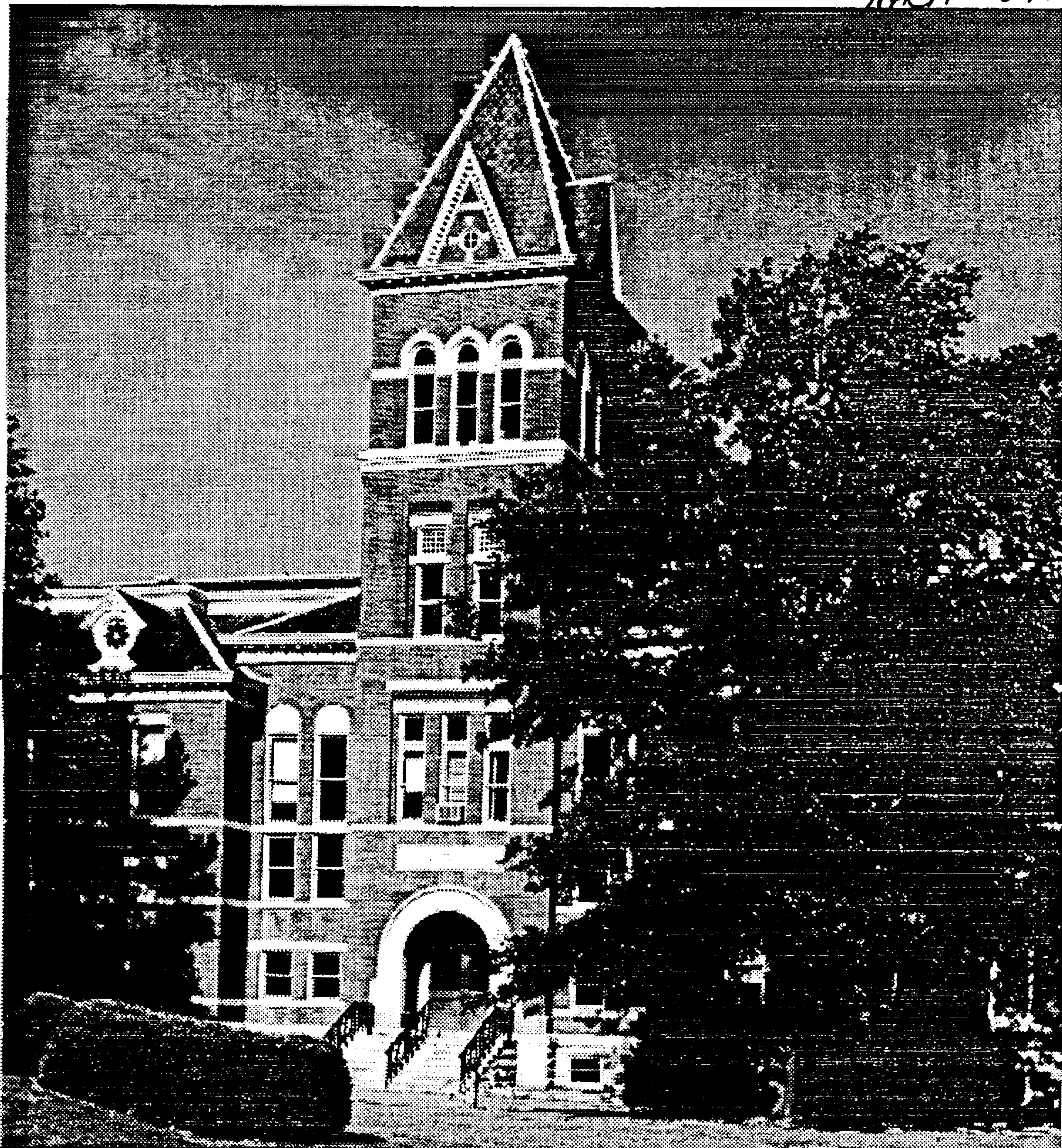
# University of Missouri–Columbia

## College of Engineering

NAG1-875



(NASA-CR-185838) ENGINEERING SPECIFICATION AND SYSTEM DESIGN FOR CAD/CAM OF CUSTOM SHOES. PHASE 5: UMC INVOLVEMENT (JANUARY 1, 1989 - JUNE 30, 1989) (Missouri Univ.) 50 p

N90-11456

CSCL 09B G3/61

Unclas

0224562

# ENGINEERING SPECIFICATION AND SYSTEM DESIGN FOR CAD/CAM OF CUSTOM SHOES

PROJECT NAG-1-875

PHASE V - UMC INVOLVEMENT
( JANUARY 1, 1989 - JUNE 30, 1989 )

A REPORT SUBMITTED TO

NATIONAL AERONAUTICS & SPACE ADMINISTRATION
LANGLEY RESEARCH CENTER, HAMPTON, VIRGINIA

BY

HAN P. BAO , PHD., PE
UNIVERSITY OF MISSOURI-COLUMBIA
COLUMBIA, MISSOURI 65211

Technical report UMC-IE-4-0889
August 1989

# CONTENTS

1 - Summary

2 - Machining of Shoe Last - Point-To-Point Configuration

2.1 - CENCIT Data

2.2 - CYBERWARE Data

3 - Machining of Shoe Last - Patch Configuration

3.1 - Review of NCSU Work

3.2 - Software Development at UMC

4 - Design and Production of Integrated Sole

4.1 - Clinical Assessment

4.2 - Sole Production Technique

4.3 - Software related to sole

APPENDICES

1- Listings of computer programs for machining surface patches

2- Listings of computer programs for machining mold for casting integrated sole

## 1 - SUMMARY


The work involved in the previous phase, phase IV May 15, 1988 to December 30, 1988 , was primarily of a theoretical nature paving the way to actual machining and casting experiments reported in this document for Phase V work.

The following topics are discussed in this report :

1- Machining of shoe last - point-to-point configuration

2- Machining of shoe last - patch configuration

3- Design and production of integrated sole

### 1- Machining of shoe last - point-to-point configuration

The solid object for machining is represented by a wireframe model with its nodes or vertices specified systematically in a grid pattern covering its entire length.

Two sets of data , respectively from CENCIT and CYBERWARE, were used for machining purpose. The machining process itself has been experimented with, using a variety of approaches as suggested in the theoretical work carried out in the previous phase.

It has been found that the indexing technique, that is turning the stock by a small angle then move the tool on a longitudinal path along the foot, yields the best result in terms of ease of programming, saving in wear and tear of machine and cutting tools, and resolution of fine surface details.

### 2- Machining of shoe last - patch configuration

The work of Dr. McAllister and his group at NCSU through the LASTMOD last design system results in a shoe last specified by a number of congruent surface patches of different sizes. This data format must therefore be adopted to carry out the downstream operation of last machining.

In this report, a means of converting this data into a form amenable to the machine tool is provided. Essentially it involves a series of sorting algorithms and interpolation algorithms to provide the grid pattern that the machine tool needs as was the case in a point-to-point configuration discussed in section 1.

The resulting machined foot agrees quite well with the one obtained by Dr. Sanii of NCSU although both of us were surprised with the actual length of the object, it being about seven inches long as compared to nine or ten inches of expected length.

3- Design and production of integrated sole

Although the design and manufacture of a shoe last is the single most important element in shoe making, many other activities also play an important role in the entire spectrum of footwear production. Examples of these activities include the prescription of different types of wedges, the molding of the inner, middle, and outer soles, the use of inserts for arch, heel, or metatarsal support, the use of rigid to soft orthotics and, in general, the use of extra-depth shoes.

This report contains an in-depth treatment of the design and production technique of an integrated sole to complement the task of design and manufacture of the shoe last. Clinical data and essential production parameters are also discussed. Examples of soles made through this process are given as illustrations.

## Summary

Because the grant was given to UMC very late, only a month before the official end of the phase period, work on orthotic devices such as pads, wedges, and inserts could not be made in time for reporting in this document. Nevertheless the work done in this phase V of the project reported here provides valuable practical solutions to the theoretical propositions laid down in the previous phase IV. They facilitate the proposed next phase work for the rest of this year.

## 2 - MACHINING OF SHOE LAST - POINT-TO-POINT CONFIGURATION

Two sets of foot data were used in this phase of the project for purpose of machining : the CENCIT (#1) data set and the CYBERWARE (#2) data set. Both of these companies specialize in 3-D digitization technology.

### 2.1 CENCIT DATA SET

The scanning technology used by CENCIT is one of photochemical light beam profilometry using up to six strobed light projectors and up to six solid-state video cameras. The published resolution is +/- 1 mm with an accuracy of +/- 0.5% of field of view. The scanning time is very short, usually less than one second for a complete 360 degree scan around the solid object.

Currently the scanning system is used for portrait sculpture, however the company has agreed to do a scan of one of our plaster foot casts. Figure 1 indicates the set-up used to scan the foot cast. Note that the final result is in terms of three views containing a maximum of 51,200 point coordinates ( 200x256 grid ) each. However, after discarding the points that are invalid, that is unmeasurable, the final count is only about 15,333 good data points per view, or 46,000 data points for the whole foot cast. This many data points, although a lot less than the 153,600 points ( 51,200 points per view times 3 views ) that could have been obtained, were considered sufficient for machining without damaging loss of detail.

#### 2.1.1 Transformation of Cencit Data

The data sets provided by CENCIT came in three diskettes respectively for view 1, view 2 , and view 3. View 1 corresponds with the inner bottom of the foot, view 2 with the outer bottom, and view 3 with the top. Because of the size of the data sets, each file must be processed separately. Figure 2 shows the types of transformations required to transform the original data into some useful data for machining.

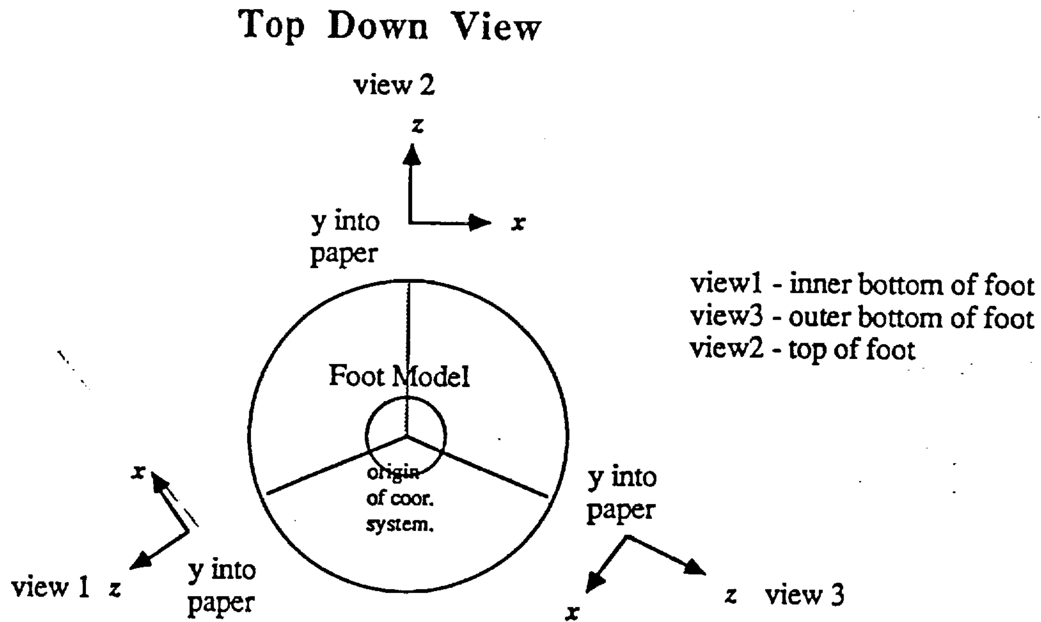
A detailed description of each piece of software is available in the report for phase IV ( Technical Report UMC-IE-3-0189, Feb. 1989 ). The following is a brief review of the software.

