (princ "\n \n")
   (princ)
   (setq entlist (car (shapesel)))
   (command "undo" "end")
;(textscr)
   (setq entlist (arc2line entlist))
   (setq dangle 5.0)   ;Default ANGLE
   (while (setq vertent (car (entsel "\nSelect the entity to be modified, or ENTER
to quit ")))
      (setq oldosmode (getvar "osmode"))
      (setvar "osmode" 1)
      (princ "\nSelect the stationary endpoint ")
      (princ "\n \n")
      (princ)

      (setq endpt (getpoint))
      (princ "\n\nEnter the angle (degrees), or press ENTER or the right \n")
      (princ "mouse button to accept default <")
      (princ dangle) (princ "> ")
      (setq ang nil)
      (setq ang (read (getstring)))
      (if ang (setq dangle ang))
      (setvar "osmode" 0)
      (setq side (getpoint "\nIndicate offset direction with a point left or right
of the line "))
;
;    Identify index of selected entity in ENTLIST
;
      (setq picpt nil)
      (setq index 0)
      (while (and (< index (length entlist)) (not (eq vertent (nth index
entlist))))
         (setq index (1+ index))
         )  ;end while

;
;    Identify which endpoint of the selected entity was selected
;


      (if (pt= endpt (cdr (assoc 10 (entget vertent)))) (progn
         (setq picpt 10)
         (setq otherendpt (cdr (assoc 11 (entget vertent))))
         ) ;end progn
      ;  else if
         (if (pt= endpt (cdr (assoc 11 (entget vertent)))) (progn
            (setq picpt 11)
            (setq otherendpt (cdr (assoc 10 (entget vertent))))
            )  ;end progn
         ;  else
```

```
            (princ "\nERROR ENCOUNTERED: Endpoint selected is not on entity
selected.")
            ) ; end if
          )  ;end if
;
;  generate a long vector in the specified direction
;
          (if (> (car side) (car endpt))
             (setq theta (list (- 90 dangle) (+ 270 dangle)))
           ;else
             (setq theta (list (+ 90 dangle) (- 270 dangle)))
             )  ;  end if

          (if (> (cadr endpt) (cadr otherendpt))
             (setq theta (cadr theta))
          ;else
             (setq theta (car theta))
             )  ;  end if
          (setq pt2 (list (+ (car endpt) (* 100 (cos (* theta 0.0174533))))
                          (+ (cadr endpt) (* 100 (sin (* theta 0.0174533))))
                          '0.0
                          ))
                                        ;  0.0174533 is deg to rad conversion

 ;     Identify the entity intersected by the new vector


          (setq entnum index)
          (if (= picpt 10) (progn
             (setq pointer 1)
             (setq ipt nil)
                (while (and (not ipt)
                            (/= entnum (mod (+ index (- (length entlist) 2)) (length
entlist)))
                            ) ;  end and
                   ;while loop conditions:
                   ;      =>  (1)  Intersection has not been found (not ipt)
                   ;      =>  (2)  The entity being tested is not the entity
                   ;               immediately preceeding index
                   ;
                   ;       Note: entnum is calculated at beginning of loop
                   ;       pointer is incremented at end, thus the value of
                   ;       entnum being tested here is one less than will be
                   ;       used for calculations in the current iteration.

                (setq entnum (mod (+ index pointer) (length entlist)))
                (setq intent (nth entnum entlist))
                (setq pt3 (cdr (assoc 10 (entget intent))))
                (setq pt4 (cdr (assoc 11 (entget intent))))
                (setq ipt (inters endpt pt2 pt3 pt4 'T))
                (setq pointer (1+ pointer))
                )  ;end while
             ;If an intersection was found, erase the entities between vertent
             ;and intent, and remove them from entlist.
             (if ipt (progn
                   (setq remlist nil)
                   (setq i (- pointer 2))
                   (while (> i 0)
                      (setq entnum (mod (+ index i) (length entlist)))
                      (command "erase" (nth entnum entlist) "")
                      (setq remlist (append remlist (list entnum)))
                      (setq i (1- i))
                      )  ;  end while

                   (setq remlist (sort remlist))
                   (while (setq i (car remlist))
                      (setq remlist (cdr remlist))
                      (setq entlist (remove i entlist))
```

```
                  )  ;  end while
               )) ;  end progn, endif {ipt}


            ;if no intersection was found on any entity, find the intersection
            ;with the extension of the first entity after index

            (if (not ipt) (progn
               (if (/= index (1- (length entlist))) ;  =>last element in list
                  (setq intent (nth (1+ index) entlist))
               ;else
                  (setq intent (car entlist))
                  )  ;end if
               (setq pt3 (cdr (assoc 10 (entget intent))))
               (setq pt4 (cdr (assoc 11 (entget intent))))
               (setq ipt (inters endpt pt2 pt3 pt4 nil))
               )) ; end progn, end if

;modify the endpoints of the selected and intersected entities
            (setq entdata (entget vertent))
            (setq entdata (subst (cons 11 ipt)
                           (assoc 11 entdata)
                           entdata
                           ))
            (entmod entdata)

            (setq entdata (entget intent))
            (setq entdata (subst (cons 10 ipt)
                           (assoc 10 entdata)
                           entdata
                           ))
            (entmod entdata)



            ))  ;end progn, endif   {= picpt 10}

         (if (= picpt 11) (progn
            (setq pointer (1- (length entlist)))
            (setq ipt nil)
               (while (and (not ipt)
                        (/= entnum (mod (+ index (+ (length entlist) 2)) (length
entlist)))
                        ) ;  end and
                  ;while loop conditions:
                  ;      =>  (1)  Intersection has not been found (not ipt)
                  ;      =>  (2)  The entity being tested is not the entity
                  ;               immediately following index
                  ;           see note in while loop for (= picpt 10) above

               (setq entnum (mod (+ index pointer) (length entlist)))
               (setq intent (nth entnum entlist))
               (setq pt3 (cdr (assoc 10 (entget intent))))
               (setq pt4 (cdr (assoc 11 (entget intent))))
               (setq ipt (inters endpt pt2 pt3 pt4 'T))
               (setq pointer (1- pointer))
               )  ;end while

            ;If an intersection was found, erase the entities between vertent
            ;and intent, and remove them from entlist.
            (if ipt (progn
                  (setq remlist nil)
                  (setq i (+ pointer 2))
                  (while (< i (length entlist))
                     (setq entnum2 (mod (+ index i) (length entlist)))
                     (command "erase" (nth entnum2 entlist) "")
                     (setq remlist (append remlist (list entnum2)))
                     (setq i (1+ i))
                     )  ;  end while
```

```
                  (setq remlist (sort remlist))
                  (while (setq i (car remlist))
                     (setq remlist (cdr remlist))
                     (setq entlist (remove i entlist))
                     )  ;  end while
               )) ;  end progn, endif {ipt}


            ;if no intersection was found on any entity, find the intersection
            ;with the extension of the first entity after index

            (if (not ipt) (progn
               (if (/= index 0) ;  => first element in list
                  (setq intent (nth (1- index) entlist))
               ;else
                  (setq intent (last entlist))
                  )  ;end if
               (setq pt3 (cdr (assoc 10 (entget intent))))
               (setq pt4 (cdr (assoc 11 (entget intent))))
               (setq ipt (inters endpt pt2 pt3 pt4 nil))
               )) ; end progn, end if

;modify the endpoints of the selected and intersected entities
            (setq entdata (entget vertent))
            (setq entdata (subst (cons 10 ipt)
                                 (assoc 10 entdata)
                                 entdata
                                 ))
            (entmod entdata)

            (setq entdata (entget intent))
            (setq entdata (subst (cons 11 ipt)
                                 (assoc 11 entdata)
                                 entdata
                                 ))
            (entmod entdata)



            ))  ;end progn, endif   {= picpt 11}


      )  ;end while
   (setvar "cmdecho" oldcmdecho)
   (princ)
   )  ;end function

;*************************************************************************
;           FUNCTION REMOVE
;*************************************************************************

;  removes the specified element from the list.  0 is the first element

(defun remove (x entlist / i templist x temp)
   (setq i 0)
   (setq templist nil)
   (while (< i (length entlist))
      (if (/= i x) (progn
         (setq temp (nth i entlist))
         (setq templist (append templist (list temp)))
         )) ;  end progn, end if
      (setq i (1+ i))
      )  ;  end while
   (setq return templist)
   )  ;end function


