PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 19/00 | A1

(11) International Publication Number: WO 97/37317

(43) International Publication Date: 9 October 1997 (09.10.97)

(21) International Application Number: PCT/US97/05419

(22) International Filing Date: 1 April 1997 (01.04.97)

(30) Priority Data:
625,142 1 April 1996 (01.04.96) US

(71) Applicant: UNIVERSITY OF UTAH RESEARCH FOUNDATION [US/US]; 210 Park Building, Salt Lake City, UT 84112 (US).

(72) Inventors: THOMAS, Charles, L.; 217 North 900 West, Salt Lake City, UT 84116 (US). LEE, Cheol, H.; 814 University Village, Salt Lake City, UT 84102 (US). KAZA, Srinivas; 611 Medical Plaza, Salt Lake City, UT 84112 (US). GAFFNEY, Thomas, M.; Apartment 12, 133 South 900 East, Salt Lake City, UT 84102 (US).

(74) Agents: BOND, Laurence, B. et al.; Trask, Britt & Rossa, P.O. Box 2550, Salt Lake City, UT 84110 (US).

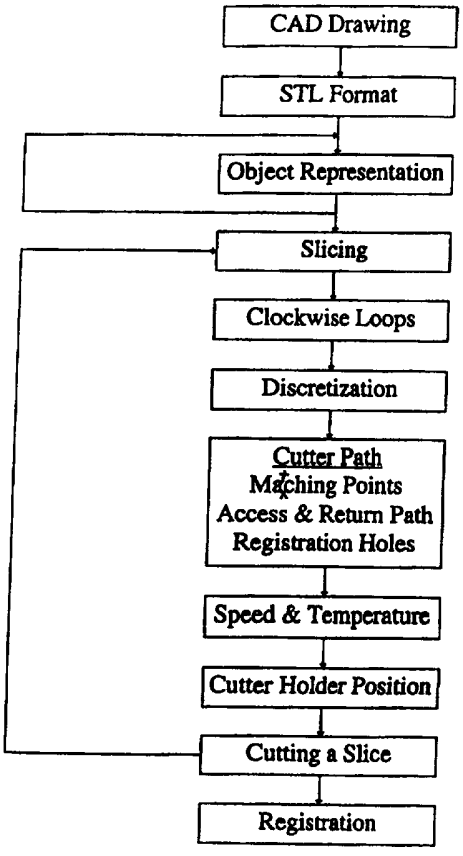
(81) Designated States: CA, JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: HIGHER ORDER CONSTRUCTION ALGORITHM METHOD FOR RAPID PROTOTYPING

(57) Abstract

A rapid prototype modeling system (20) operates to first electronically decompose a discrete part represented by an STL file into electronic layers, using a paradigm characterized by a higher than zero order of fit with respect to the surface of the part. Physical layers are generated from the electronic layers by mechanical devices constructed and arranged to operate in accordance with the paradigm. The physical layers are then stacked appropriately to create a physical model of the discrete part.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

HIGHER ORDER CONSTRUCTION ALGORITHM METHOD FOR RAPID PROTOTYPING

TECHNICAL FIELD

This invention relates to rapid prototype modeling techniques. It is particularly directed to the use of higher order algorithms for the construction of solid prototype parts derived from 3D CAD software-generated models.

BACKGROUND ART

To remain competitive in a global market, manufacturers are called upon to bring a variety of products to market with continually decreasing lead times and development costs. These commercial realities are a driving force demanding the development of improved rapid prototyping techniques.