FLINT.PAS
The original data sets are in integer format and, furthermore, the bytes (LSB and MSB ) are in reverse order. Flint.Pas was written using Turbo Pascal to flip the low byte and high byte in each integer.

SCTFOOT.PAS
The CENCIT scanner was designed for an object much larger than a foot.

# Foot Scanning (CENCIT Inc.)

## Top Down View




## Grid Setup

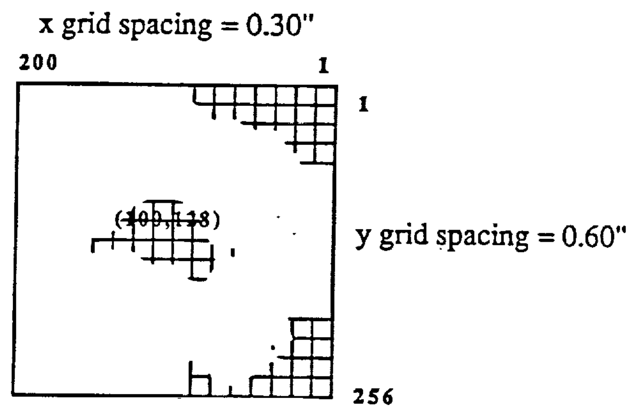



Figure 1 Foot Scanning (CENCIT Inc.)

Therefore it was expected that quite a few data points would be missing from the scan result. These missing points are indicated by a large Z value, ie. Z=-32,000 , in the data sets provided by CENCIT. SCTFOOT.PAS simply removes these invalid or non-exixting points.

PROTY.PAS
This program performs a coordinate transformation to merge all three views into one. Thus, using view 2 as the reference view, the other two views are merged into this one.

RECTCYL.PAS
This Fortran progam transforms the cartesian coordinates into cylindrical coordinates using the standard relations

$$R= SQRT(X^2+Z^2) \quad \text{and } TH= TAN^{-1}(Z/X)$$

SORTY.SAS
A SAS program is used to sort the data obtained from RECTCYL by Y and TH.

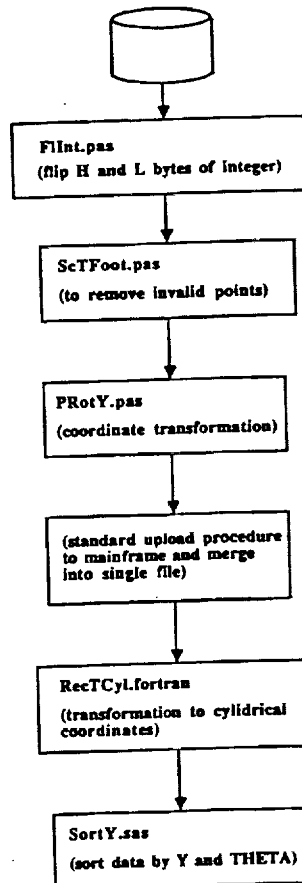



Figure 2 CENCIT Data file manipulation

### 2.1.2 **Machining Activity**

To reproduce the foot cast by machining, two general approaches have been used : the point-to-point configuration and patch configuration. These approaches differ only in the way the solid is specified. Thus, in the former case, a wireframe model is used which is composed of nodes or vertices and links ( straight lines joining 2 consecutive points on a same contour ). In the latter case, the object is represented by a number of curvi-linear 4-side patches arranged in a systematic spatial order.

Irrespective of which method to represent the object, as far as the machine tool is concerned, its tool must be be programmed to move in an euclidean motion, that is point to point, so that inherently the wireframe solid representation is more advantageous than the other type of representation.

Machining equipment

The machine tool used in these experiments was a 4-axis numerical controlled milling machine with built-in RS-232 interface to a personal computer like an IBM PC/XT. Figure 3 shows the machine in the manufacturing laboratory of the University of Missouri-Columbia.

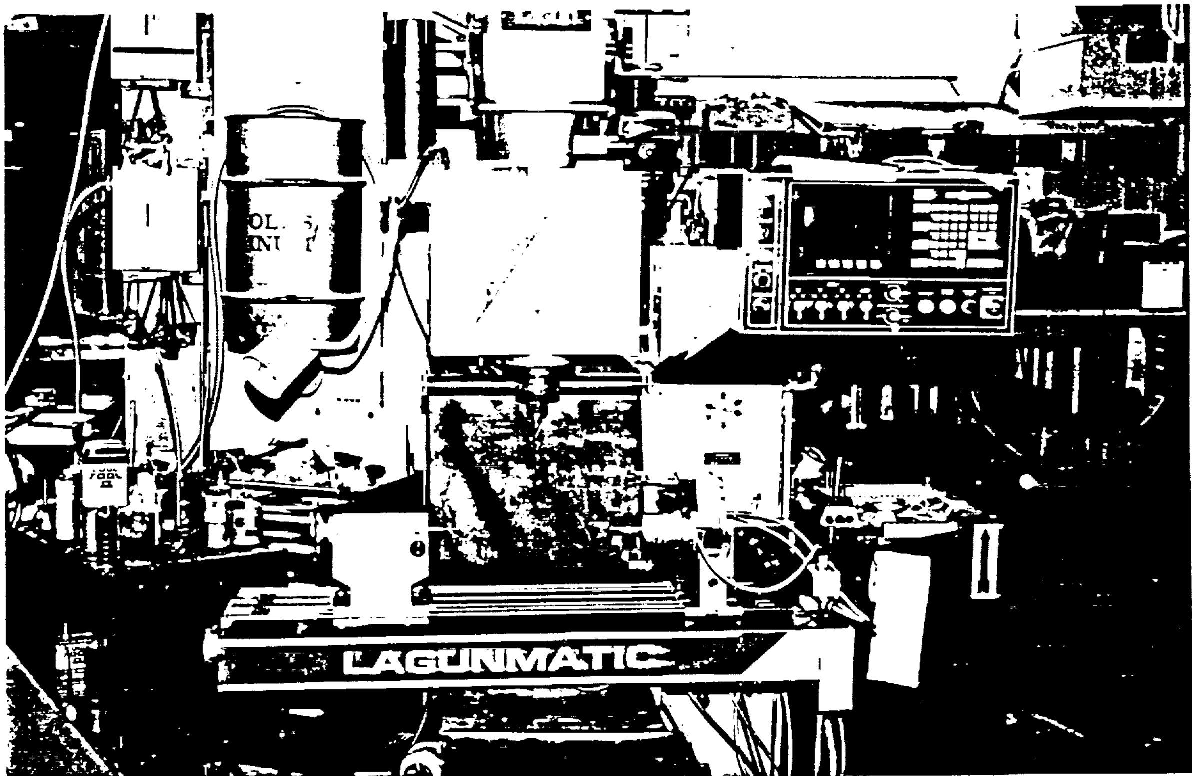



Fig.3 CNC Milling Machine

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

Cylindrical plaster blocks were casted in plastic molds as shown in figure 4. Not shown in this figure are the aluminum inserts at the two ends of the block for holding it between the rotary table and the tail stock holder of the machine tool.

Fig.4 Plastic cylinders for casting plaster blocks

As recommended by practitioners at CENCIT and CYBERWARE, conical high speed steel tools were used. One such tool is shown in figure 5 with a 3/16 in diameter at the small end and a 3 degree taper per side.

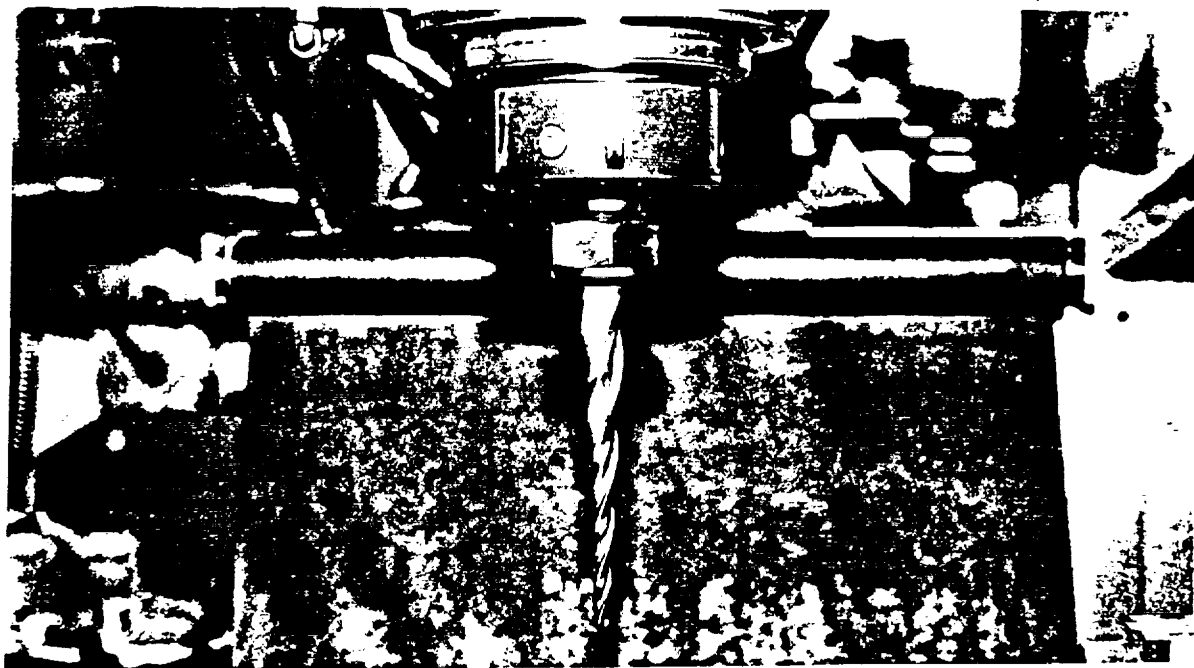
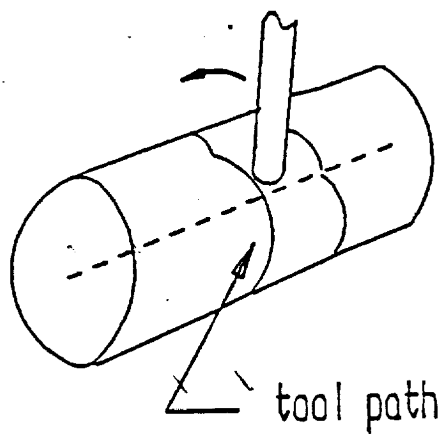
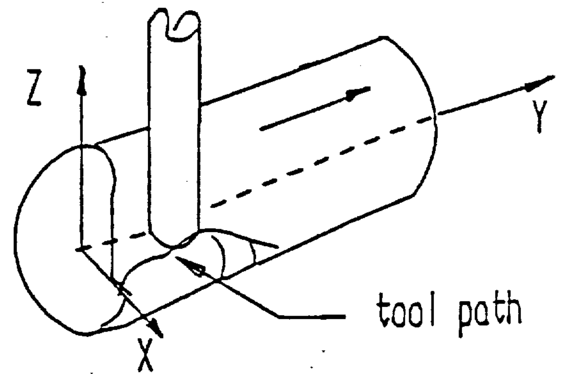


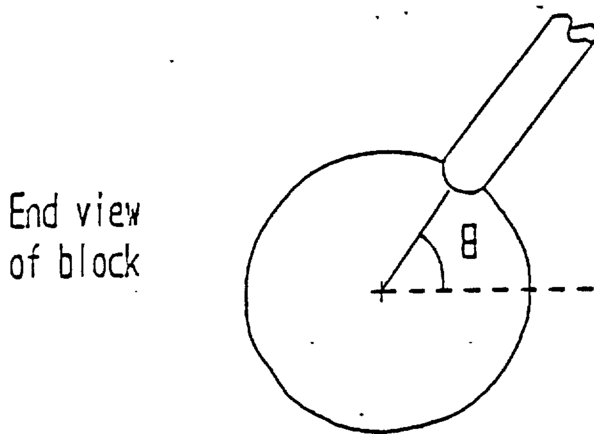
Fig.5 Tapered tool for machining

The feeds and speeds used in the machining experiments were on a trial and error basis because not much published literature is available concerning the machining of plaster.