;*************************************************************************
;        FUNCTION SORT
```

```
;*************************************************************************

(defun sort (lis)
      ;from J.R. ANDERSON, ET AL, ESSENTIAL LISP
      (cond ((null lis) nil)
         (t (insert (car lis) (sort (cdr lis))))))

(defun insert (item lis)
      (cond ((null lis) (list item))
         ((< item (car lis)) (cons item lis))
         (t (cons (car lis)(insert item (cdr lis))))))


;*********************************************************************
;*********************************************************************
;*********************************************************************
;      PORGRAM SHAPE SELECT
;
;      BY FARID MASRI
;      MODIFIED VERSION OF THE PORGRAM SHAPE-SELECT ORIGINALLY
;      WRITTEN BY CHRIS FISCHER AND ANBU RATHINAVEL
:
:      THIS PROGRAM PROMPTS THE USER TO SELECT THE REQUIRED SHAPE
;      WITH A WINDOW. PROGRAM WILL RETURN AN ORDERD LIST OF
;      THE SHAPE COMPONENTS IN CONSECUTIVE ORDER.
;
;
;
(DEFUN select1shape ()
   (prompt "\n    ")
   (prompt "\n")
   (PROMPT "\nSelect the desired shape with a window")
   (SETQ SHAPE (SSGET "W" (SETQ CRNR (GETPOINT))(GETCORNER CRNR)))
   ;PROMPT USER FOR TO SELECT SHAPE WITH A WINDOW.
   ;
   (SETQ ORG-LAYER (GETVAR "CLAYER")); STORE CURRENT LAYER AND
   ;MOVE SELECTED OBJECT TO NEW LAYER FOR EASIER MANIPULATION.
   ;
   (COMMAND "LAYER" "MAKE" "NLAYER" "")
   (COMMAND "LAYER" "SET" ORG-LAYER  "")
   (COMMAND "CHANGE" SHAPE "" "PROP" "LAYER" "NLAYER" "")
   ;
   ;REORDER THE ELEMENTS SO THAT THEY ARE IN CONSECUTIVE ORDER
   (SETQ ELAST (ENTLAST))
   (COMMAND "PEDIT" SHAPE "Y" "J" SHAPE "" "X")
   (SETQ PSHAPE (SSGET "X" (LIST (CONS 8 "NLAYER"))))
   ;(COMMAND "AREA" "ENTITY" PSHAPE)
   ;(SETQ SHPAREA (GETVAR "AREA"))
   (COMMAND "EXPLODE" PSHAPE)
   (SETQ PSHAPE NIL)
   (SETQ TEMP ELAST)
   (WHILE (SETQ TEMP (ENTNEXT TEMP))
      (SETQ PSHAPE (APPEND PSHAPE (LIST TEMP)))
      );  END WHILE
   (SETQ entlist PSHAPE)
); END PROGRAM
```

FILE LATERAL.LSP

```
;*************************************************************************
;lateral flow diagram                                        by farid masri
;*************************************************************************
; This function draws a graph of the metal lateral flow. It requires
; the user to select the initial shape to be rolled and the final shape.
; Note that the shapes should be aligned one underneath the other befor running
; the program.  This alignment should be similar to the relative positions
; that the actual shapes assume when enetering and leaving the rolling mill.
; Then the program asks for the maginification of the yaxis and the resolution
; of the graph.  the resolution means the number of segments that the shapes
; are divided into in order to draw the graph (this is also the number of
;  lines  used to draw the graph).
;
(defun c:lateral ( / xstart xbegin xend xpos )
;
;get current  snap and color settings and save them. these variables are
;changed during execution and will be restored at the end of the progarm:
;
(setq org-color (getvar "cecolor")); get original color.
(setq org-osnap (getvar "osmode"))
(setq org-snap (getvar "snapmode"))
(setq org-cmdecho (getvar "cmdecho"))
;
(setvar "cmdecho" 0); switch off command echo for faster execution
(command "osnap" "none"); turn off all snap modes for proper execution
(command "snap" "off")
;
;
;get current layer
(setq cl (getvar "clayer"))
; check to see if layer ouarea already exists
; if it does then swith to that layer ( this will make execution faster)
(setq chk (tblsearch "layer" "ouarea" ))
(if (/= chk nil)
    (command "layer" "set" "ouarea" "")
); end if
(princ "\n \n")(princ)
(prompt "\nselect the initial shape with a window")
(princ "\n \n")(princ)
;
(setq entlist1 (shapesel2))
(setq shparea1 (cadr entlist1))
(setq entlist1 (car entlist1))
;
(prompt "\nselect the final shape with a window")
(princ "\n \n")(princ)
;
(setq entlist2 (shapesel2))
(setq shparea2 (cadr entlist2))
(setq entlist2 (car entlist2))
;
(princ "\n \n") (princ)
;
(prompt "\ntype in the y coordinate scaling factor of the graph <10>  ")
(princ "\n \n") (princ)
(setq scal (read (read-line)));
;
(prompt "\ntype in the graph resolution (10 - 30)     <20 default>  ")
(princ "\n \n") (princ)
(setq ress (read (read-line)))
;
; (princ "\n \n")(princ)
(prompt "\nselect the starting point of the lateral-flow graph")
(princ "\n \n")(princ)
;
(setq xbegin (getpoint)); get starting point of graph
(setq entlist1 (arc2line entlist1))
(setq entlist2 (arc2line entlist2))
```

```
;
; reorder the entity list to start with the leftmost one
;
(setq initminpt (find-minpt entlist1))
(if (> (length initminpt) 1 )
     (setq initminpt (find-ymax initminpt))
     )       ;endif
(setq entlist1 (startwith entlist1 initminpt))

;  re-order finlshp to start with standard point

(setq finlminpt (find-minpt entlist2))
(if (> (length finlminpt) 1 )
   (setq finlminpt (find-ymax finlminpt))
   )  ;endif
(setq entlist2 (startwith entlist2 finlminpt))
;
; set y scaling factor if different from default
(setq scaley 10); default scale factor
(if ( /= scal nil)
    (setq scaley scal)
); end if
;
; set the resolution of the graph
(setq res 25)
(if (/= ress nil)
    (setq res ress)
);end if
;
(setq aratio (/ shparea2 shparea1)); find area ratio of shapes
;
(setq xmin-1 (x-min entlist1)); find minimum and maximum x coordinates
(setq xmax-1 (x-max entlist1)); for the two shapes.
(setq xmin-2 (x-min entlist2))
(setq xmax-2 (x-max entlist2))
;
;find graph length by setting it equal to the widest of the two shapes.
;
(setq l1 (- xmax-1 xmin-1))
(setq l2 (- xmax-2 xmin-2))
(setq graphlength l1)
   (if (> l2 l1)
       (setq graphlength l2)
   ); end if
;
;  find the endpoint of the graph
(setq gendpoint (list ( + (car xbegin) graphlength) (cadr xbegin)))
;
;divide shape into  equal intervals
(setq deltax (/ graphlength res))
;
(setq xstart xmin-1);      starting point of mapping
    (if (< xmin-2 xmin-1)
        (setq xstart xmin-2)
    ); end if
(command "color" "5"); set color blue
(command "line"  xbegin gendpoint ""); draw ordinate line of graph
(setq tempp (entlast))
(command "color" "9") ; set color dark red
;
(latcalc entlist1 entlist2 scaley res);
;
; hatching the graph
;
; (command "color" "1")
;(command "hatch"  "ansi37"  "0.06"  "0"  graph  "")
;
; put text on the graph
;
```

```
    (command "color" "11")
    (setq xtext (list (- (car xbegin ) 1.0)(+ (cadr xbegin) 0.35 )))
    (command "text" "s" "standard" xtext "0.15" "0" "above; flow to the left" )
    (ssadd (entlast) graph)
    (setq xtext (list ( - (car gendpoint) 2.25) (- (cadr xbegin) 0.35)))
    (command "text" "s" "standard" xtext "0.15" "0" " below; flow to the right" )
    (ssadd (entlast) graph)
    ;
    ; move graph to its own layer
    ; (command "layer" "make" "oulateral" "")
    ;(command "change" graph  "" "prop" "layer" "oulateral" "")
    ;(command "layer" "off" "ounlayer" "")
    ;
    ; restore original settings
    (command "layer" "set" cl ""); restore current layer to original layer
    (command "redraw") ; clean up the window
    (setvar "osmode" org-osnap); restore original osnap mode
    (setvar "snapmode" org-snap); restore snap mode
    (command "color" org-color);  restore original color
    (setvar "cmdecho" org-cmdecho)
    (prompt "\n done")
    (princ "\n \n")
    (princ)
    ;
 );  end of function lateral