A rapid prototyping technique (RP) is considered for purposes of this disclosure to include any automatic technique for producing a solid three dimensional object from a computer assisted drawing (CAD) representation of the object. RP systems typically require minimal skill and interaction on the part of the operator. Most RP systems are quite flexible, allowing the creation of practically any shape the designer can imagine. RP thus inherently allows the creation of models that are difficult or impossible to construct by other techniques. While there exist significant differences between RP systems, RP generally produces parts that fit within a build volume of less than about a cubic meter (typically a third of a cubic meter), builds these parts at a rate of approximately 2.54 vertical centimeters (one vertical inch) per hour, and normally provides an accuracy within the range of about 0.2% - 0.5% of model dimensions, with a lower limit of 0.0127 centimeter (cm) (0.005 inches). The cost for a part from an RP service bureau ranges up to several thousand dollars for a part that fills the build volume.

RP techniques are useful for producing small parts, such as disk drive cases, telephone hand sets, piston rods and the like. However, if the part to be prototyped is large; e.g., a car, the wing of an aircraft, or the support housing for a commercial power generator, the part must be generated in pieces, and the pieces then assembled. The resulting part typically requires days to construct, and its cost of construction is proportionately very high.

The paradigm currently used by most RP techniques has a limited range of applicability. Parts larger than one cubic centimeter and smaller than one cubic meter are efficiently produced. Outside this range, prototypes either lose accuracy, or become expensive and time consuming to construct.

Most RP techniques that are currently in use operate under a common construction paradigm. Working from a stereolithography format, faceted approximation of the solid model, the model is first decomposed into a series of parallel cross sections spaced at a distance equal to the eventual thickness of the construction layers. The part is then constructed by generating these layers from some solid material, and bonding the layers together. The cross sections are physically realized by means of a construction technique characteristic of the specific RP system utilized.

There are several commercially available rapid prototyping processes, each using a unique production process for creating prototype models. A few of the more popular rapid prototyping techniques are Stereolithography (SLA), photo exposure layering (PEL), laminated object manufacturing (LOM), selective laser sintering (SLS), and fused deposition modeling (FDM). SLA-based systems solidify each layer from liquid photopolymer. LOM-based systems cut each layer from a sheet material. SLS-based systems fuse layers of powder into each cross section shape. These techniques construct each layer with essentially vertical edges, creating a generalized cylinder. For purposes of this disclosure, they are considered to rely upon the "vertical cylinder paradigm" (VCP). These vertical cylinders result in a stepped profile in a prototype representing a part that does not in fact have vertical edges. VCP produces a good representation of objects that have only vertical surfaces in at least one axis. The accuracy of the representation decreases as the surface shape becomes more general and the layer thickness employed increases.

An example of an alternative paradigm currently in use for RP applications is in the "Protoform" software produced by Pentari. The operating paradigm used by Protoform is referred to in this disclosure as the "flat pattern paradigm" (FPP). By this RP system, a solid model is tessellated into flat polygon surfaces and then unfolded into a flat pattern. Tabs are generated at the cut edges, and the resulting pattern is plotted. The pattern is cut, folded, and tabs are joined, creating a three

dimensional object. RP using FPP produces only the surface of an object, thereby requiring very little construction material. Accurate representations of curved or domed surfaces require very small tessellations, which are difficult to create. Thus, a relatively crude approximation is produced by this approach applied to non-planar surfaces.

While each of the current RP systems is useful, each has at least one significant limitation. Some general examples of these limitations include: large procurement cost, expensive prototype production costs, and the use of resins and polymers that produce harmful gasses. Most importantly, all of them are limited to build volumes of less than about a cubic meter, and in some cases less than 0.02832 cubic meter (a cubic foot (ft^3)).

As presently understood, merely modifying existing RP machines would not enable effective operation significantly outside the currently available manufacturing envelope. Making vertical cylinders faster could presumably expand the envelope, but an RP machine that is efficient for producing 10 cm prototypes cannot be expected efficiently to produce 10 meter prototypes. There remains a need for an RP system capable of expanding the build volume of current systems. Moreover, in the construction of large objects, it would be preferable to operate within construction paradigms capable of realizing a closer fit between the prototype and the design part.

DISCLOSURE OF INVENTION

This invention provides an RP system which avoids many of the limitations of existing systems. It expands the build volume of prototype parts through the use of higher order paradigms. It also provides a system, including machine components, capable of implementing software such as that set forth at pages 14 through 117.