Theoretical considerations of the various ways of machining a shoe last were treated in the previous report ( Technical Report UMC-IE-3-0189 ). Figure 6 taken from this report illustrates these different ways.



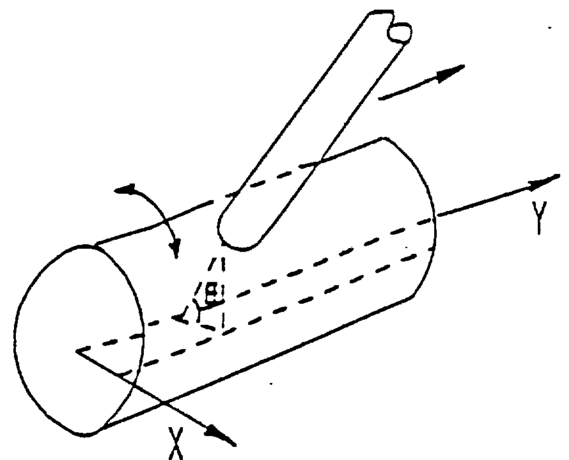




Fig.6 The four ways of machining a shoe last

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

In this phase of the research, actual cuttings were made and comparison made between the various products. The experimental results confirm the theoretical predictions made then, notably :

1- "Contour machining" and "Constant O per pass" are the easiest to program but, in the case of contour machining, due to rotational limitation of the rotary table, the tool must be programmed to go clockwise in one contour then counter-clockwise in the next contour. Due to constant changes of direction, the rotary table may be subject to excessive wear and tear. For this reason, a decision was made not to pursue further this method of machining.

2- For the same reason for the contour machining approach, the "variable O per pass" approach was also discarded not only because of excessive wear and tear of the rotary table but also because of the complexity in calculating the proper delta angle in both direction for the rotary table.

3- The "Cartesian machining" approach is potentially the least cost technique since it may not even require the use of the rotary table. Figure 7 shows a cylindrical plaster block held in a conventional vise and machined accordingly. The major problem for this approach is the increase in depth of ridges, ie rougher steps, as the tool moves further away from its central position atop of the block.

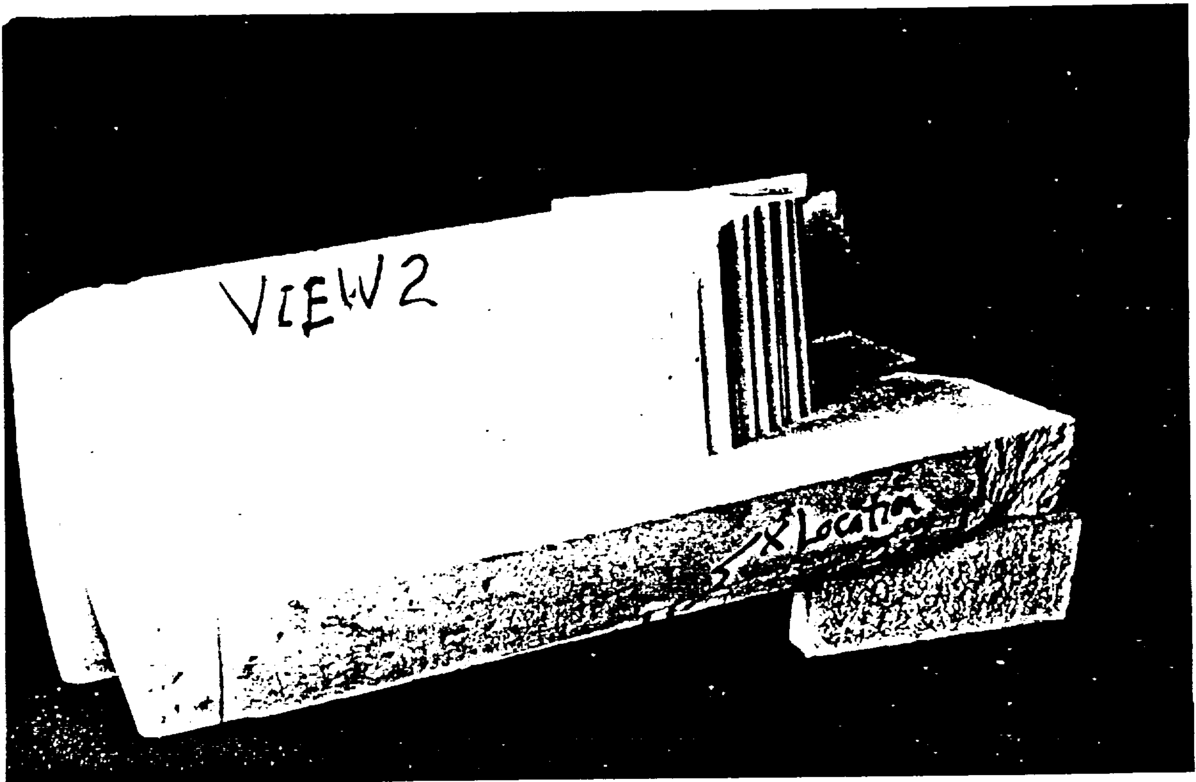



Fig.7 Machined part using conventional vise

4- Of all the proposed machining techniques, the " Constant O per pass " otherwise also called indexing machining, offers the best solution in terms of ease of data preparation, practicality, and lower wear and tear of the machine tool. Therefore it is recommended as the preferred technique for making shoe lasts. Figure 8 shows a foot using the CENCIT data and this machining technique.

Fig.8 Shoe last machined by the indexing technique and using the CENCIT data

## 2.2 CYBERWARE DATA

The scanning technology used by CYBERWARE is based on low-power laser light and precision CCD cameras physically configured for portrait sculpture just like the CENCIT system discussed previously. A 360 degree scan around the object, in this case a human head, takes typically 15 seconds and provide up to 250,000 data points ( 245 latitudes x 512 longitudes ). For a solid like a foot cast, the latitudes are separated by a constant distance of 1.531 mm while the longitude separation angular distance is 0.0122 rad. Figure 9 shows the format of the data set as provided to us by CYBERWARE.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

Before machining can be arranged, the data set must be processed to be amenable to the machine tool. It turns out that the data, which is in a cylindrical format, lends itself almost immediately to the technique of indexing machining as discussed previously. Therefore, besides the usual requirement of speed and feed, the only change to this data set is to re-arrange it so that the tool cuts material in both directions along the longitudes. This way there is no wasted time cutting "air" on the return motion. Figure 10 shows a plaster block almost fully machined. It is noted that sharp detail of the toes is maintained.

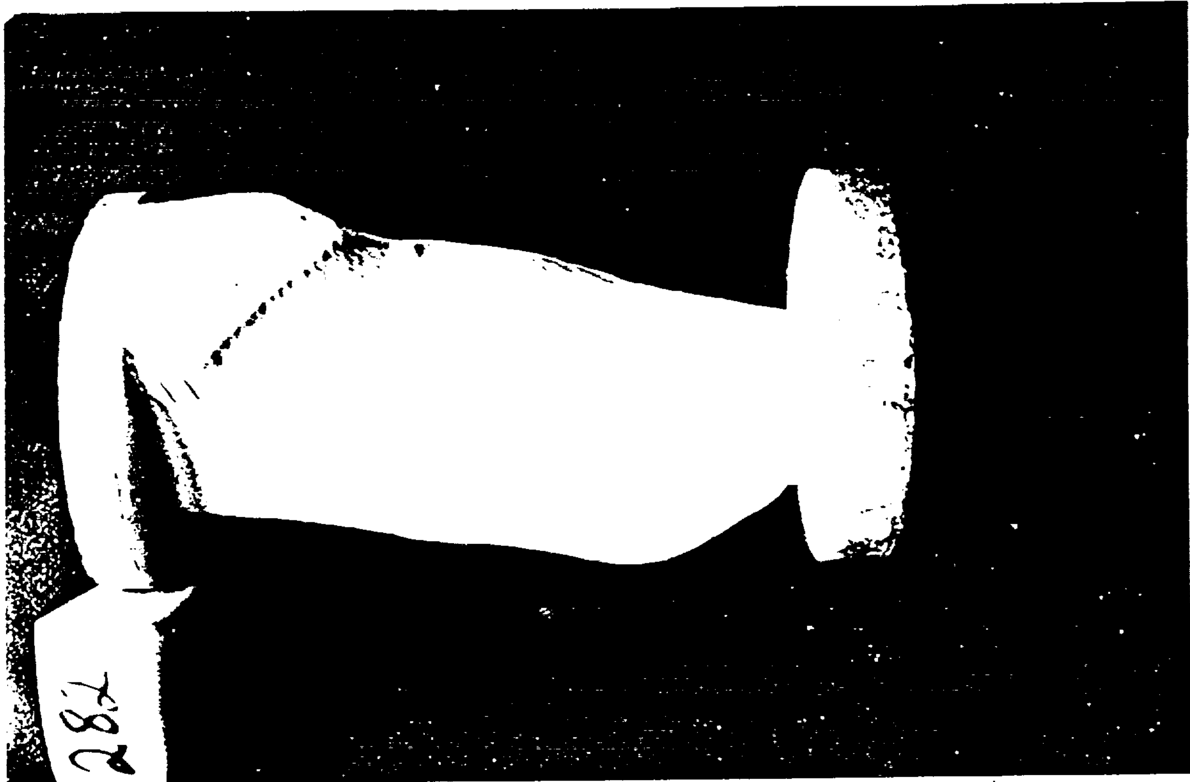
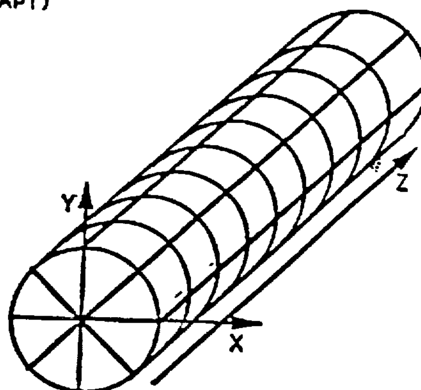
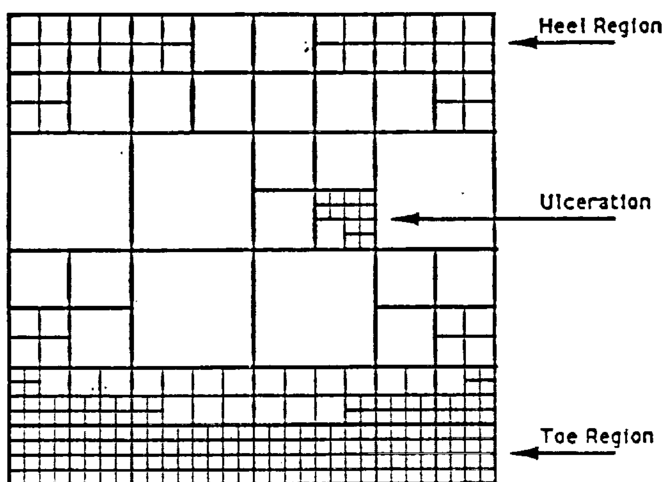


Fig.10 Foot machined by the indexing technique and using the CYBERWARE data

## 3- MACHINING OF SHOE LAST - PATCH CONFIGURATION

The work of Dr. McAllister and his group at NCSU has resulted in a shoe last being represented by a collection of Coons patches of various sizes as shown in figure 11.