;*************************************************************************
;     porgram shape select
;
;     by farid masri
;     modified version of the porgram shape-select originally
;     written by chris fischer and anbu rathinavel
:
:     this program prompts the user to select the required shape
;     with a window. program will return an orderd list of
;     the shape components in consecutive order.
;
;
;
(defun shapesel2 ()
;    (prompt "\nselect the desired shape with a window")
    (setq shape (ssget "w" (setq crnr (getpoint))(getcorner crnr)))
    (setq first  (entget (ssname shape 0))); get first entity of selection
    (setq currentl (cdr (assoc 8 first))); get layer of selected
                                                ;shape.
    (setq index 0)
    (setq pline nil)
    (while (and (< index (sslength shape)) (/= pline 1))
       (setq etype (cdr (assoc 0 (entget (ssname shape index)))))
       (if (= etype "POLYLINE")
          (setq pline 1)
       ); end if
    (setq index ( 1+  index))
    ); end while
    (setq org-layer (getvar "clayer")); store current layer and
    ;move selected object to new layer for easier manipulation.
    ;
    (command "layer" "make" "ounlayer" "")
    (command "layer" "set" org-layer  "")
    (command "change" shape "" "prop" "layer" "ounlayer" "")
    ;
    ;reorder the elements so that they are in consecutive order
    (setq elast (entlast))
    (if (> (sslength shape) 1)
       ( progn
       ( if (= pline 1)
          (command "pedit" shape "j" shape "" "x")
        ;else
          (command "pedit" shape "y" "j" shape "" "x")
```

```
      ); end if
      ); end progn
   ); end if
   (setq pshape (ssget "x" (list (cons 8 "ounlayer"))))
   (command "area" "entity" pshape)
   (setq shparea (getvar "area"))
   (command "explode" pshape)
   (setq shape (ssget "x" (list (cons 8 "ounlayer"))))
   (setq pshape nil)
   (setq temp elast)
   (while (setq temp (entnext temp))
      (setq pshape (append pshape (list temp)))
      );  end while
   (command "change" shape "" "prop" "layer" currentl ""); return shape
                                                ; to its original layer
   (setq return (list pshape shparea))
); end function
;*************************************************************************
;  function: lateralcalc                                 by farid masri
;*************************************************************************
; this function does all lateral flow calculations and mapping.
; the function uses the function dt to obtain delta t and then
; the results are plotted on the screen.

(defun latcalc (l1 l2 scaley res)
  (setq point1 xbegin)
  (setq graph (ssadd))
  (ssadd tempp graph)
  (setq xpos ( + xstart deltax))
  ;
  (while ( <= xpos (+ graphlength xstart))
    (setq deltat (dt entlist1 entlist2 xpos xmax-1 xmax-2))
    (setq deltat ( * deltat scaley)); scale the output in the y direction
     ;
     ; graphing the points
     ;
     (setq point2 (list (+ (car point1) deltax) (+ (cadr xbegin) deltat)))
     (command "line" point1 point2 "")
     (setq tempp (entlast))
     (ssadd tempp graph)
     (setq point1 point2)
     (setq xpos (+ xpos deltax))
  ); end while
  (command "line" point1 gendpoint ""); draw last line to close graph
  (setq tempp (entlast))
  (ssadd tempp graph)
); end function
;*************************************************************************
; function dt                                            by farid masri
;*************************************************************************
; this function calculates delta t (thickness) for two shapes as used
; in determining lateral flow. it needs the entity lists of both shapes
; and the current x-position and the maximum x coordinate of each shape
;
(defun dt (entlist1 entlist2 xpos xmax-1 xmax-2 / deltt templist areai areaf)
     (setq templist (idents entlist1 xpos))
     (setq areai 0)
     (if ( and ( = templist nil) (> xpos xmax-1))
       ( setq areai shparea1)
       ; else
       (if (/= templist nil )
         (progn
         (setq l (car templist))
         (setq u (cadr templist))
         (setq areai (arealeft entlist1 u l xpos));find initial area at interval
         );end progn
       );end if
     ); end if
     (setq templist (idents entlist2 xpos))
```

```
      (setq areaf 0)
      ( if (and ( = templist nil)(> xpos xmax-2))
        (setq areaf shparea2)
        ; else
        (if (/= templist nil)
          (progn
          (setq l (car templist))
          (setq u (cadr templist))
          (setq areaf (arealeft entlist2 u l xpos)); find final area at interval
          ); end progn
        );end if
      ); end if
      (setq deltt ( - areaf (* aratio areai)))
      (setq return deltt)
); end function dt


;*************************************************************************
;   function autolineup                                    by farid masri
;*************************************************************************
;  this function lines up two shapes vertically according to their geometric
;  centers.
;
(defun c:autolineup ( )
       (setq org-osnap (getvar "osmode"))
     ;  (setq org-ortho (getvar "orthomode"))
       (setq org-cmdecho (getvar "cmdecho"))
       (setvar "cmdecho" 0)
       (command "osnap" "none")
      ; (command "ortho" "on")
     ;  (command  "undo" "control" )
       (prompt "\n select the shape to align with a window")

       (princ "\n \n")(princ)
       (setq entlist  (shapesel))
       (setq entlist2 (car entlist))
       (setq shape2 (cadr entlist))
       (setq nshape2 shape2)
       (prompt "\n select reference shape with a window")
       (princ "\n \n")(princ)
       (setq entlist (shapesel))
       (setq entlist1 (car entlist))
       (setq shape1 (cadr entlist))
       (setq entlist nil)
      ;(setq shape1 (ssget "w" (setq crnr (getpoint))(getcorner crnr)))
       (prompt "\nrunning........please wait")
       (princ "\n \n")(princ)
     ;  (command  "undo" "mark" )
     ;  (setq index 0)
      ; (setq entlist1 nil)
      ;   (while (< index (sslength shape1))
      ;     (setq entlist1 (cons  (ssname shape1 index) entlist1))
      ;     (setq index (1+ index))
      ;   ); end while

      (setq entlist1 (arc2line entlist1))
       (setq lastent (entlast))

       (setq entlist2 (arc2line entlist2))
     ;  (while (entnext lastent)      ; add the lines generated by arc2line
     ;         (setq temp (entnext lastent)); to the shape
     ;         (ssadd temp shape2)
     ;         (setq lastent temp)
     ;  ); end while


       (setq cent1 (/ (+ (x-max entlist1) (x-min entlist1)) 2))
```

```
        (setq cent2 (/ (+ (x-max entlist2) (x-min entlist2)) 2))
        (setq cent2 (list cent2 (y-min entlist2))); center point of shape2
        (setq refpoint (list cent1 (y-min entlist2)))
        (command "undo" "end")
        (command "undo" "back")
       ; (command "undo" "end")
        (command "undo" "mark")

        (command "move" nshape2 "" cent2 refpoint)
        (command "redraw")
        (setvar "osmode" org-osnap)
       ; (setvar "orthomode" org-ortho)
        (setvar "cmdecho" org-cmdecho)
        (princ "\n \n")(princ)

); end function autolineup


;*****************************************************************************
;*****************************************************************************
; function lineup                                              by FARID MASRI
;*****************************************************************************
; this function lines up a shape in reference of another shape


(defun c:lineup ( / dist basepnt shape )
        (setq org-osnap (getvar "osmode"))
        (setq org-ortho (getvar "orthomode"))
        (setq org-cmdecho (getvar "cmdecho"))
        (setvar "cmdecho" 0)
        (command "osnap" "end")
        (command "ortho" "on")
        (prompt "\n select the shape to be moved with a window")
        (princ "\n \n")(princ)
        (setq shape (ssget "w" (setq crnr (getpoint))(getcorner crnr)))
        (princ  "\n \n pick a reference point on the shape to be moved ")
        (princ "\n \n")(princ)
        (setq basepnt (getpoint ))
        (prompt "\nrunning........please wait")
        (princ "\n \n")(princ)