Viewed broadly, a rapid prototype system of this invention comprises the steps of electronically decomposing an electronic model of an object into a first series of electronic layers in accordance with a paradigm characterized by a higher than zero order fit with respect to the surface of said object; generating a second series of physical layers from a construction material by mechanical means constructed and arranged to operate in accordance with that paradigm, the second

series corresponding layer by layer to the first series; and constructing a physical prototype of the object by assembling the physical layers.

The mechanical means for practicing the invention is structured and arranged to provide four degrees of freedom in positioning a cutting device with respect to sheets of construction material. The mechanical means is generally constructed and arranged to accept and manipulate sheets of construction material 1.22 meters (four feet) wide, or more, and is operated by control means responsive to software equivalent to that set forth at pages 14 through 117.

The mechanical means generally includes support structure, operable to move a sheet of construction material forwards and backwards along a travel path in a plane; an electronically heated wire with a first end held by a first linear positioning device and a second end held by a second linear positioning device, the first and second positioning devices being supported to position the wire transverse the plane to intersect and cut a sheet of construction material traveling on the support structure; first structure, mounting the first positioning device such that it may travel back and forth transverse the travel path; second structure, mounting the second positioning device such that it may travel independently of the first positioning device back and forth transverse the travel path; and electronic control means, responsive to operating software, for controlling the relative movements of a sheet of construction material on the support structure, the first positioning means and the second positioning means. The mechanical means may also include heat control means, responsive to the operating software, constructed and arranged to adjust the temperature of the wire. The operating software is ideally equivalent to that set forth at pages 14 through 117.

According to this invention, a rapid prototype modeling system operates to first electronically decompose a discrete part represented by an STL file into electronic layers, using a paradigm characterized by a higher than zero order of fit with respect to the surface of the part. Physical layers are generated from the electronic layers by mechanical devices constructed and arranged to operate in accordance with the paradigm. The physical layers are then stacked appropriately to create a physical model of the discrete part.

In practice, an electronic model of an object is electronically decomposed into a first series of electronic layers in accordance with a paradigm characterized

by a higher than zero order fit with respect to the surface of the object. A second series of physical layers is generated from a construction material, the second series corresponding layer-by-layer to the first series. This second series of physical layers is generated by mechanical means constructed and arranged to operate in accordance with the paradigm relied upon for decomposing the electronic model. A physical prototype of the object is then constructed by assembling the physical layers. The paradigm has a higher order of fit than that of the vertical cylinder paradigm (VCP) common to RP. It may be selected from either the trapezoidal cylinder paradigm (TCP) or the arc cylinder paradigm (ACP), although other higher order construction algorithms (HOCA) are operable.

In mathematical terms, VCP is considered to produce a zero order curve fit to the model surface at the edge of each layer. By increasing the order of this fit, this invention produces a more accurate representation at the same layer thickness, and/or increased speed of construction at no less accuracy by increasing the thickness of the construction layers.

A principle similar to that of the trapezoidal rule, often used in numerical integration, is applied by this invention to the construction of layered parts, thereby producing a first order curve fit at the layer edge. To apply this trapezoidal cylinder paradigm (TCP), an electronic model is first decomposed into a series of cross sections in conventional RP fashion. Rather than reconstructing each layer based on information from a single cross section, however, the layer is constructed as a linear blend between two consecutive cross sections. A slice taken vertically through this reconstructed layer would produce a trapezoid. Thus, these layers can be considered generalized trapezoidal cylinders. While VCP allows reconstruction of vertical-edged parts with very little error, the trapezoidal cylinder paradigm (TCP) additionally allows reconstruction of parts from models generated with ruled surfaces with very little error. Modeling of double curved surfaces by TCP will still produce error, but much less error than by VCP.