Patch sizes :

largest 8 * 8

smallest 64 * 64

Fig.11 Surface patch representation

A Coons patch can be represented by position vectors, gradient vectors, and cross-boundary gradient vectors at its four corners as shown in figure 12.

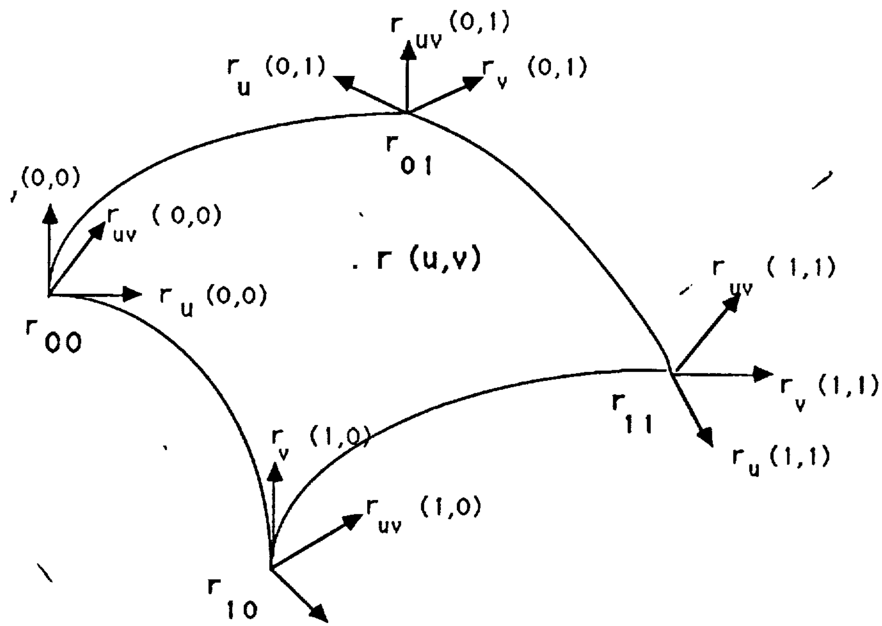



Fig. 12 Coons patch

The position vector r(u,v) of a point ( u,v) on this surface is given by

$$
r(u,v) = [1 \;\; u \;\; u^2 \;\; u^3] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} Q \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ v \\ v^2 \\ v^3 \end{bmatrix}
$$

with u, v varying between 0 and 1

$$
\text{and } Q = \begin{matrix} r(0,0) & r(0,1) & r_v(0,0) & r_v(0,1) \\ r(1,0) & r(1,1) & r_v(1,0) & r_v(1,1) \\ r_u(0,0) & r_u(0,1) & r_{uv}(0,0) & r_{uv}(0,1) \\ r_u(1,0) & r_u(1,1) & r_{uv}(1,0) & r_{uv}(1,1) \end{matrix}
$$

From Dr. McAllister's group, two output files were provided and called respectively patch file and point file. Their structures are illustrated in figures 13 and 14 respectively.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| patch 0 | 0 | 1 | 2 | 3 | 2 | 2 | 0 | C0 C1 C2 C3 Sizu Sizv Sector |
| patch 1 | 2 | 3 | 4 | 5 | 2 | 2 | 0 | C0 C1 C2 C3 are corners of patches and Sizu, Sizv are the patch sizes in u and v directions |
| | | etc | | | | | | |

Figure 13 Structure of patch file

| | | | |
|---|---|---|---|
| 1.804486e-02 | 1.00000e-02 | 0.00000e+00 | Px Py Py |
| 2.185643e-03 | 0.00000e+00 | 3.556769e-03 | Sux Suv Suz |
| 0.00000e+00 | 0.00000e+00 | 0.00000e+00 | Svx Svy Svz |
| 0.00000e+00 | 0.00000e_00 | -1.862645e-09 | Twx Twy Twz |
| | etc.. | | |

where Px Py Pz are the point coordinates, Sux Suv Suz are the tangents in the U direction, Svx Svy Svz are the tangents in the V direction, and Twx Twy Twz are the twist vectors

Figure 14 Structure of point file

## 3.1 Software development

Equipped with the patch file and point file provided by NCSU, a number of programs were developed to transform the data into useful machining codes. The sequence of "C" programs written for this purpose is shown in figure 15. The listings of these programs are given in the appendix 1. In what follows, a brief explanation of the function of each program is provided.

XYZ.C

This program reads data from the point file then discard all data related to the gradient vectors and the twist vectors.

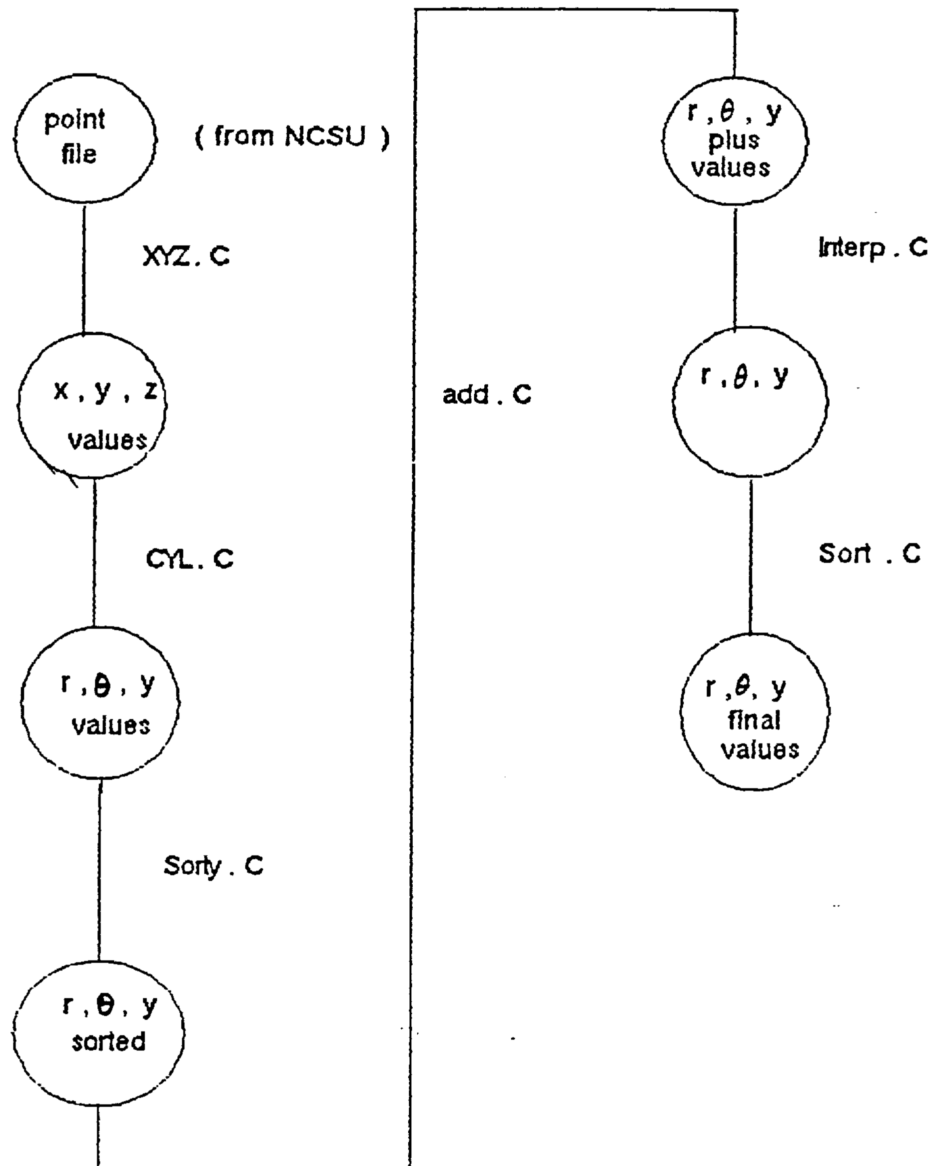
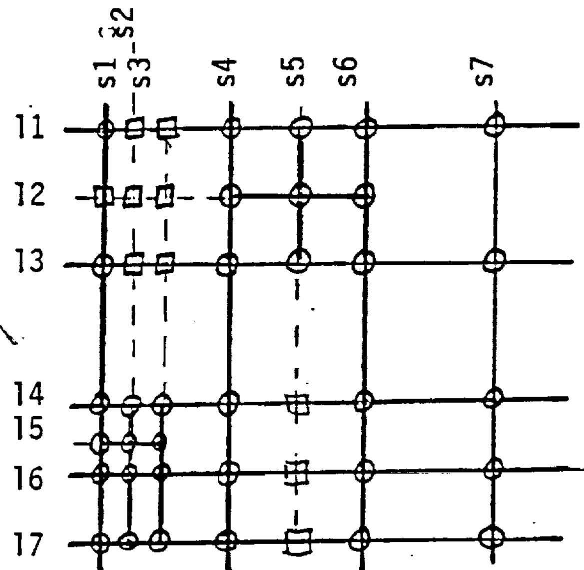



Fig.15 Programs to transform point file ( from NCSU )
to NC codes for machining tool

CYL.C
This program converts the data into cylindrical coordinates with parameters r ( radial distance ), O ( angular distance ), and y ( elevation ).

SORTY.C
The data is sorted by y ( elevation ) first, then by O ( angular distance ).

INTERP.C
In both longitudinal and lattitudinal directions the intra- distances are not the same ( due to different Coons patch sizes ),so that a number of grid points would not have data in r ( radial distance ). This program finds out where these grid points are then, using quadratic interpolation, determine the approximate radial distance for these grid points. The process is illustrated in figure 16.




o Points available from data set ( from NCSU )

□ Missing points to be added ( by Linear interpolation )

Fig. 16 Interpolation process

SORT.C

Because the tool is moved along the longitudes, the data is sorted again, this time by O first then, for each value of O, by Y. At this stage the data is ready to be sent to the machine tool after speed and feed parameters are added.

Figure 17 shows a foot represented by surface patches and machined according to the process described above. Note that it is strikingly similar to the one machined at NCSU by Dr. Sanii's research group. The length of the foot turns out to be about seven inches, which is about two inches shorter than expected. There seems to be no explanation for this except for the possibility that, somehow when they manipulated the data through LASTMOD, the researchers under Dr. McAllister might have inadvertently changed the size of the foot.