        (setq dist (di))
        (command "move" shape "" basepnt dist)
        (command "redraw")
        (setvar "osmode" org-osnap)
        (setvar "orthomode" org-ortho)
        (setvar "cmdecho" org-cmdecho)
        (princ "\n \n")(princ)
      ); end function lineup


(defun di ( / b)
        (princ "\n pick a reference point on reference shape: " )
        (princ "\n \n")(princ)
        ( setq b  (getpoint))
        (princ "\n enter point of new location of shape: ")
        (princ "\n \n")(princ)
        (getpoint b)

); end function di
```

FILE LMUTIL.LSP

```
;*************************************************************************
;        FUNCTION FIND-MINPT
;*************************************************************************
;
;  By Chris Fischer
;
;     Finds and returns the entity in a list of entities (a shape) with
;  the smallest x coordinate of its starting endpoint.
;  If there is more than one endpoint with equal x values, all are returned
;  in a list.
;
(defun find-minpt (entlist)
   (setq minpt (list (car entlist)))      ;minpt is the entity whose first
   (setq entlist (cdr entlist))           ;endpoint has the smallest value
                                          ;so far
   (setq xmin (cadr (assoc 10 (entget (car minpt))))) ;entget returns the
                                          ;attribute list for the entity
                                          ;as a series of dotted pairs.
                                          ;assoc 10 returns the first endpt
                                          ;and cadr gets the second atom in
                                          ;the list, which is the x coord.

   (while (setq temp (car entlist))
      (setq entlist (cdr entlist))
      (setq xtemp (cadr (assoc 10 (entget temp))))

      (if (< xtemp xmin) (progn
         (setq minpt (list temp))
         (setq xmin xtemp)
         )) ;end progn, endif

      (if (= xtemp xmin)
         (setq minpt (append minpt (list temp)))
         )  ;endif
      )     ;end while
   (setq return minpt)
)     ;end function

;*************************************************************************
;        FUNCTION FIND-YMAX
;*************************************************************************
;
;  By Chris Fischer
;
;     Finds and returns the entity in a list of entities with
;  the largest y coordinate of its starting endpoint.
;
;
(defun find-ymax (entlist)
   (setq ymax (list (car entlist)))      ;ymax is the entity whose first
   (setq entlist (cdr entlist))          ;endpoint has the largest value
                                         ;so far
   (setq maxval (caddr (assoc 10 (entget (car ymax))))) ;entget returns the
                                         ;attribute list for the entity
                                         ;as a series of dotted pairs.
                                         ;assoc 10 returns the first endpt
                                         ;and caddr gets the third atom in
                                         ;the list, which is the y coord.

   (while (setq temp (car entlist))
      (setq entlist (cdr entlist))
      (setq ytemp (caddr (assoc 10 (entget temp))))

      (if (> ytemp maxval) (progn
         (setq ymax (list temp))
         (setq maxval ytemp)
         )) ;end progn, endif
```

```
      )     ;end while
   (setq return ymax)
)     ;end function

;*************************************************************************
;        FUNCTION STARTWITH
;*************************************************************************
;     Receives a list of the entity names in a shape (in consecutive order)
;  and the name of the entity to start the output list, and returns a list
;  in the same sequence with the specified name starting.
;
(defun startwith (entlist entname)
   (if (not (atom entname)) (setq entname (car entname)))
      ;endif
   (setq listb nil)
   (while (setq temp (car entlist))
      (if (= temp entname) (progn
         (setq retlist (append entlist listb))
         (setq entlist nil)
         ))  ;end progn  end if
      (setq listb (append listb (list temp)))
      (setq entlist (cdr entlist))
      ) ;  end while
   (setq return retlist)
   )  ;  end function

;*************************************************************************
;        FUNCTION READELEM
;*************************************************************************
;
;     Returns an arbitrary element specified by "index" from a list "l."
;
(defun readelem (l index)

        (setq pointer 1)                          ;pointer is used to keep
                                                  ;   track of the current
                                                  ;   position in the array
        (while (< pointer index)                  ;Index through the array,
                (setq l (cdr l))                  ;   removing each element from
                (setq pointer (1+ pointer))       ;   the front of the array.
                )       ;end while

        (setq return (car l))                ;return desired value
)                       ;end function


;*************************************************************************
;        FUNCTION AREALEFT
;*************************************************************************
;
;     Returns the area of the shape defined by the entities in entlist
;  to the left of the x coordinate specified in the parameter XPOINT.
;  The entities intersected by this point are  UPPERENTity and LOWERENTity,
;  where upper and lower refer to the entity indices, and not necessarily
;  their physical orientation.
;     The right endpoints of the entities specified are passed in the
;  parameters UpperEND and LowerEND.  (One of these parameters will be 10,
;  and the other 11, refering to "assoc 10" and "assoc 11" -- specifications
;  of the starting or ending endpoints of the element.
;     The parameters IUPPERENT and ILOWERENT are the indices of the
;  respective indices.
;
(defun arealeft (entlist iupperent ilowerent  xpoint / index)
   (if (not areacount) (setq areacount 0)) ;initialize areacount if it is nil
   (setq areacount (1+ areacount))
;  for timed execution:  start timer every 25th execution
   (if (= (mod areacount 25) 0)
      (command "time" "reset" "on" "")
      )  ;endif
```

```
    (setq e entlist)
    (setq u iupperent)
    (setq l ilowerent)
    (setq x xpoint)
    (command "undo" "control" "none")
    (command "undo" "all")
    (command "undo" "mark")
    (command "undo" "mark")     ;extra mark prevents "this will undo
                                ; everything" query
    (command "undo" "group")

    (setq upperent (readelem entlist iupperent))    ;  readelem returns the
                                                    ;entity name from entlist
    (setq lowerent (readelem entlist ilowerent))
    (setq ulend (cdr (assoc 11 (entget upperent))))  ;  UpperLeftEND
    (setq urend (cdr (assoc 10 (entget upperent))))  ;  UpperRightEND
    (setq llend (cdr (assoc 10 (entget lowerent))))  ;  LowerLeftEND
    (setq lrend (cdr (assoc 11 (entget lowerent))))  ;  LowerRightEND
    (setq pnt1 (list xpoint 0 0))
    (setq pnt2 (list xpoint 1 0))
    (setq uipt (inters pnt1 pnt2 ulend urend nil))     ;  Returns the
    (setq lipt (inters pnt1 pnt2 llend lrend nil))     ;intersection between
                                          ;infinite extensions of the line
                                          ;defined by pnt1, pnt2 and the
                                          ;upper or lower entity, respectively
  ; If an entity lies vertically in line with XPOINT, set the intersetction
  ; point equal to the "right" endpoint of the entity in question.

    (if (= xpoint (car ulend) (car urend))
       (setq uipt urend))   ;endif
    (if (= xpoint (car llend) (car lrend))
       (setq lipt lrend))   ;endif

  ; Read all entities to the left of the intersected entities

    (setq selset nil)
    (setq index 1)
    (setq selset (ssadd))
    (while (< index ilowerent)
       (setq selset (ssadd (readelem entlist index) selset))
       (setq index (1+ index))
       )  ; end while
    (setq index ( 1+ iupperent))
    (while (<= index (length entlist))
       (setq selset (ssadd (readelem entlist index) selset))
       (setq index (1+ index))
       )  ; end while

;   Create a polyline from the selection set plus the lines from ulend
;   to uipt, from uipt to lipt, and from lipt to llend

    (command "line" ulend uipt "")
    (setq selset (ssadd (entlast) selset))
    (command "line" uipt lipt "")
    (setq selset (ssadd (entlast) selset))
    (command "line" lipt llend "")
    (setq selset (ssadd (entlast) selset))

    (command "layer" "make" "ouarea" "")
    (command "change" selset "" "prop" "layer" "ouarea" "")

    (command "pedit" selset "y" "j" selset "" "x")
    (setq pshape (ssget "x" (list (cons 8 "ouarea"))))
    (command "area" "entity" pshape)
    (setq larea (getvar "area"))

;     (command "undo" "back" )
```

```
   (command "undo" "end")
   (command "undo" "back" )
;  (command "redraw")

;  for debug, stop timer if it was started, and write result to list
   (if (= (mod areacount 25) 0) (progn
      (command "time" "off" "")
      (setq timer (getvar "tdusrtimer"))
      (setq timelist (append timelist (list timer)))
      ))  ;end progn end if