This invention alternatively applies an arc cylinder paradigm (ACP), whereby a variable radius arc is fit to the edge of a layer. This curve fit is essentially of second order at the layer edge. Layers produced using ACP can be considered generalized cylinders with concave or convex curved edges. The radius of curvature will generally vary with position along the edge, possibly transitioning

from convex to concave on a single layer. An algorithm is required to select the radius that will minimize the error between the model and the reconstructed layer.

It is within contemplation to utilize paradigms using even higher order curve fits for layer edges. For example, by requiring the surface gradient at the edge of adjoining layers to be equal, a reconstructed part will be characterized by a smooth transition from layer to layer. A higher order version of the FPP would further enhance the reconstruction of curved surfaces.

This invention contemplates the design and implementation of rapid prototyping devices using higher order construction algorithms (HOCA) for use in rapid prototyping of large scale master models. Examples of practical applications include full scale models of automobiles, boat hulls, wind turbine air foils, or molds for use with room temperature layup of composites. Lost foam casting and other conventional casting techniques may be used to convert large polystyrene foam models into metal castings.

A specific embodiment of this invention utilizes TCP to build prototypes from "trapezoidal disks." TCP creates linear interpolation between the upper and lower cross sections which define one layer or slice of a CAD model being constructed. The height of each trapezoid is equal to the thickness of material used to construct the prototype model. By using thick layers, construction time for the overall model is significantly reduced. It is practical, for example, to cut layers from 1.22 meters by 2.44 meters by 2.54 cm (4 ft by 8 ft by 1 inch) thick polystyrene foam insulation boards. Stacking and bonding of the layers may be done manually or by automated systems.

BRIEF DESCRIPTION OF DRAWINGS

In the drawings, which illustrate what is currently regarded as the best mode for carrying out the invention:

FIG. 1 is a pictorial view of a machine embodying the invention, with an end plate removed to permit better visibility of components;

FIG. 2 is a fragmentary view of the machine of FIG. 1, illustrating components which manipulate the positions of a sheet of construction material and a cutter;

FIG. 3 is a fragmentary view of components of a cutting assembly of the machine of FIG. 1, illustrating alternative orientations of a cutter wire;

FIG. 4 is a flow diagram of the method of this invention;

FIG. 5 is an illustration of an electronic model of a vase;

FIG. 6 is a fragmentary illustration of the model of FIG. 5 decomposed into electronic layers;

FIGS. 7, 8 and 9 are diagrammatic illustrations of the respective fits with respect to a layer surface offered by VCP, TCP and ACP paradigms in accordance with the invention;

FIG. 10 is a plot of the percent error of surface fit the paradigms of FIGs 7, 8 and 9 as a function of layer thickness;

FIG. 11 is an illustration of an electronic model of an air foil;

FIG. 12 is an illustration of the model of FIG. 11 decomposed into electronic layers; and

FIG. 13 is a pictorial view of an assembled machine incorporating the components illustrated by FIGS. 1, 2 and 3 for operation in accordance with the software set forth at pages 14 through 117.

BEST MODE FOR CARRYING OUT THE INVENTION

As illustrated by FIG. 1, a machine of this invention, designated generally 20, may utilize an electronically heated wire 22 (FIGS. 2 and 3) to cut rigid sheets 24. The sheets 24 are manipulated in a forward or reversed direction along a selected travel axis, A--A, atop horizontal drive rollers 26, 28. Above each drive roller is a free floating nip/idler roller 30, 32 to ensure contact between the drive rollers 26, 28 and the foam sheet 24.