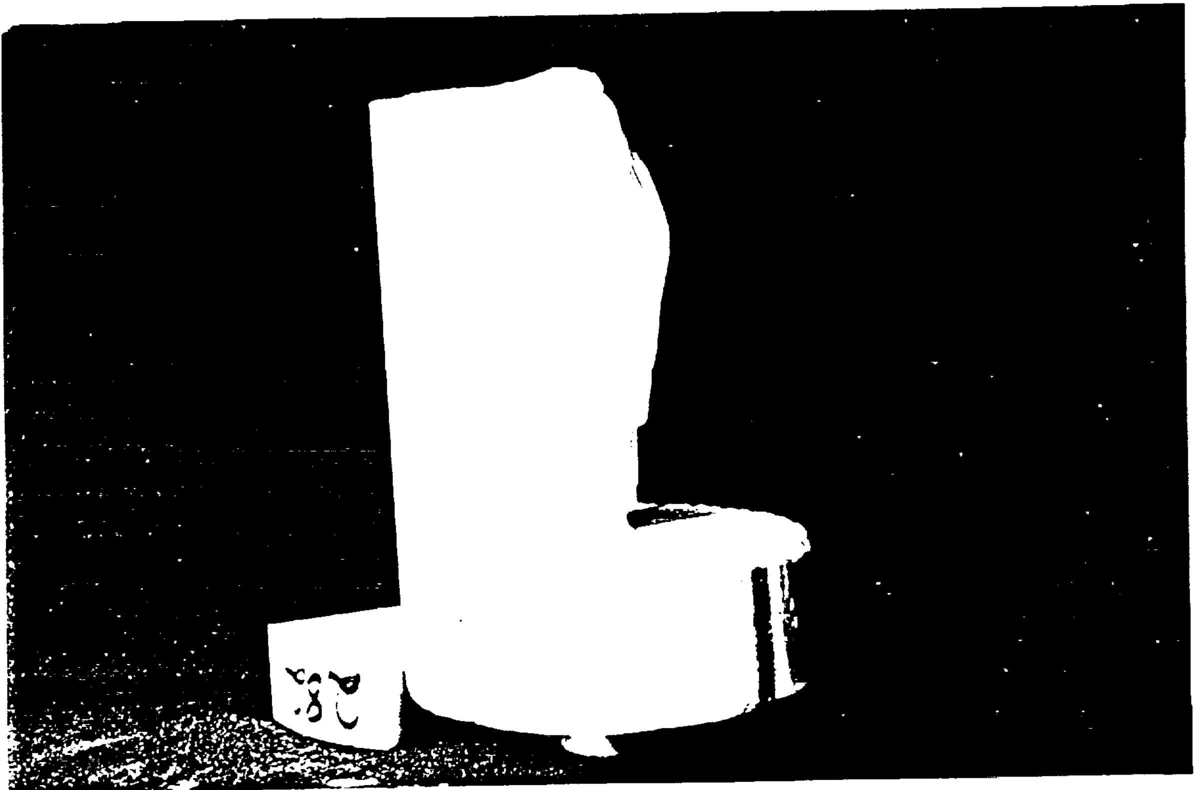


Fig.17 Foot represented by surface patches and machined accordingly

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

## 4- DESIGN AND PRODUCTION OF INTEGRATED SOLE

Minor foot deformities can usually be fixed through the use of foot aids such as pads, wedges, or orthotic devices, in conjunction with the extra-depth shoes. On the other hand, arthritic, diabetic, and neuropathic feet will require full molded insoles to protect them from repetitive stress which may cause neurotropic ulcers and intractable infection. Molded insoles have the functional characteristic of " bringing the ground up to the foot ".

The current practice of molded insole is to use a combination of PPT and Plastazote topped by a nylon material for reducing friction. PPT has a high energy absorption charateristic and does not bottom out significantly, even after many years of use. Plastazote on the other hand is easily molded to the foot or last by heat and offers good resistance against shear.

While the use of PPT/Plastazote material represents a major advance in molded insole, its use does not exclude the usual manual operation of grinding the lower layer to shape. Usually a mid-sole must be built up underneath the insole, and it too requires shaping through manual shaving and grinding. The manual labor just mentioned could be reduced if a technique for fabricating an integrated sole, that is a combination of insole and midsole, can be found. The work in this phase provides a viable means of producing such an item, as explained below.

### 4.1 Foot position requirement

It is as important for the foot to be in the correct stance before being digitized by a scanner as during the more conventional casting operation. Likewise this stance must be maintained before the design of the integrated sole can take place.

In general, the three design criteria are the heel pitch, the balance line, and the rigidity condition of the foot. The heel pitch is specified according to to aesthetic requirement ( male or female ) and anatomical features such as leg-length discrepancy. There exist general heel pitch values, and they should be used as much as possible. The balance line is a line drawn on the back of the last for purpose of balancing the cast after the addition of the heel. This balance line is usually vertical along the length of the tibias but, when conditions of foot inflexibility and severe pronation or supination exist, it should be selected appropriately and used to orient the last before one can proceed with the design of the sole.

Generally speaking, if the foot is flexible then a certain amount of cast inversion or eversion is tolerated. This means that the angle of inversion or eversion must be specified by the user. If the foot is inflexible or deformation conditions such as cavus foot, equino-varus, or considerable tibia varum exist then the balance line is drawn while the foot is in its natural orientation.

## 4.2 Sole Production Technique

The critical data discussed in the previous section will serve as design parameters in the orientation and location of the foot relative to ground prior to the determination of the shape of the sole. Figure 18 shows a sketch of a foot in the "ideal" position.

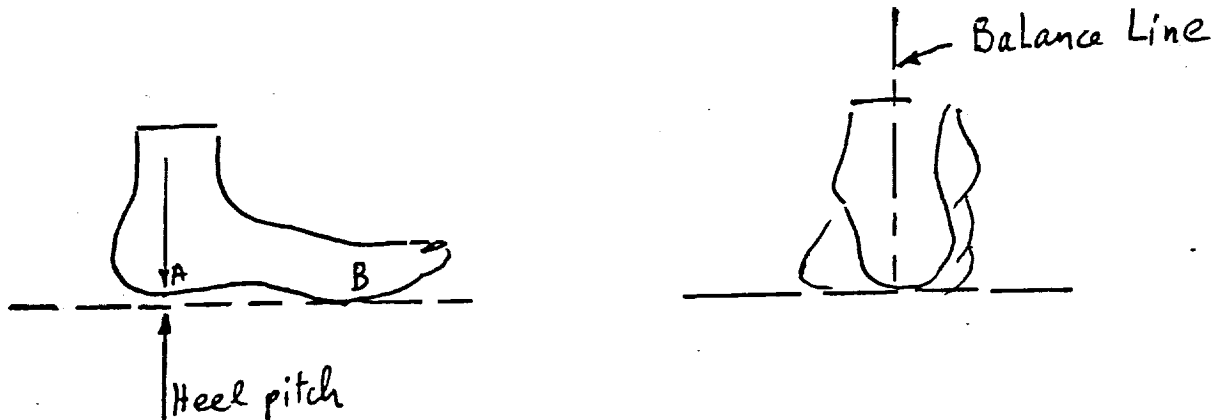



Fig.18 Foot position prior to sole design

To make the mold for casting the sole, imagine that the foot can be placed in an upside down position so that the plantar surface now becomes the bottom surface of the mold cavity as shown in figure 19.

Knowing the spatial location of the plantar surface, a milling machine can be programmed to machine a cavity in a wooden block with its bottom surface being the foot plantar surface in its upside down position. Liquid latex can then be poured into the cavity to form the sole. Figure 20 shows a wooden mold ready to accept the liquid latex. Figure 21 shows the sole after solidification.

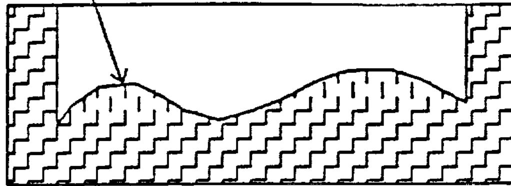
Our experience has shown that the liquid latex that we use, a product called TC 281 from BJB enterprise (#1), is an odorless and fast setting foam with a tendency to expand to 2 or 3 times its original volume during reaction time. Because of this expansion, we found it necessary to contain the liquid in a close mold to force it to have a higher density, at least on its skin. This product has a high energy absorption characteristic and is easily casted in the appropriate mold. Its durability is however not known at this stage although some shoe orthotists have highly recommended it for combination sole. In the next phase, further experiments are needed to improve its production process and discover its other characteristics such as water resistance, durability, tendency to hold up under constant compression, etc...