   (setq return larea)
   )  ;  end function

;*************************************************************************
;              PROGRAM IDENTS
;*************************************************************************
;
;   IDentify ENTitieS:  Identify indices of lower and upper entities
;   intersecting a vertical line through XPOINT.  Returns nil if xpoint
;   is outside of the shape.  Not valid for arcs
;
(defun idents (entlist xpoint / index)
   (setq ents nil)
   (setq index 1)
   (while (<= index (length entlist))
      (setq temp (readelem entlist index))
      (setq x1 (cadr (assoc 10 (entget temp))))
      (setq x2 (cadr (assoc 11 (entget temp))))

;sort x1 and x2
      (if (< x2 x1) (progn
         (setq t x2)
         (setq x2 x1)
         (setq x1 t)
         ))    ; end progn, endif

;if the entity is very nearly vertical (within 5.0E-4), adjust it to vertical
;so it will not cause problems with the intersection function

      (if (and (< (abs (- x1 x2)) 5.0E-4) (/= x1 x2))(progn
         ;update assoc 10 of the entity
         (setq entdata (entget temp))
         (setq p1 (cdr (assoc 10 entdata)))
         (setq p1 (cons x1 (cdr p1)))
         (setq entdata (subst (cons 10 p1)
                        (assoc 10 entdata)
                        entdata
                        ))
         (entmod entdata)
         ;uptade assoc 11 of the previous entity
         (if (/= index 1) (progn
            (setq temp (readelem entlist (1- index)))
            (setq entdata (entget temp))
            (setq p1 (cdr (assoc 11 entdata)))
            (setq p1 (cons x1 (cdr p1)))
            (setq entdata (subst (cons 11 p1)
                           (assoc 11 entdata)
                           entdata
                           ))
            ) ;end progn
         ;else
            (progn
            (setq temp (readelem entlist (length entlist)))
            (setq entdata (entget temp))
            (setq p1 (cdr (assoc 11 entdata)))
            (setq p1 (cons x1 (cdr p1)))
            (setq entdata (subst (cons 11 p1)
                           (assoc 11 entdata)
```

```
                              entdata
                              ))
                 ))  ;end progn, end if  {/= index 1}
              )) ;end progn, endif  { abs (- x1 x2) < 5.0e-4}1
              (entmod entdata)

;test if xpoint is between the endpoints of the entity
;and be sure the entity is not vertical
      (if (and (<= x1 xpoint x2) (/= x1 x2))
         (setq ents (append ents (list index)))
         )  ;  end if
      (setq index (1+ index))
      )  ;end while

;select the first and last entities in the list

   (setq ents (list (car ents) (last ents)))
   (if (/= (car ents) nil)
      (setq return ents)
      ; else
      (setq return nil)
      )  ;endif
   )  ;end function
```

FILE MAP.LSP

```
(defun c:lm () (load "map"))  ;utility to reload program after editing

;*************************************************************************
;           PROGRAM C:MAP
;*************************************************************************
;  calling routine for mapping functions
;
(defun c:map ()
   (prompt "\n")
   (prompt "\n")
;   (prompt "NOTE:  THIS PROGRAM HAS A KNOWN BUG GENERATING INTERMEDIATE SHAPES")
;              bug corrected -- caused by snap
;   (setq pt (ssget "x" (list (cons 0 "point"))))
;   (command "erase" pt )

;   (load "centroid")
;   (command "reset")
   (setq ce (getvar "cmdecho"))
   (setvar "cmdecho" 0)
   (command "snap" "off")
   (COMMAND "OSNAP" "NONE")
   (setq a (shape-select))
   (setq initshp (car a))
   (setq initarea (car (cadr a)))
   (setq finlshp (caddr a))
   (setq finlarea (car (cadddr a)))
;   (setq initrot (rot-dir initshp))
;   (setq finlrot (rot-dir finlshp))
;  re-order initshp to start with standard point (left-top)

   (setq initminpt (find-minpt initshp))
   (if (> (length initminpt) 1 )
        (setq initminpt (find-ymax initminpt))
        )       ;endif
   (setq initshp (startwith initshp initminpt))

;  re-order finlshp to start with standard point

   (setq finlminpt (find-minpt finlshp))
   (if (> (length finlminpt) 1 )
      (setq finlminpt (find-ymax finlminpt))
      )  ;endif
   (setq finlshp (startwith finlshp finlminpt))
   (setq initshp (arc2line initshp))
   (setq finlshp (arc2line finlshp))
   (setmap)
   (setvar "cmdecho" ce)
   (command "redraw")
   );end program


(defun setmap()
   (setq mlist (callmap initshp finlshp initarea finlarea))

   )  ;end program


;*************************************************************************
;           PROGRAM SHAPE-SELECT
;*************************************************************************
;
;     By:  Chris Fischer
;          (based in part on a similar program by Anbu Rathinavel)
;
;     Prompts user to select initial and final shapes for mapping and slab
;  analysis routines, calculates the area of the shapes, and returns a list
;  of the component elements in consecutive order.  The format of the list
```

```
;  is ((list of entity names in starting shape) (area of starting shape)
;       (list of entity names in final shape) (area of final shape))

(defun shape-select ()
;     Prompt the user to select the initial and final shapes with a window
   (prompt "\nSelect the starting shape with a window ")
   (setq initshp (ssget "w" (setq crnr (getpoint)) (getcorner crnr)))
   (prompt "\nSelect the final shape with a window ")
   (setq finlshp (ssget "w" (setq crnr (getpoint)) (getcorner crnr)))
   ;
   ;get original layer names of the shapes:
   (setq first (entget (ssname initshp 0))); get first entity of initshp
   (setq lay1 (cdr (assoc 8 first ))); get current layer name of initshp
   (setq first (entget (ssname finlshp 0))); get first entity of finlshp
   (setq lay2 (cdr (assoc 8 first ))); get current layer name of finlshap

;     Store the current layer, then move the selected shapes to new layers
;  for easier manipulation
   (setq orig-layer (getvar "clayer"))
   (command "layer" "make" "ouinit" "make" "oufinl" "")
   (command "layer" "set" orig-layer "")
   (command "change" initshp "" "prop" "layer" "ouinit" "")
   (command "change" finlshp "" "prop" "layer" "oufinl" "")

;     Re-order the elements so they are in consecutive order.  Calculate
;  the area of the shape at the same time.
   (setq ret-list (re-order initshp "ouinit" lay1))
   (setq initshp (car ret-list))
   (setq initarea (cdr ret-list))

   (setq ret-list (re-order finlshp "oufinl" lay2))
   (setq finlshp (car ret-list))
   (setq finlarea (cdr ret-list))
   ;     Assemble the values into a list and return them to the calling routine

   (setq return (list initshp initarea finlshp finlarea))
)     ;end program

;*****************************************************************************
;              PROGRAM RE-ORDER
;*****************************************************************************
;
;     Called by select;  into consecutive order.
;
(defun re-order (shape layername layn)

   (setq s shape l layername)      ;for tracing variables during debug

   (setq elast (entlast))           ;stores the name of the current last
                                    ;element in the database.  Elements
                                    ;generated in this routine will be added
                                    ;at the end of the database, so this
                                    ;provides a starting point for accesing
                                    ;them.
;  Converting the element to a polyline allows the area to be calculated,
;and ensures that the elements will be in consecutive order when it is
;exploded

   (setq index 0)
   (setq pline nil)
   (while (and (< index (sslength shape)) (/= pline 'T))
      (setq etype (cdr (assoc 0 (entget (ssname shape index)))))
      (if (= etype "POLYLINE")
         (setq pline 'T)
         )  ; endif
      (setq index (1+ index))
      )  ;end while

   (if pline
```

```
      (command "pedit" shape "j" shape "" "x")
   ;else
      (command "pedit" shape "y" "j" shape "" "x")
      )  ;endif

   (setq pshape (ssget "x" (list (cons 8 layername)))) ;this is done in 2
   (command "area" "entity" pshape)        ;lines to keep the code simple.
   (setq shparea (getvar "area"))
   (command "explode" pshape)
   (setq pshape (ssget "x" (list (cons 8 layername))))
   (setq shape nil)
   (setq temp elast)
   (while (setq temp (entnext temp))
     ; (command "change" temp "" "prop" "layer" (read (getvar "clayer")) "")
      (setq shape (append shape (list temp)))
      )  ;end while
   (command "change" pshape "" "prop" "layer" layn ""); return shape to its
                                                      ;original layer
   (setq return (list shape shparea))
);end program