The heated wire 22 is shown by FIG. 2 positioned transverse the sheet 24 approximately vertically. The attitude of the wire 22 with respect to vertical transverse the travel axis A--A may be adjusted as illustrated by FIG. 3, through the relative positioning of a first linear positioning device 36, mounted to travel along a first, lower rail 37 and a second linear positioning device 38, mounted to travel along a second, upper rail 39 approximately parallel the first rail 37. As best shown by FIG. 2, the attitude of the wire 22 with respect to vertical parallel the travel axis A--A is adjustable by means of a third linear motion device 40, mounted

to travel approximately normal the second rail 39. In operation, the linear motion devices 36, 38, 40 are driven in coordination with the manipulations of the sheet 24 along the axis A--A. Motion control for each linear positioning device 36, 38, 40, as well as for the drive rollers 26, 28 is supplied by stepper motors (not shown), which receive input signals from a software control program set forth at pages 14 through 117.

The combination of these three motion devices 36, 38, 40 provide the illustrated mechanism 20 three degrees of freedom. The powered drive rollers 26, 28 feed the construction sheet 24 into the cutting wire 22, thereby contributing a fourth degree of freedom.

Operation of the illustrated device is similar to the operation of a roll plotter, but with an additional two degrees of freedom in an x-y coordinate system. The drive rollers 26, 28 and lower linear motion device 36 act in the fashion of a roll plotter to position the base 42 of the cutting wire anywhere on the bottom surface 44 (FIG. 3) of the foam sheet 24. The upper two linear motion devices 38, 40 add two more degrees of freedom, allowing the upper end 46 (FIG. 2) of the cutter wire 22 to be positioned independently of the lower end 42.

The respective ends 42, 46 of the electrically heated cutting wire 22 are directly attached to two of the three independent linear motion devices 36, 38. A coil spring 48 connects the upper end 46 of the cutting wire 22 to the device 38. The coil spring 48 accommodates length adjustments required of the wire 22 as it is positioned for making linearly interpolated cuts at an angle from the vertical plane. Temperature of the wire 22 is adjusted by varying the electric current applied to it. The cutting speed is adjusted to minimize cutter deflections caused by the construction material, in the illustrated instance, foam plastic board 24.

The main tasks of the machine control software set forth at pages 14 through 17 are to recognize coordinates representing a CAD 3D solid object, and to find a sequence of cutter motions to produce the required cross sectional slices. Once an object is created in a CAD system, it can be easily saved in STL format, the file format currently utilized for many RP techniques. In selecting an STL file format, it should be recognized that the accuracy of an approximated object saved in the STL file is the maximum accuracy that the prototype machine will be able to produce. The machine uses geometry information saved in the STL file, not the

CAD drawing file. The invention applied to files of greater accuracy will produce more accurate prototypes. Algorithms that generate cross sections directly from the CAD model are within contemplation.

After reading an STL file, the control software regenerates the object on a computer screen. A user may then determine the most efficient slice direction by rotating, transforming, and scaling the object on screen. The software then slices the object into layers equal to a selected material thickness, and defines straight line segments which represent the edges of the object on the cut plane. The end result of this slicing algorithm is a series of independent cross sections.

For the actual part layers to be cut, it is necessary to define synchronized positionings of the cutter wire 22. The cutter wire 22 must be repositioned over time, in coordination with movements of a layer 24 of construction material, to produce somewhat different cross-sectional configurations at the upper 50 and lower 44 surfaces of the layer 24 in a simultaneous coordinated motion. Defining the cutter path involves optimally matching points on the perimeters of the top and bottom surface cross sections of a slice. Cutter paths are generally determined by matching each top cross section point with the closest bottom cross section point. This distance minimization approach is generally suitable because it can accommodate virtually any shape. This technique can thus generate thick slices with small error, and is suitable for prototyping large objects rapidly.

The defined cutter path corresponds to respective cutting paths on the upper and lower surfaces of a construction sheet 24. The actual hot wire 22 extends beyond the upper 50 and lower 44 surfaces and pivots about points associated with anchor rods 52, 54 displaced significantly above and below those surfaces. Accordingly, after finding cutter paths, the software calculates the appropriate positions of the respective cutter pivot points to trace the proper cutter path on the material. Finally, the software converts position coordinates into a string of pulses to drive the stepper motors. A flow chart for the complete algorithm is presented by FIG. 4.