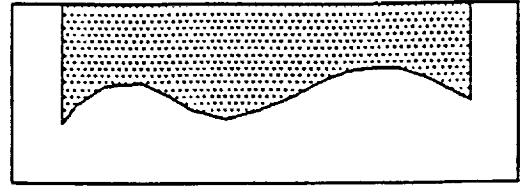
---

(#1) BJB Enterprise, 13912 Nautilus Dr., Garden Grove, CA 92643

## SOLE MOLD

Plantar surface extracted from computer model of last





a) machining of mold

b) casting of sole using liquid latex

## SOFTWARE FOR GENERATING NC CODES FOR MOLD MACHINING

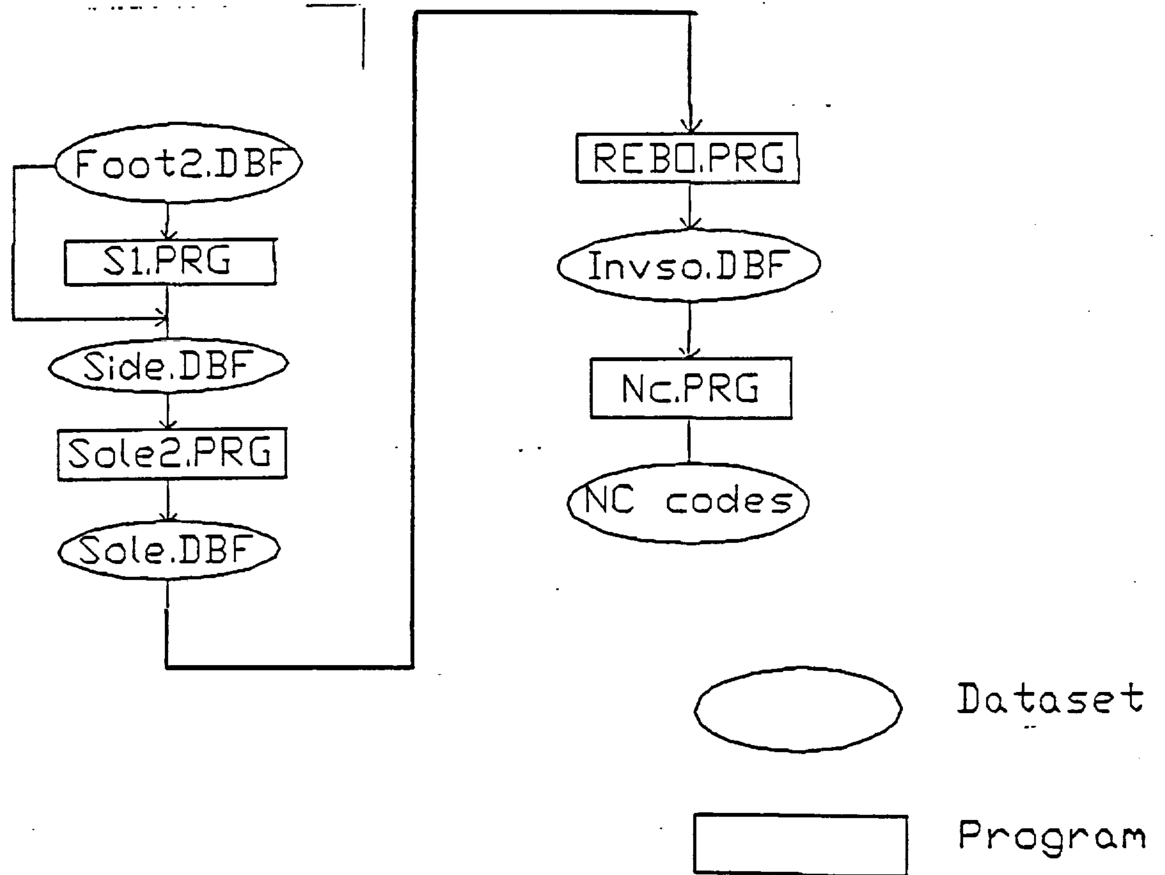



Fig.19 Mold for casting sole

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

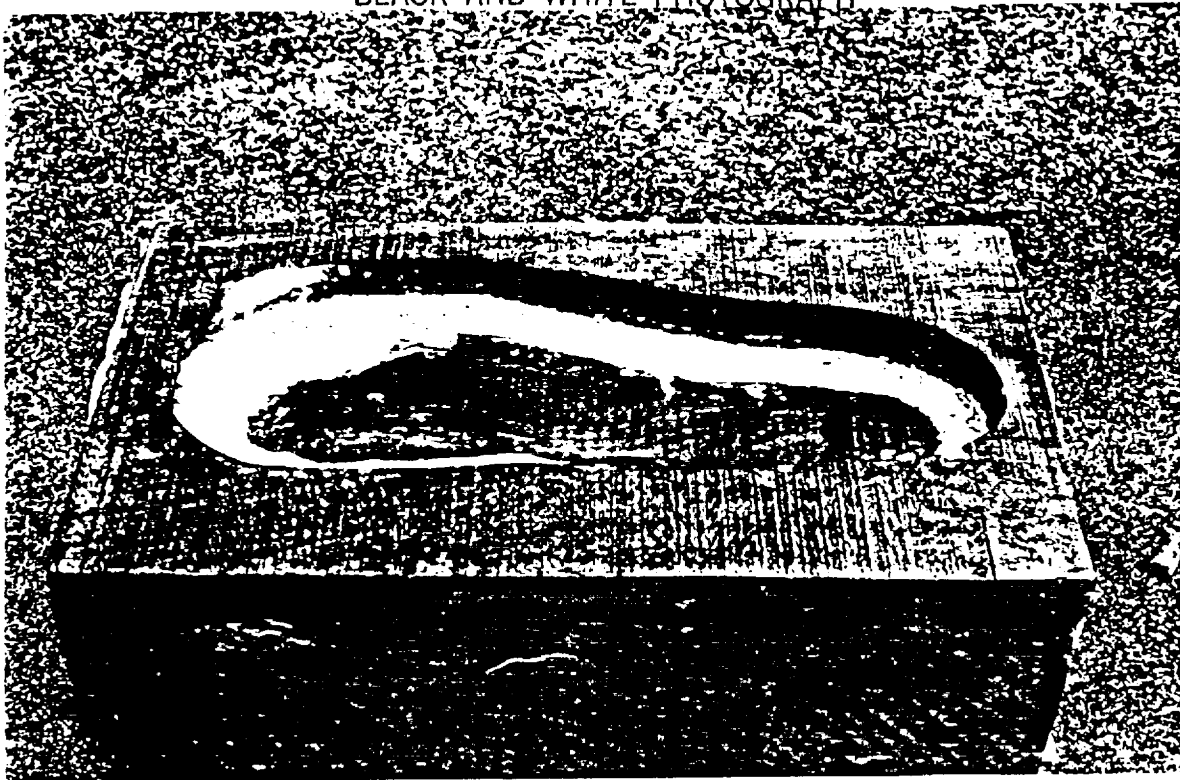


Fig.20 Wooden mold used to cast sole

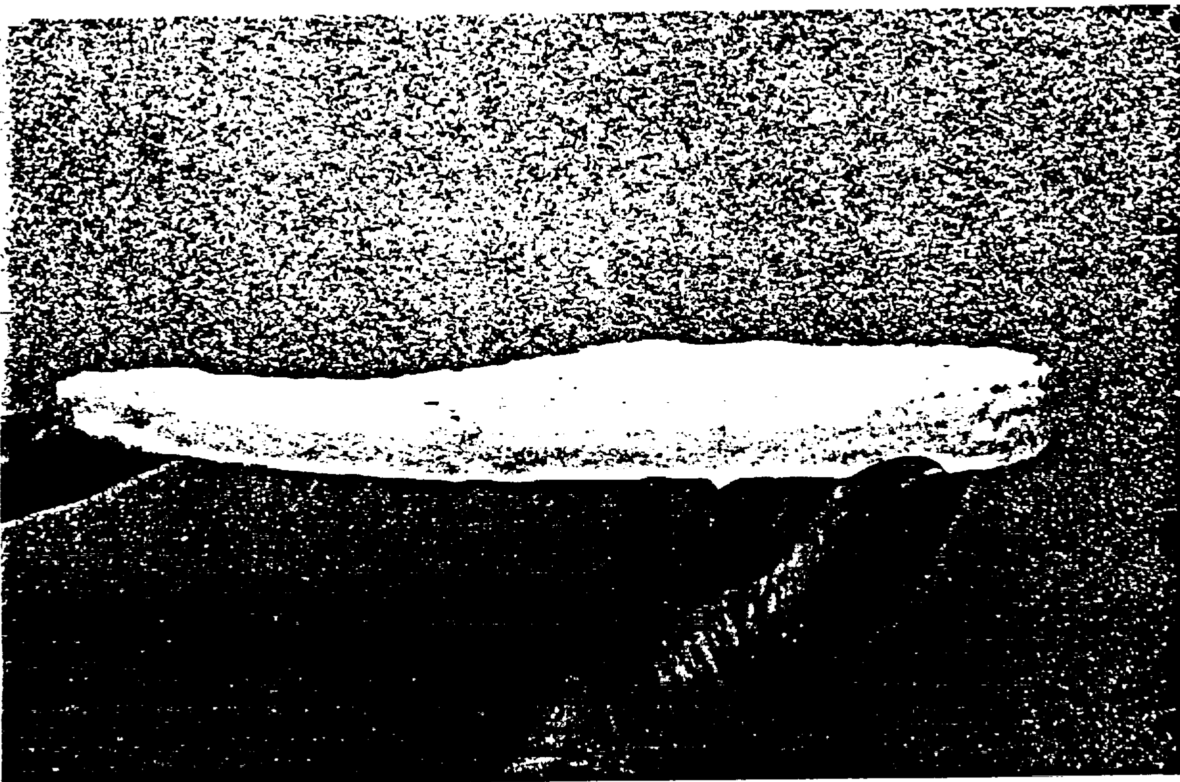


Fig.21 Sole casted in TC 281 product

### 4.3 Software development

One final discussion is about the software developed for preparing the data for the machine tool. A sequence of programs have been written for this purpose as shown in figure 19. To help explain the function of each program in this sequence, figure 22 is provided to indicate the line of maximum foot width.

Fig.22 Line of Maximum width

FOOT2.DBF is the foot data set from either CENCIT or CYBERWARE. Note that it must be in cylindrical coordinates and it must be converted to a Dbase file format since the Dbase III Plus data base management system was used for this purpose.

S1.PRG extracts the smallest and largest y values for each x value then puts the result in file SIDE.DBF .

SOLE2.PRG uses both data sets FOOT23.DBF and SIDE.DBF to determine the points between the Y min and Y max values and below the line passing through these extreme points.

REBO.PRG rearranges the sequence of the points in SOLE.DBF for continuous two-way motion of the tool.

NC.PRG finally generates the NC codes required by the milling machine.

The listings of all of the above programs are available in appendix 2 at the end of the report.

Figures 23 and 24 show two examples of integrated soles, respectively in silicone and in liquid latex ( TC 282 product ).