;*************************************************************************
;        PROGRAM ROT-DIR
;*************************************************************************
;
;  BY Chris Fischer  Jan 31, 1992
;  Detrmines rotation direction of consecutive elements in an exploded
;  polyline
;
(defun rot-dir (entlist)

;  select the shape in  the layer "layername"
   (setq intlist nil)
   (setq layername (cdr (assoc 8 (entget (car entlist)))))
   (setq selset (ssget "x" (list (cons 8 layername))))
   (setq centr (centroid selset))
   (command "layer" "on" "*" "")
   (setq p2 (list (car centr) (+ (cadr centr) 50))) ; a point directly
                                                    ;  above the centroid
   (while (setq current (car entlist))
      (setq entlist (cdr entlist))
      (setq p3 (cdr (assoc 10 (entget current))))
      (setq p4 (cdr (assoc 11 (entget current))))
      (setq intr (inters centr p2 p3 p4))
      (if intr
         (setq intlist (append intlist (list current intr)))
         )  ;  end if
      )  ;end while

   (if (> (length intlist) 2)
      (progn
      (setq y1 (cadr (cadr intlist)))
      (setq y2 (cadr (cadddr intlist)))
      (if (> y1 y2 )
         (setq ent (car intlist))
         (setq ent (caddr intlist))
         )    ;   end if
      )    ;end progn
      (setq ent (car intlist))
      )    ; end if

   (setq p1 (cadr (assoc 10 (entget ent))))
   (setq p2 (cadr (assoc 11 (entget ent))))
   (if (> p1 p2)
      (setq dir 'ccw)
      (setq dir 'cw)
      )  ;end if
   )  ;end function
```

```
;*************************************************************************
;        FUNCTION CALLMAP
;*************************************************************************
;
;     Calls the mapping routine to map the initial shape to the final shape,
;  then the final shape to the initial shape.  Returns a list of associated
;  initial and final x coordinates of vertical lines for mapping.

(defun callmap (initshp finlshp initarea finlarea)
;
   (setq cecho (getvar "cmdecho"))
   (setvar "cmdecho" 0)

;    For debug:  open a file to write execution time data
   (setq fp (open "timedata" "w"))

;     Recalculate initial and final areas after straight line approximation
;  of arcs.
   (setq initxmax (findxmax initshp))
   (setq finlxmax (findxmax finlshp))
;   (textscr)
   (setq ents (idents initshp initxmax))
   (setq initarea (arealeft initshp (cadr ents) (car ents) initxmax))
   (setq ents (idents finlshp finlxmax))
   (setq finlarea (arealeft finlshp (cadr ents) (car ents) finlxmax))

   (setq list1 nil)
   (setq list2 nil)
;     Map the initial shape to the final shape.  Returns two lists.  The
;  first contains the vertical line divisions in the initial shape.  The
;  second contains the vertical line divisions in the final shape.
;
   ;first, reset global variables used in matcharea function
   (setq oldx nil)
   (setq oldarea nil)
   (setq dxda nil)

   ;continue with mapping function
   (setq ret-list (map initshp finlshp initarea finlarea))
   (setq initlist (car ret-list))
   (setq finllist (cadr ret-list))
   (while (setq initx (car initlist))
      (setq finlx (car finllist))
      (setq list1 (append list1 (list (list initx finlx))))
      (setq initlist (cdr initlist))
      (setq finllist (cdr finllist))
      )  ;  end while

;     Map the final shape to the initial shape.  In this case, the first
;  list contains the divisions of the final shape, and the second contains
;  the division of the initial shape.  They are reversed, then assembled.

   ;reset global variables
   (setq oldx nil)
   (setq oldarea nil)
   (setq dxda nil)

   ;continue with mapping
   (setq ret-list (map finlshp initshp finlarea initarea))
   (setq initlist (cadr ret-list))        ;note these two lines are the
   (setq finllist (car ret-list))         ;reverse of those in the above
   (while (setq initx (car initlist))                  ;block
      (setq finlx (car finllist))
      (setq list2 (append list2 (list (list initx finlx))))
      (setq initlist (cdr initlist))
      (setq finllist (cdr finllist))
      )  ;  end while

;  list1 and list2 are assembled into a single list, and returned to the
```

```
;  calling program.

   (setq assembly (list (list (cadr (assoc 10 (entget (car initshp))))
                              (cadr (assoc 10 (entget (car finlshp))))
                              )))     ;start with the first x point in
                                      ;each list

   (while (or (setq x1 (car list1)) (car list2))

            (setq x2 (car list2))    ;repeated because the second item in
                                     ;an or list may not be evaluated

      (if ( and (or (< (car x1) (car x2)) (not x2)) x1) (progn
               ;IN ENGLISH:  [(x1 < x2) or (not x2)] and x1
         (setq assembly (append assembly (list x1)))
         (setq list1 (cdr list1))
         ));end progn, end if
      ;else
      (if (and (or (> (car x1) (car x2)) (not x1)) x2) (progn
               ; [(x1 > x2) or (not x1)] and x2
         (setq assembly (append assembly (list x2)))
         (setq list2 (cdr list2))
         ))  ;end progn, end if
      ;else  {if they are equal, dump the element from the first list
               ;  and continue with the next element from each list}
      (if (= (car x1) (car x2))
         (setq list1 (cdr list1))
         )  ;end if
      )  ;end while

   (setvar "cmdecho" cecho)
   (setq return assembly)
   )  ;end function

;*************************************************************************
;        FUNCTION MAP
;*************************************************************************
;
;     This function is called with a starting and ending shape.  The
;  starting shape is scanned left to right.  Each time an entity endpoint
;  is encountered, a vertical line is drawn, and the area to the left of
;  that line is calculated.  A vertical line is then located on the
;  final shape partitioning an area of equivalent ratio.  These x
;  coordinates are stored in list1 and list2, and returned to the
;  calling routine.

(defun map (startshp endshp startarea endarea)
   (setq startlist nil)
   (setq endlist nil)
;  calculate area ratio
   (setq ratio (/ endarea startarea))

   (setq nents (length startshp))
   (setq loindex 1)
   (setq hiindex nents)
   (setq entmin (readelem startshp 1))
   (setq xmin (cadr (assoc 10 (entget entmin))))

   (while (/= loindex hiindex)
      (setq entlo (readelem startshp loindex))  ;entlo and enthi are the low
      (setq enthi (readelem startshp hiindex))  ;and high entities, respectively
      (setq xlo (cadr (assoc 11 (entget entlo))))
      (setq xhi (cadr (assoc 10 (entget enthi))))
      (if (< xlo xhi) (progn
         (if (/= xlo xmin) (progn
            (setq area1 (arealeft startshp hiindex loindex xlo))
            (setq area2 (* area1 ratio))
            (setq endx (matcharea endshp area2))
```

```
            (setq startlist (append startlist (list xlo)))
            (setq endlist (append endlist (list endx)))
            ));end progn, endif {/= xlo}
         (setq loindex (1+ loindex))
         )  ;  end progn
      ;else
      (progn
         (if (/= xhi xmin) (progn
            (setq area1 (arealeft startshp hiindex loindex xhi))
            (setq area2 (* area1 ratio))
            (setq endx (matcharea endshp area2))
            (setq startlist (append startlist (list xhi)))
            (setq endlist (append endlist (list endx)))
            ));   end progn, endif {/= xhi}
         (setq hiindex (1- hiindex))
         )  ;  end progn
         )  ;  end if