Ideally, the cutting machine should be constructed and arranged to have a pen up function. The machine is then capable of moving the cutter device from point to point without cutting material, thereby avoiding the need for the layout of access paths to loops and return paths to a home position. When a large object is

built, registration of slices is an important concern. Adding paths and holes for registration can be done by the software following rules defined by the user.

Two main parameters controlling the quality of the cut surface are cutter speed and temperature. Speed is defined by the pulse rate sent to the stepper motors which is related to the torque of the motors. The cutter temperature is a function of electric current going through the cutter wire 22. Undesirable combinations of speed and temperature can result in local areas where the cutter melt radius becomes excessive. Cutting too fast or too cool results in excessive deflection of the hot wire which also reduces accuracy. Optimal combination of these parameters depends on material properties and the characteristics of the object geometry, and will generally be determined through careful experiments. It has been observed that maintaining a constant speed and temperature over the working period is beneficial.

EXAMPLE 1.

A simple vase 55 (FIG. 5) was designed to test construction accuracy as a function of layer thickness for the construction paradigms: VCP, TCP, and ACP. Under the assumption that the speed of construction of a layer is independent of layer thickness, it follows that parts built with thicker layers will require less time to construct.

The vase shape may be represented mathematically, using a series of two dimensional (x,y) arcs and lines that are revolved about the y axis:

Outer Surface:

Vertical Dimension (y)	Equation
0 - 28	$\frac{(x-3)^2}{100}+\frac{(y-15)^2}{225} = 1$
28 - 30	$3x + 5y = 164$
31 - 35	$x = 3$
35 - 36	$(x-4)^2 + (y-35)^2 = 1$
36 - 38	$(x-6)^2 + (y-37)^2 = 1$

Inner Surface:

Vertical Dimension (y)	Equation
0 - 3	$x = 0$
3 - 28	$\frac{x^2}{100} + \frac{(y-15)^2}{225} = 1$
28 - 30	$2x + 3y = 94$
30 - 35	$x = 2$
35 - 37	$(x-4)^2 + (y-35)^2 = 4$
37 - 38	$(x-4)^2 + (y-38)^2 = 1$

For this simple shape, as can be seen from FIG. 7, all cross sections are either solid circular cylinders 56 or solid circular annuli 58, and a specific cross section is completely defined by one or two coordinates on the x axis. To apply VCP, an algorithm is needed to select the x coordinates for each layer (one or two numbers). For TCP, each layer requires x coordinates for the upper surface and the lower surface (two or four numbers). ACP requires x coordinates for upper and lower surfaces along with one or two radii (three or six numbers). The algorithms used in this example for each paradigm were:

VCP

The radius of each layer is forced to match the model at the top of each layer.

TCP

The algorithm for TCP utilizes the cross sections generated for VCP. The top cross section for a given layer is identical to the bottom layer of the closest layer above it.

ACP

Arc cylinders are generated from the cross sections generated for VCP. The arc radius is generated by passing an arc through the upper surface point, the lower surface point, and a third point that lies on the model surface at the midpoint of the layer. Implementation was as follows:

1) identify the x coordinate for a new cross section in the center of the layer.

2) define a line segment from the upper surface to the center and a second from the lower surface to the center.

3) The bisecting normals of these line segments intersect at the center of curvature, identifying the radius.

The fits of these construction algorithms are shown schematically in FIGS. 7, 8 and 9, respectively. The accuracy of each algorithm was measured by calculating the error between the volume of material required by the exact model, represented by the solid line 60, and that required by the reconstructed model, represented by the broken lines 62, 64 and 66, respectively. First, the exact volume of the mathematical model for the vase was calculated. Next, the model was decomposed into layers (FIG. 6), and the volume of the combined layers was calculated. This procedure was performed for each algorithm, varying layer thickness from 0.1 mm to 2 mm. The percent error associated with a given algorithm and layer thickness was calculated by:

$$\% \text{ Error} = \left(\frac{V_0 - V_{rc}}{V_0} \right) 100$$

where V_0 is the volume of the mathematical model and V_{rc} is the volume of the reconstructed model.