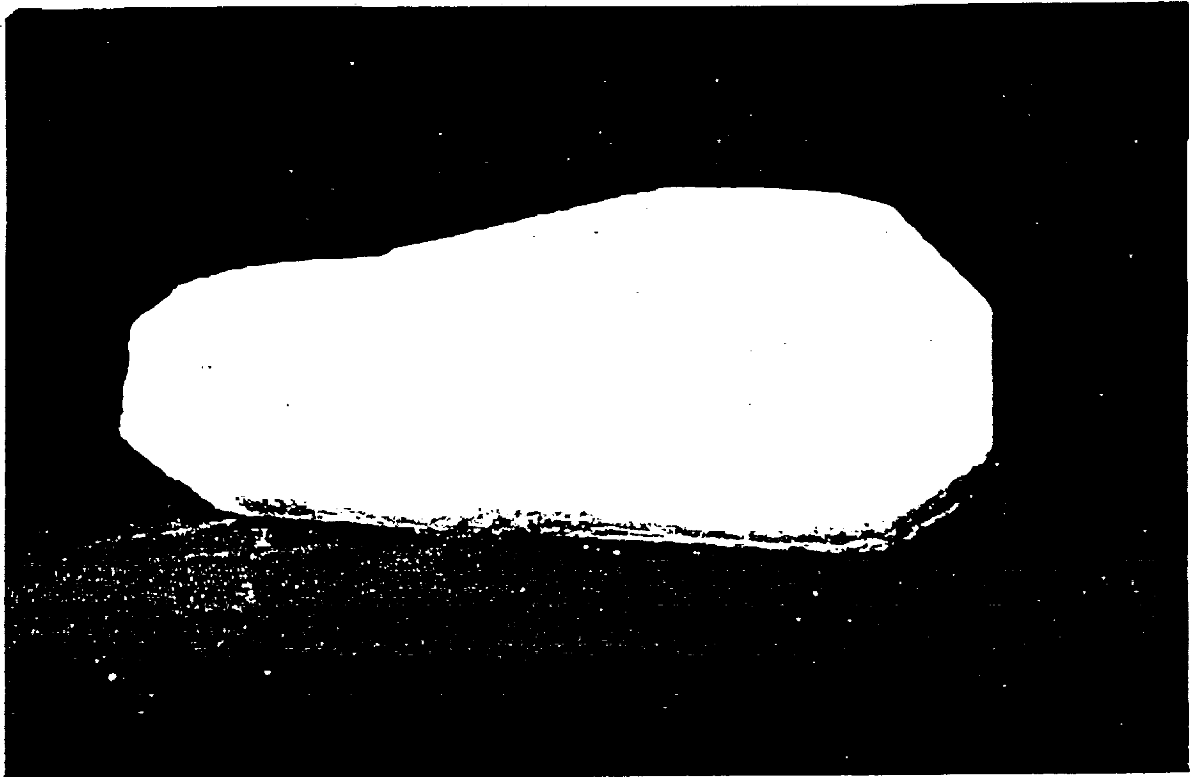


Fig.23 Sole made of silicone

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

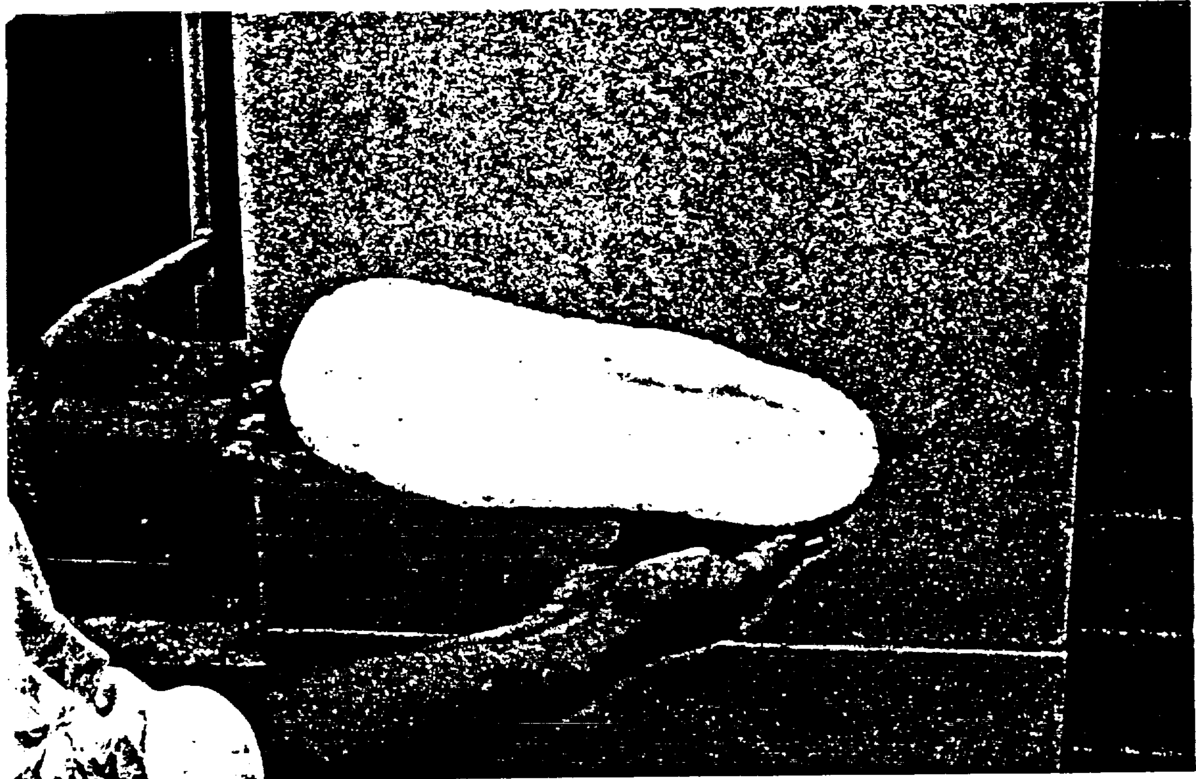


Fig.24 Sole made of TC 281 liquid foam

# APPENDIX 1

## LISTING OF COMPUTER PROGRAMS FOR MACHINING SURFACE PATCHES

XYZ.C

```
/* reads data from COONPOIN.TXT and write those representing
   the x y z points                                              */

#include <stdio.h>
float  x[5000], y[5000], z[5000];
main()
{
   FILE  *fin1;
   FILE  *fout1;
   int   count=0, countr=0;

   fin1  = fopen("coonpoin.txt", "r");
   fout1 = fopen("xyz.pnt", "w");

   while( fscanf(fin1, "%E %E %E", &x[count], &y[count], &z[count]) != EOF)
        {
/*      printf( "%d %f %f %f\n", count,x[count],y[count],z[count]);*/
        count++;
        };
   for(countr=0; countr<count; countr+=4)
      {
      fprintf(fout1,"%e %e %e\n", x[countr], y[countr], z[countr]);
      printf("%f %f %f\n",x[countr], y[countr], z[countr]);
       /*printf( "%10d.\n", countr); */
      }


   fclose(fin1);
   fclose(fout1);
}
```

```
/***************************************************
/*  Convert points into CYLINDRICAL COORDINATE
/*  with 3 decimal places of the T.
/***************************************************/

#include <stdio.h>
#include <math.h>
#define pi 3.14159265359

struct point
    {
    float x;
    float y;
    float z;
    };

struct point xyz[907];
int    numpoint;

main()
{
   FILE  *fin1;
   FILE  *fout1;
   int   count1=0, count2=0, countr=0;

   fin1  = fopen("xyz.pnt", "r");
   fout1 = fopen("rTHy.pnt", "w");

   while( fscanf(fin1, "%E %E %E",
                &xyz[count1].x, &xyz[count1].y, &xyz[count1].z) != EOF)
   {
       count1++;
   }

   numpoint=count1--;

   cylin();

   for (countr = 0; countr < numpoint; countr++)
       fprintf(fout1,"%f %.3f %f\n",
                    xyz[countr].x, xyz[countr].z, xyz[countr].y);


   fclose(fin1);
   fclose(fout1);
}

cylin()
{
    int count;
    float r,th, yy,xx;
    for (count = 0; count < numpoint; count++)
    {
```

```
            xx = xyz[count].x; yy = xyz[count].z;
            r = xx * xx + yy * yy;
            r = (float)sqrt( (double)r );
            if ((xx==0)&&(yy==0)) printf("count=%d\n xx=%f yy=%f\n" , count,xx,yy);
            th = (float)atan2( (float)yy, (float)xx );
            xyz[count].x = r;
            xyz[count].z = th*180.0/pi;
        }

    }
```

```
/***************************************************
/* This is the final sorting program
/* Sorting Y and T field
/***************************************************/

#include <stdio.h>
#include <string.h>
#include <search.h>
#include <math.h>

struct point
     {
     float r;
     float t;
     float y;
     };

struct point rty[906];
int     numpoint;
float    *pp[907];

float  x[10];
float *mr, *my;\
int     numpoint;
int     mycompare();

main()
{
   FILE  *fin1;
   FILE  *fout1, *fout2;
   int   count1=0, count2=0, countr=0;

   fin1  = fopen("rthy.pnt", "r");
   fout1 = fopen("rthy.srt", "w");
   fout2 = fopen("&.pnt","w");
   while( fscanf(fin1, "%f %f %f", &rty[count1].r,
                                    &rty[count1].t,
                                    &rty[count1].y
                                                 ) != EOF)  count1++;
    printf("count1=%d\n", count1);
    for (count2=0; count2<count1; count2++)
         {
         pp[count2]= &rty[count2].y;
         fprintf(fout2, "%d %d %d %d\n %d\n %d\n",
           &rty[count2].r, &rty[count2].t, &rty[count2].y, count2, pp[count2],
           &pp[count2]);
         }

   numpoint=count1;
   for (count2=0; count2 < count1; count2++)
    fprintf(fout2,"%d, %f\n", pp[count2], *pp[count2]);

   fclose(fout2);
```

```
     qsort( pp, numpoint, 2, mycompare);

    for (countr = 0; countr < count1; countr++)
     {         mr =  pp[countr]-2;     /* pointers to r and t*/
               my =  pp[countr]-1;

/*    fprintf(fout1,"%f %.2f %f %d %d %d %d\n",
/*             *mr, *my, *pp[countr], mr, my, pp[countr], countr);
/*    }       /* r t y*/
      fprintf(fout1,"%f %.3f %f\n",
               *mr, *my, *pp[countr]);
      }       /* r t y*/

  fclose(fin1);      fclose(fout1);
}

int mycompare(int *arg1, int *arg2 )
{
       float *elm1, *elm2;
       float result;

      elm1 = (float *)*arg1; elm2 = (float *)*arg2;

/*    printf("%d %d\n",arg1, arg2);
   printf("%d %d\n",*arg1,*arg2);
   printf("%d %d %f %f\n",elm1, elm2, *elm1, *elm2);
*/
     result = (*elm1 - *elm2)*10.0;
     if (result == 0.0 )
     {
      result = (*(elm1-1) - *(elm2-1))*10.0; /*compare t*/
      /*printf ("%d %d %f\n",elm1+1, elm2+1, result);*/
     }
     return( (int) result);
}
```

```
/***************************************************
/* The smallest increment in T is 5.625 degrees.
/* This program interpolate for each y with
/* 5.625 degree increment.
/***************************************************/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

struct point
    {
    float r;
    float t;
    float y;
    };

struct point rty[906];
struct point frty[3400];
int    numpoint;
int  cy=0; /* number of contours along the y */

/* 64 segments a contour */
main()
{
   FILE  *fin1;
   FILE  *fout1, *fout2;
   void fillin();
   void firstcontour();
   void allintpol();
   int   count1=0, count2=0;
   float y=0;
/* check number of y
/* there are 52 contour along the y (actual file)*/

   fin1  = fopen("rthy.srt", "r");/* try with smaller number of points */
   while( fscanf(fin1, "%f %f %f", &rty[count1].r,
                                    &rty[count1].t,
                                    &rty[count1].y
                                                 ) != EOF)
   {
    if (y != rty[count1].y) cy++;
    y = rty[count1].y;
    count1++;
   }
   numpoint = 64*cy;
   fclose (fin1);
   printf("number of contour on y = %d", cy);

   fillin();
    fout2 = fopen("fill.sr2", "w");
    for(count1=0; count1< 64*cy; count1++)
    fprintf( fout2, "%10.6f%10.3f%10.6f\n",
             frty[count1].r, frty[count1].t, frty[count1].y );
```

```
     fclose(fout2);

    firstcontour();
    allintpol();

     fout1 = fopen("fill.sr1", "w");
     for(count1=0; count1< 64*cy; count1++)
     fprintf( fout1, "%10.6f%10.3f%10.6f\n",
              frty[count1].r, frty[count1].t, frty[count1].y );
     fclose(fout1);

}
/***********************************************************************
 * f i l l i n()
 *      fills in unassigned points with 999.0
 ***********************************************************************/
void fillin()
{
     int npnt=0, npntf=0;
     float angle=-180.0, incra= 5.625, yy, tt;

     yy = rty[0].y;
     while (npntf < 64*cy)
          {
               if (angle < 180)
                    {    tt = rty[npnt].t;
                         if (tt != angle)
                              {
                                   frty[npntf].r = 999;
                                   frty[npntf].t = angle;
                                   frty[npntf].y = yy;
                              }
                         else
                              { if (yy == rty[npnt].y)
                                     {
                                        frty[npntf].r = rty[npnt].r;
                                        frty[npntf].t = rty[npnt].t;
                                        frty[npntf].y = rty[npnt].y;
                                        npnt++;
                                     }
                                else
                                     {
                                        frty[npntf].r = 999;
                                        frty[npntf].t = angle;
                                        frty[npntf].y = yy;
                                     }
                              }

                         npntf++;
                         angle += incra;
                    }
               else
                    {
```

```
                    angle = -180;
                    yy = rty[npnt].y;
                }
        }
}
/**********************************************************************
 * f i r s t c o n t o u r()
 *      interpolates any 999.0 points from its neighbors.
 **********************************************************************/
void firstcontour()
{
    int npnt = 0, back, back1, forw;
    int search();
    float intpol();
    FILE *fout1;
    fout1 = fopen("first.con","w");
    while (npnt<64)
        {
            if (frty[npnt].r == 999.0)
                {

                    back = search(0,npnt);
                    forw = search(1,npnt);
                    frty[npnt].r = intpol(frty[npnt].t,
                                          frty[forw].t,
                                          frty[back].t,
                                          frty[forw].r,
                                          frty[back].r);
                }
            fprintf(fout1,"b=%d r=%f f=%d r=%f npnt=%d r=%f\n",
                    back, frty[back].r, forw, frty[forw].r, npnt, frty[npnt].r);
            npnt++;
        }
    fclose(fout1);
}
/**********************************************************************
 * a l l i n t p o l()
 *      replace any 999.0 points by either
 *              interpolating from its neighbors of same contour
 *          or
 *              their counterpart in the previous contour.
 **********************************************************************/
void allintpol()
{
    int npnt=64, back, forw;
    int search();
    float intpol();
    FILE *fout3;
    fout3 = fopen("all.con","w");     printf("numpoint=%d\n",numpoint);
    while (npnt<numpoint)
        {
            if (frty[npnt].r == 999.0)
                {
```

```
                    back = search(0,npnt);
                    forw = search(1,npnt);
                    if ( min(forw-back,(npnt+64-back)) <= 3 )
                        {
                            frty[npnt].r = intpol(frty[npnt].t,
                                                  frty[forw].t,
                                                  frty[back].t,
                                                  frty[forw].r,
                                                  frty[back].r);
                            fprintf(fout3," less 3\n");
                        }
                    else
                        {
                            frty[npnt].r = frty[npnt-64].r;
                            fprintf(fout3," more 3\n");
                        }
                }
            npnt++;
        }
    fclose(fout3);
}
/*********************************************************************
 * s e a r c h(df, npnt0)
 *      searches next non 999.0 points
 *               either forward or backward direction
 *********************************************************************/
int search(bf,npnt0)
int     bf, npnt0;
{
    int npnt1;

    if (bf==0)
        {
            if ( fmod(npnt0,64)==0)
                {
                    npnt1 = npnt0 + 63;
                    while(frty[npnt1].r == 999.0)
                        {
                            /*printf("npnt1=%d frty.r=%f\n",npnt1,frty[npnt1].r);*/
                              npnt1--;
                        }
                  /*printf("return npnt1=%d frty[npnt1].r=%f\n",npnt1, frty[npnt1].r

                    return(npnt1);
                }
            else
                {
                    npnt1 = npnt0 - 1;
                    while(frty[npnt1].r == 999.0)
                        {
                            /*printf("npnt1=%d frty.r=%f\n",npnt1,frty[npnt1].r);*/
                              npnt1--;
                        }
```

```
                        /*printf("return npnt1=%d frty[npnt1].r=%f\n",npnt1, frty[npnt1].r

                           return(npnt1);
                        }
                }
        else
                {
                        npnt1 = npnt0 + 1;
                        if ( fmod(npnt1,64)==0 )
                            {search(1, npnt0-63);}
                        else
                            {while(frty[npnt1].r == 999.0)
                                    {
                                        /*printf("npnt1=%d frty.r=%f\n",npnt1,frty[npnt1].r);*/
                                          npnt1++;
                                    }
                              /*printf("return npnt1=%d frty[npnt1].r=%f\n",npnt1, frty[npnt1].r


                            return(npnt1);
                          }
                }
}
/*********************************************************************
 * i n t p o l(t, r1, r2)
 *      intpolates a polar point from 2 polar-to-rect points
 *********************************************************************/
float intpol(t, t1, t2, r1, r2)
float   t, t1, t2, r1, r2;
{
    float tt;
    tt=t2-t1;
    if (tt!=0.0)
    return( (r2-r1)/(tt)*(t-t1) + r1 );
    else return(9999);

}
```

```
/*************************************************
/* This is the final sorting program
/* Sorting T and Y field
/*************************************************/

#include <stdio.h>
#include <string.h>
#include <search.h>
#include <math.h>

struct point
    {
    float r;
    float t;
    float y;
    };

struct point rty[2820];
int    numpoint;
float   *pp[2820];

float *mr, *my;
int    numpoint;
int    mycompare();

main()
{
   FILE  *fin1;
   FILE  *fout1, *fout2;
   int   count1=0, count2=0, countr=0;

   fin1  = fopen("fill.sr1", "r");
   fout1 = fopen("fill.ty ", "w");
   fout2 = fopen("&.pnt","w");
   while( fscanf(fin1, "%f %f %f", &rty[count1].r,
                                   &rty[count1].t,
                                   &rty[count1].y
                                               ) != EOF)  count1++;
    printf("count1=%d\n", count1);
    for (count2=0; count2<count1; count2++)
        {
        pp[count2]= &rty[count2].t;
        fprintf(fout2, "%d %d %d\n %d\n %d\n",
          &rty[count2].r, &rty[count2].t, &rty[count2].y, count2, pp[count2],
          &pp[count2]);
        }
   printf("read ok\n");
   numpoint=count1;
   for (count2=0; count2 < count1; count2++)
    fprintf(fout2,"%d, %f\n", pp[count2], *pp[count2]);

   fclose(fout2);
```

```
    qsort( pp, numpoint, 2, mycompare);

   for (countr = 0; countr < count1; countr++)
    {         mr =  pp[countr]-1;     /* pointers to r and t*/
              my =  pp[countr]-2;

/*    fprintf(fout1,"%f %.2f %f %d %d %d %d\n",
/*            *mr, *my, *pp[countr], mr, my, pp[countr], countr);
/*    }       /* r t y*/
      fprintf(fout1,"%f %.3f %f\n",
              *mr, *pp[countr], *my );
      }       /* r t y*/

  fclose(fin1);      fclose(fout1);
}

int mycompare(int *arg1, int *arg2 )
{
      float *elm1, *elm2;
      double result;

     elm1 = (float *)*arg1; elm2 = (float *)*arg2;

/*    printf("%d \%d\n",arg1, arg2);
   printf("%d %d\n",*arg1,*arg2);
   printf("%d %d %f %f\n",elm1, elm2, *elm1, *elm2);
*/
    result = (*elm1 - *elm2)*10.0;
    if (result == 0.0 )
    {
       result = (*(elm1+1) - *(elm2+1))*10.0; /*compare y*/

   /*printf ("%d %d %f\n",elm1+1, elm2+1, result);*/
    }
    return( (int) result);
}
```