      )  ;end while

   (setq return (list startlist endlist))
   )  ;end function

;*************************************************************************
;           FUNCTION MATCHAREA
;*************************************************************************
;     Iterates on shape to find the x position of a vertical line with the
;  specified area to the left of it.
;     New version of the function, uses faster iteration scheme and values
;  stored in global variables to achieve faster convergence
;
;     Global variables:  OLDX, OLDAREA, DXDA
;
(defun matcharea (shape targetarea)
   (setq sh shape)
   (setq xmax (findxmax shape))
   (setq xmin (cadr (assoc 10 (entget (car shape)))))
   (if (not oldx) (setq oldx xmin))
   (if (not oldarea) (setq oldarea 0.0))
   (if (not dxda) (setq dxda 1.0))
   (setq newx oldx)
   (while (> (abs (- oldarea targetarea)) 0.001)
      (setq newx (+ (* (- targetarea oldarea) dxda) oldx))
      (if (> newx xmax) (setq newx xmax))
      (if (< newx xmin) (setq newx xmin))
      (setq ents (idents shape newx))
      (setq currentarea (arealeft shape (cadr ents) (car ents) newx))
      (if (> (abs (- currentarea oldarea)) 0.001)
         (setq dxda (/ (- newx oldx) (- currentarea oldarea))))
      (setq oldarea currentarea)
      (setq oldx newx)
      )  ;end while
   (setq return newx)
   )  ;end progn


;*************************************************************************
;        FUNCTION OLDMATCHAREA
;*************************************************************************
;
;     Iterates on shape to find the x position of a vertical line with the
;  specified area to the left of it.
;     This function is renamed from matcharea, and is not called.  It is
;  an old version of the program
;(defun matcharea (shape area)
(defun oldmatcharea (shape area)
   (setq ar area)
   ;find minimum x value of the shape
```

```
   (command "undo" "mark")
   (command "undo" "mark")
   (setq x (cadr (assoc 10 (entget (car shape)))))
   ;find approx area of vertical line by incrementing 1 inch at a time
   ;to the right
   (setq calcarea 0)
   (setq ents 't)
   (while (and (> (- area calcarea) 0.0005) ents)
      (setq x (1+ x))
      (setq ents (idents  shape x))   ;identify the entities at the specified
      (if ents (progn
         (setq entlo (car ents))  ;x coordinate
         (setq enthi (cadr ents))
         (setq calcarea (arealeft shape enthi entlo x))
         )); end progn, end if
      )  ;end while

   (setq delarea (abs (- calcarea area)))
   (setq delx 1.0)
   (prompt "\n(starting second phase of matcharea")
;iterate:  calculate area, compare to target area, and narrow the window
;by 1/2 for each step.
   (while (> delarea 0.0005)
      (setq delx (/ delx 2.0))
      (setq ents (idents shape x))
      (if ents (progn
         (setq entlo (car ents))
         (setq enthi (cadr ents))
         (setq calcarea (arealeft shape enthi entlo x))
         (setq delarea (abs (- calcarea area)))
         (if ( > delarea 0.0005) (progn
            (if (< calcarea area)
               (setq x (+ x delx))
               ;else  {calcarea}
               (setq x (- x delx))
               )  ;end if {calcarea}
            ))  ;end progn, end if {delarea}
         );end progn
         ;else {ents}
         (setq x (- x delx))
         )  ;end if {ents}
         )  ;end while
   (prompt "\nexiting second phase of matcharea")

   (command "undo" "end")
   (command "undo" "back" )
;   (command "redraw")
   (setq return x)
   )  ;end function

;*************************************************************************
;        FUNCTION INTER-SHP
;*************************************************************************
;
;     Generates the intermediate shape from a list of coresponding
;  points on the initial and final shapes.
(defun inter-shp (initshp finlshp maplist)

;  Define reference points for the initial, final, and mapped shapes.
;  Initial and final reference points are the first endpoint of the
;  first entity in the list.  The reference point for the mapped shape
;  is selected by the user

   (prompt "\nSelect the location for the left end of the mapped product")

   (setq mref (getpoint))           ;Mapped shape REFerence point
   (setq mrefx (car mref))          ;Mapped shape REFerence point X coord
   (setq mrefy (cadr mref))         ;Mapped shape REFerence point Y coord
   (setq iref (cdr (assoc 10 (entget (car initshp)))));Initial shape REFerence
```

```
   (setq fref (cdr (assoc 10 (entget (car finlshp)))));Final shape REFerence
   (setq irefx (car iref))          ;Initial REFerence X coord
   (setq irefy (cadr iref))         ;Initial REFerence Y coord
   (setq frefx (car fref))          ;Final REFerence X coord
   (setq frefy (cadr fref))         ;Final REFernece Y coord
;************************DEBUG
   (prompt "\nEnter the mapping ratio factor (0 to 1)")
   (while (or (< (setq n (getreal)) 0) (> n 1))
      (prompt "\nThe value must be between 0 and 1. Reenter.")
      )  ;end while

   (graphscr)

;In a loop, read each pair of points, then generate the mapped shape

   (setq npts (length maplist))
   (setq count 1)
;*********************************DEBUG
   (while (<= count npts)
  ;  (while (<= count 2)
;***********************************
      (setq temp (car maplist))
      (setq maplist (cdr maplist))
      (setq ix (car temp))      ; initial x map point (local coords)
      (setq fx (cadr temp))     ; final x map point (local coords)

   ;initial shape
      (setq ients (idents initshp ix))   ; idents uses global coords
   ;lower entity index
      (setq iloent (car ients))
      (setq workent (readelem initshp iloent))
      (setq pt1 (cdr (assoc 10 (entget workent))))
      (setq pt2 (cdr (assoc 11 (entget workent))))
      (setq xpt1 (list ix '0))   ;  these two points form a vertical
      (setq xpt2 (list ix '1))   ;line through the x map point
      (setq ilo (inters pt1 pt2 xpt1 xpt2 nil))
      (setq iloy (cadr ilo))

   ;higher entity index
      (setq ihient (cadr ients))
      (setq workent (readelem initshp ihient))
      (setq pt1 (cdr (assoc 10 (entget workent))))
      (setq pt2 (cdr (assoc 11 (entget workent))))
      (setq ihi (inters pt1 pt2 xpt1 xpt2 nil))
      (setq ihiy (cadr ihi))

   ;final shape
      (setq fents (idents finlshp fx))
   ;lower entity index
      (setq floent (car fents))
      (setq workent (readelem finlshp floent))
      (setq pt1 (cdr (assoc 10 (entget workent))))
      (setq pt2 (cdr (assoc 11 (entget workent))))
      (setq xpt1 (list fx  '0))
      (setq xpt2 (list fx  '1))
      (setq flo (inters pt1 pt2 xpt1 xpt2 nil))
      (setq floy (cadr flo))

   ;higher entity index
      (setq fhient (cadr fents))
      (setq workent (readelem finlshp fhient ))
      (setq pt1 (cdr (assoc 10 (entget workent))))
      (setq pt2 (cdr (assoc 11 (entget workent))))
      (setq fhi (inters pt1 pt2 xpt1 xpt2 nil))
      (setq fhiy (cadr fhi))

   ;sort hi and lo y values
      (setq ylist (sort2 floy fhiy))
      (setq floy (car ylist))
```

```
      (setq fhiy (cadr ylist))
      (setq ylist (sort2 iloy ihiy))
      (setq iloy (car ylist))
      (setq ihiy (cadr ylist))

   ;calculate mapped points

   ;  calculate the distance from the initial and final reference points to
   ;     the current points.
      (setq dix (- ix irefx))    ; Delta I X -- distance from IX to IREFX
      (setq dfx (- fx frefx))    ; Delta F X -- distance from FX to FREFX
      (setq diloy (- iloy irefy))
      (setq dihiy (- ihiy irefy))
      (setq dfloy (- floy frefy))
      (setq dfhiy (- fhiy frefy))

   ;  calculate mapped points as a scaled ratio of DeltaI and DeltaF for each
   ;     point.

      (setq dmx (+ (* (- 1 n) (- dix dfx)) dfx))   ;  dm = (1-n)(di - df) + df
      (setq mx (+ dmx mrefx))
      (setq dmloy (+ (* ( - 1 n) (- diloy dfloy)) dfloy))
      (setq dmhiy (+ (* (- 1 n) (- dihiy dfhiy)) dfhiy))
      (setq mloy (+ dmloy mrefy))
      (setq mhiy (+ dmhiy mrefy))
      (setq mlo (list mx mloy))
      (setq mhi (list mx mhiy))

;  Now draw the lines.