The data from these calculations are plotted in FIG. 10. As expected, the error is small for any of the paradigms if the layer thickness is very small. As layer thickness increases, the accuracy decreases for each method. The error for VCP increases dramatically faster than the higher order algorithms, exceeding 15 percent error as the layer thickness approaches 1 cm. At 1 cm thickness, both higher order techniques perform at under three percent error.

EXAMPLE 2.

A computer generated model of a NASA LS(l)-417MOD wind turbine air foil 70 is shown in FIGS. 11 and 12. Construction of this air foil was attempted with good success using the equipment described in connection with FIGS. 1-4 assembled as illustrated by FIG. 13. The foil shape is described by a series of complex spline curves 71, which have a spline taper through the height (h) and

width (w). The model is further complicated by a 42° angle of twist along the foil length (l). The total length of the model is 4.57 meters (15 feet). The initial width (at the base 75) of the air foil is 116.8 centimeters (46 inches), reducing to 25.4 centimeters (10 inches) at the tip 77. The height of the air foil starts at 19.84 centimeters (7.81 inches) at the base 75, reducing to 2.84 centimeters (1.12 inches) at the tip 77.

The computer model of the air foil was created using Parametric Technology's Pro/Engineer software. Spline points at various cross sections the air foil were generated from normalized data developed at NASA. These spline points were subsequently read into Pro/Engineer and series of spline curves were fit to the points as shown in Fig 12. Smooth spline surfaces 80 were then generated between the spline curves 71. A computer solid model (Fig. 11) was created from the hollow spline shell. This computer solid model was stored as a standard stereolithography format file. The equipment illustrated by Fig. 13 was used to cut out the 180 layers required to construct the air foil prototype, requiring approximately 17 sheets of 1.22 meters by 2.44 meters by 2.54 centimeters (4 feet by 8 feet by 1 inch) thick sheets of bead board foam. Each layer of foam was then manually registered and bonded.

The accuracy of this air foil prototype was in the range of 0.08 to 0.6 % error, when compared to chord lengths in the computer model. This level of accuracy is considered acceptable, given the size of the prototype and the facet mesh size of the STL file. The surface finish of the model was fairly rough, because of the STL file approximation of the computer model. Further facet refinement of the STL file would correspondingly improve the surface finish.

The air foil computer model was created in approximately 2 hours using Pro/Engineer. Cut out and assembly time for the air foil prototype was estimated at 9 hours.