# APPENDIX 2

## LISTING OF COMPUTER PROGRAMS FOR MACHINING MOLD FOR CASTING SOLE

```
******************  PROGRAM NAME : NC.PRG  ************************
close databases
RE='&'
STA='(NCEIA1  )'+RE
select 2
use nceia
APPEND BLANK
REPLACE STAMENT WITH STA
select 1
use invso3
N='N'
LI=100
G1='G00'
G2='G01'
F='F60'
XX='X'
YY='Y'
ZZ='Z'
PE='(E)'
M1='M00'
M2='M03'
S1=LTRIM(STR(LI,4))
S2=LTRIM(STR(INVSO3->IX,8,4))
S3=LTRIM(STR(INVSO3->IY,8,4))
S4=LTRIM(STR(INVSO3->IZ,8,4))
UP='5'
IF LEN(S1)<4
   S1='0'+S1
ENDIF
STA=N+S1+PE+'G90'+RE
SELECT 2
APPEND BLANK
REPLACE STAMENT WITH STA
LI=LI+1
STA=N+'0'+LTRIM(STR(LI,4))+'(9)'+'M06'+'T01'+RE
SELECT 2
APPEND BLANK
REPLACE STAMENT WITH STA
LI=LI+1
STA=N+'0'+LTRIM(STR(LI,4))+'(9)'+'M03'+'S757'+RE
SELECT 2
APPEND BLANK
REPLACE STAMENT WITH STA
LI=LI+1
STA=N+'0'+LTRIM(STR(LI,4))+PE+G1+XX+S2+YY+S3+ZZ+S4+RE
SELECT 2
APPEND BLANK
REPLACE STAMENT WITH STA
SELECT 1
SKIP
LI=LI+1
S1=LTRIM(STR(LI,4))
S2=LTRIM(STR(INVSO3->IX,8,4))
S3=LTRIM(STR(INVSO3->IY,8,4))
```

```
S4=LTRIM(STR(INVSO3->IZ,8,4))
IF LEN(S1)<4
   S1='0'+S1
ENDIF
STA=N+S1+PE+G2+XX+S2+YY+S3+ZZ+S4+F+RE
SELECT 2
APPEND BLANK
REPLACE STAMENT WITH STA
SELECT 1
SKIP
DO WHILE .T.
   IF .NOT.EOF()
      LI=LI+1
      S1=LTRIM(STR(LI,4))
      S2=LTRIM(STR(INVSO3->IX,8,4))
      S3=LTRIM(STR(INVSO3->IY,8,4))
      S4=LTRIM(STR(INVSO3->IZ,8,4))
      IF LEN(S1)<4
         S1='0'+S1
      ENDIF
      STA=N+S1+PE+G2+XX+S2+YY+S3+ZZ+S4+F+RE
      SELECT 2
      APPEND BLANK
      REPLACE STAMENT WITH STA
      SELECT 1
      SKIP
      LOOP
   ELSE
      LI=LI+1
      STA=N+LTRIM(STR(LI,4))+PE+'G01'+ZZ+UP+RE
      SELECT 2
      APPEND BLANK
      REPLACE STAMENT WITH STA
      LI=LI+1
      STA=N+LTRIM(STR(LI,4))+PE+'G00'+XX+'0.0'+YY+'0.0'+ZZ+'0.0'+RE
      APPEND BLANK
      REPLACE STAMENT WITH STA
      LI=LI+1
      STA=N+LTRIM(STR(LI,4))+PE+'G17'+RE
      APPEND BLANK
      REPLACE STAMENT WITH STA
      LI=LI+1
      STA=N+LTRIM(STR(LI,4))+'(9)'+'M05'+RE
      APPEND BLANK
      REPLACE STAMENT WITH STA
      LI=LI+1
      STA=N+LTRIM(STR(LI,4))+'(9)'+'M02'+RE
      APPEND BLANK
      REPLACE STAMENT WITH STA
      LI=LI+1
      STA=N+LTRIM(STR(LI,4))+'(9)'+'M30'+RE
      APPEND BLANK
      REPLACE STAMENT WITH STA
      STA='END'+RE
```

```
        APPEND BLANK
        REPLACE STAMENT WITH STA
        EXIT
    ENDIF
ENDDO
CLOSE DATABASES
```

```
********************  PROGRAM NAME : S1.PRG  *************************
close databases
select 2
use side
select 1
use foot2
rx=x
maax=x
maay=y
maaz=z
miix=x
miiy=y
miiz=z
skip
do while .t.
select 1
if .not.(eof())
   if x<>rx+0.5
      if y>maay
         maay=y
         maax=x
         maaz=z
         skip
         loop
      else
         if y<miiy
            miix=x
            miiy=y
            miiz=z
            skip
            loop
         else
            skip
            loop
         endif
      endif
   else
      select 2
      append blank
      replace lax with maax,lay with maay,laz with maaz
      replace smx with miix,smy with miiy,smz with miiz
      select 1
      rx=x
      maax=x
      maay=y
      maaz=z
      miix=x
      miiy=y
      miiz=z
      loop
   endif
else
   select 2
   append blank
```

```
    replace lax with maax,lay with maay,laz with maaz
    replace smx with miix,smy with miiy,smz with miiz
    exit
 endif
 enddo
```

```
*******************  PROGRAM NAME : SOLE2.PRG  *************************
close databases
select 1
use foot2
rx=x
select 2
use side
select 3
use sole
flag1=0
flag2=0
flag3=0
do while .t.
select 1
if x<>rx+0.5.and.(.not.eof())
   if y=side->lay
      flag1=1
   endif
   if y=side->smy
      flag2=1
      flag3=1
   endif
   do case
          case flag1=0.and.flag2=0.and.flag3=0
               select 1
               skip
               loop
          case flag1=1.and.flag2=0.and.flag3=0
               select 3
               append blank
               replace sox with foot2->x,soy with foot2->y,soz with foot
               select 1
               skip
               loop
          case flag1=1.and.flag2=1.and.flag3=1
               select 3
               append blank
               replace sox with foot2->x,soy with foot2->y,soz with foot
               flag3=0
               select 1
               skip
               loop
          case flag1=1.and.flag2=1.and.flag3=0
               select 1
               skip
               loop


   endcase
else
   if .not.(eof())
      flag1=0
      flag2=0
```

```
          rx=x
          select 2
          skip
          loop
       else
          exit
       endif
    endif
    enddo
```

```
******************  PROGRAM NAME : REBO.PRG  ************************
close databases
select 2
use invso2
select 1
use sole2
rx=sox
sh=0
cc=0
do while .t.
   if sox=rx.and.(.not.eof())
      select 2
      append blank
      replace ix with sole2->sox,iy with sole2->soy,iz with sole2->soz
      select 1
      skip
      loop
   else
      if sox=rx+0.1.and.(.not.eof())
         cc=cc+1
         skip
         loop
      else
         if sox=rx+0.2.and.(.not.eof())
            rx=sox
            bb=cc
            skip -1
            do while .t.
               if cc<>0
                  select 2
                  append blank
             replace ix with sole2->sox,iy with sole2->soy,iz with sole2
                  select 1
                  cc=cc-1
                  skip -1
                  loop
               else
                  select 1
                  dd=recno()+bb+1
                  goto dd
                  exit
               endif
            enddo
            loop
         else
            if eof()
               exit
            endif
         endif
      endif
   endif
enddo
```

```
******************* PROGRAM NAME : K0.PRG ************************
select 1
use invso3
select 2
use invso2
pp=0
do while .t.
if iz>0.and.(.not.eof())
   select 1
   append blank
   replace ix with invso2->ix,iy with invso2->iy,iz with pp
   select 2
   skip
   loop
else
   if (.not.eof())
     select 1
     append blank
     replace ix with invso2->ix,iy with invso2->iy,iz with invso2->iz
     select 2
     skip
     loop
   else
     exit
endif
enddo
return
```