      (if (= count 1)
         (progn
         (command "line" mlo mhi "")
         (setq oldmlo mlo)
         (setq oldmhi mhi)
         ) ;  end progn
      ;else
        (progn
        (if (not (pt= oldmlo mlo))
            (progn
            (command "line" oldmlo mlo "")
            (setq oldmlo mlo)
            )) ; end progn, endif {not pt= oldmlo}
        (if (not (pt= oldmhi mhi))
            (progn
            (command "line" oldmhi mhi "")
            (setq oldmhi mhi)
            )) ; end progn, endif {not pt= oldmhi}
         )) ;end progn end if

      (if (= count npts)
         (command "line" mlo mhi "")
         ) ;end if
      (SETQ OL OLDMLO)
      (SETQ OH OLDMHI)
      (setq count (1+ count))
 ;*******************WHILE LOOP COMMENTED OUT FOR DEBUGGING
      )  ;end while
)  ;end function

;*****************************************************************************
;     function SORT2
;*****************************************************************************
;     sorts two values, and returns a list containing the lower one first

(defun sort2 (i j)
   (if (< j i)
      (progn
      (setq t j)
```

```
      (setq j i)
      (setq i t)
      ))  ;end progn, end if
   (setq return (list i j))
   ) ;end function

(defun c:is() (inter-shp initshp finlshp mlist))

;*************************************************************************
;           FUNCTION FINDXMAX
;*************************************************************************
;     returns the largest x coordinate found in ENTLIST
;
(defun findxmax (entlist)
   (setq xmax -1.0e10)
   (while (setq temp (car entlist))
      (setq entlist (cdr entlist))
      (setq xtemp (cadr (assoc 10 (entget temp))))

      (if (> xtemp xmax)
         (setq xmax xtemp)
         )  ;end if
      )  ;end while
   (setq return xmax)
   )  ; end function
```

```
;*************************************************************************
; THICKNESS MAPPING PROGRAM                                by Farid Masri
;*************************************************************************
;
;  this program will map a selected shape and draw thickness profile
;  of it. the program is first loaded by using the autocad "load" command
;  (load "thickmap"), then it can be used as an autocad command
;  ( "thickmap"). the progarm will then ask the user to select
;  the desired shape with a window. after the selection, the progarm
;  will then ask the user for a point on the screen; this will be the
;  starting point of the new drawing that's generated by the porgram.
;
;*************************************************************************
;*************************************************************************
  (defun c:thickmap ()
;*************************************************************************
;
;  get current  snap and color settings and save them. these variables are
;  changed during execution and will be restored at the end of the progarm:
;
  (setq org-color (getvar "cecolor")); get original color.
  (setq org-osnap (getvar "osmode"))
  (setq org-snap (getvar "snapmode"))
  (setq org-cmdecho (getvar "cmdecho"))
  (setq cl (getvar "celayer"))
;
  (setvar "cmdecho" 0); switch off command echo for faster execution
  (command "osnap" "none"); turn off all snap modes for proper execution
  (command "snap" "off")
;
;
  (prompt "\nselect the desired shape to be mapped with a window")
  (princ "\n \n ")(princ)
  (setq entlist (car (shapesel)))
  (command "undo" "end")
  (prompt "\nselect starting point of mapped shape")
  (princ "\n \n") (princ)
  (setq xstart (getpoint))
  (prompt "\nrunning..... please wait")
  (princ "\n \n") (princ)
  (setq entlist (arc2line entlist)); convert arcs in shape to line segments
  (setq xmin (x-min entlist)) ; fin minimum and maximum cordinates of shape
  (setq xmax (x-max entlist))
  (setq ymin (y-min entlist))
  (setq ymax (y-max entlist))

  (setq xstart (list (car xstart)(cadr xstart)));
  (setq xend (list (+ (- xmax xmin)(car xstart)) (car (cdr xstart))))
  (command "line" xstart xend ""); draw baseline of thikness profile
  (setq xpos (+ xmin 0.005)); shift the current position pointer slightly
  (setq tempp xstart)
  (setq tmplist entlist); tmplist is used by midents

      (linter xpos)
      (setq ylength (abs (- (cadr inters1) (cadr inters2))))
      (setq npoint (list (+ (car xstart)(- x1 xmin)) (+ (cadr xstart) ylength)))
     (command "line" tempp npoint "")
      (setq tempp npoint)

  (while (<= xpos xmax)
      (linter xpos)
      (setq ylength (abs (- (cadr inters3)( cadr inters4))))
      (setq npoint (list (+ (car xstart)(- x2 xmin)) (+ (cadr xstart) ylength)))
      (command "line" tempp npoint "")
      (setq xpos ( + x2 0.005))
      (setq tempp npoint)
  ); end while
  (command "line" npoint xend ""); join the last point with the base line
;
```

```
  (command "redraw") ; clean up the window
;
   (setq tmplist nil  entlist nil shape nil pshape nil)
;  restore original settings
  (setvar "osmode" org-osnap)
  (setvar "snapmode" org-snap)
  (command "color" org-color)
  (command "layer" "set" cl "")
  (setvar "cmdecho" org-cmdecho)
  (princ "\n \n")
  (princ)
  ); end program thickmap
;


;*************************************************************************
;   program midents                                          by Farid Masri
;*************************************************************************
;   modified version of program idents  by chris fischer
;   identify entities:  identify indices of lower and upper entities
;   intersecting a vertical line through xpoint.  returns nil if xpoint
;   is outside of the shape.
;   by chris fischer.
   (defun midents (tmplist xpoint)
   (setq ents nil)
   (setq index 1)
   (while ( < (length ents) 2)
      (setq temp (car tmplist))
      (setq tmplist (cdr tmplist))
      (setq x1 (cadr (assoc 10 (entget temp))))
      (setq x2 (cadr (assoc 11 (entget temp))))

    ;sort x1 and x2
      (if (< x2 x1) (progn
         (setq t x2)
         (setq x2 x1)
         (setq x1 t)
      ))    ; end progn, endif

;    test if xpoint is between the endpoints of the entity
;    and be sure the entity is not vertical and is not a duplicate.
            (if (and (<= x1 xpoint x2) (/= x1 x2) ( /= temp (car ents)) )
               (setq ents (append ents (list temp)))
            ); end if

         (setq tmplist  (append tmplist (list temp)));

     )  ;end while

;    select the first and last entities in the list

    (if (/= (car ents) nil)
      (setq return ents)
      ; else
      (setq return nil)
     )  ;endif

   )  ;end function


;*************************************************************************
;  function linter (line intersection)                    by Farid Masri
;
;  this function reurns the intersection points of a vertical line
;  at a given x position and the entities that it intersects.
;
;*************************************************************************

   (defun linter (xp / xpo pt1 pt2 pt3 pt4 pt5 pt6 pttemp)
```

```
(setq xpo xp)
(setq xp (midents tmplist xp))
(setq a  (car ents))
(setq b (cadr ents))
;
;
(setq pt1 (list (cadr(assoc 10 (entget a)))(caddr (assoc 10 (entget a)))))
(setq pt2 (list (cadr(assoc 11 (entget a)))(caddr (assoc 11 (entget a)))))
(setq pt3 (list (cadr(assoc 10 (entget b)))(caddr (assoc 10 (entget b)))))
(setq pt4 (list (cadr(assoc 11 (entget b)))(caddr (assoc 11 (entget b)))))
   (if ( < (car pt2) (car pt1)); if pt2 is less than pt1 then exchange
   ;            them in order to find the minimum x-coord. of the line
      (progn
         (setq pttemp pt1)
         (setq pt1 pt2)
         (setq pt2 pttemp)
      ); end progn
   );end if
   ;
   (if (< (car pt4) (car pt3)); if pt4 is less than pt3 then exchange them
      (progn
         (setq pttemp pt3)
         (setq pt3 pt4)
         (setq pt4 pttemp)
      ); end progn
   ); end if
   (setq x1 (car pt1))
   (setq x2  (car pt2))
   (if  ( <  ( car pt1) (car pt3))
     ( setq x1 (car pt3))
  ); end if
  (if ( <  ( car pt4) (car pt2))
     (setq x2 (car pt4))
 ); end if
(setq pt5 (list x1 ymin))
(setq pt6 (list x1 ymax))
(setq inters1 (inters pt1 pt2 pt5 pt6 nil))
(setq inters2 (inters pt3 pt4 pt5 pt6 nil))
(setq pt5 (list x2 ymin))
(setq pt6 (list x2 ymax))
(setq inters3 (inters pt1 pt2 pt5 pt6 nil))
(setq inters4 (inters pt3 pt4 pt5 pt6 nil))
); end function
```