Reference in this disclosure to details of specific embodiments is not intended to limit the scope of the appended claims which themselves recite those details regarded as significant to the invention. It is intended for the appended claims to embrace equivalents.

```
!************************************************************
! Written by Cheol H. Lee
! University of Utah
! File : 72readme.tru
! Date : 12/96
! Object : General instruction for SM2 control program
!************************************************************
1. General Process
------------------
  - In CAD, a stl file (*.stl) is made.
  - All text in stl are removed and all coords are saved in numeric
    form in a vertex file (*.vtx)
  - An object is sliced and lines that represent outlines of each cross
    section are save in line file (*.ln)
  - All lines in line file are sorted to make closed loops in counter
    clockwise direction and saved in a loop file (*.lp)
  - All lines in loop file are discretized into points which are saved
    in a point file (*.pt)
  - All loops of top and bottom cross sections of a slice are matched and
    cutter movements are determined and saved in a cut file (*.ct)
  - Cutter movements are converted into number of steps to move machine
    which are saved in a step file (*.st)
  - A C program reads step file and generates commands to drive moters
2. Directory Structure
----------------------
  - \                  : all True Basic program files, *.tru
  - \bc31\bin\         : rp-mt4.c
  - \object\           : *.stl, *.vtx, setup.rp
  - \object\[object_name]\ : *.ln, *.lp, *.lpx, *.pt, *.ptx, *.ct, *.ctx
                       *.st, ctreport.txt
  - \object\temp\      : *.st, ctreport.txt
3. File Structures
------------------
  - A CAD s/w makes *.stl file
  - A True Basic program makes all other files from *.vtx to *.st
  - A C program drives machine and is called by a True Basic program
  - *.stl : stl file. made by CAD s/w. text or binary file.
  - *.vtx : vertex file or object file. record file.
        9 records for each triangle (x1,y1,z1,x2,y2,z2,x3,y3,z3)
  - *.ln  : line file. record file. 4 records for each line (x1,y1,x2,y2)
  - *.lp  : loop file. record file. 4 records for each ln (x1,y1,x2,y2)
  - *.lpx : loop index file. record file.  each loop has 10 records
        (start line num, end line num, xd,yd, xu,yu, xl,yl, xr,yr)
        xd,yd=ymin pt. xu,yu=ymax pt. xl,yl=xmin pt, xr,yr=xmax pt
  - *.pt  : point file. record file. 2 records for each pt (x1,y1)
  - *.ptx : point index file. record file. each loop has 10 records
        (start pt num, end pt num, xd,yd,xu,yu,xl,yl,xr,yr)
  - *.ct  : cut file. record file.
        4 records for each cutter movement (xb,yb,xt,yt)
  - *.ctx : cut index file. record file.
        13 records for each matching loop, and 5 rec at the end
        (start num of cut,end num of cut,xbstart,ybstart,
        xtstart,ytstart,xmin,xmax,ymin,ymax,index_b,index_t,max_angle)
        ... (xmin,xmax,ymin,ymax,max_angle for all match loops)
  - *.st  : step file. text file. 4 numbers for each cutter movement
        (x1,x2,y1,y2)
  - setup.rp : setup file. record file. 15 jop setup parameters
```

(object name,num_tri,dh,angle,space_t,space_b,d_loop,d_point
xmax,ymax,zmax,xh,yh,decimal,skip)

- ctreport.txt : cut report. text file.
16 global parameters and 4 parameters for each slice
(object$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,
xmax,ymax,zmax,xh,yh,decimal,skip,slice,size of each slice,
max angle of each slice, num of loops of each slice)

4. Program Files

- 72readme.tru : general instruction for SM2 control program

- 72-main.tru : main program
- 72-menu.tru : prepare menus

- 72-1-vtx.tru : *.stl -> *.vtx
- 72-2-ln1.tru : *.vtx -> *.ln
72-2-ln2.tru
- 72-3-lp1.tru : *.ln -> *.lp
72-3-lp2.tru
- 72-4-pt.tru : *.lp -> *.pt
- 72-5-ct1.tru : *.pt -> *.ct
72-5-ct2.tru :
- 72-6-st.tru : *.ct -> *.st

- rp-mt4.c : *.st -> step pulse

- 72-gvt.tru : graphics for *.vtx
- 72-gln.tru : graphics for *.ln, *.lp, and *.pt
- 72-gct.tru : graphics for *.ct and *.st
- 72-ut.tru : utilities to read index file and cut report file

!**********
! End of File
!**********

```
!*********************************************************************
! Written by Cheol Lee
!
! File: 72-ut.tru
! Date: 10/95
!
! Object : display contents of index file and cut report file
! Input  : *.lpx, *.ptx, *.ctx, ctreport.txt
! Output : display info on screen
!*********************************************************************
! MENU
!
! - Cut report is in object\[object_name]\ctreport.txt text file
!     (object$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,
!      xmax,ymax,zmax,xh,yh,decimal,skip,slice,size of each slice,
!      max angle of each slice, num of loops of each slice)
!*********************************************************************

library "utillib.tru"

name$="sink"
start,end=1
call auto_read_ct_index(name$,start,end)
!call read_cut_report
stop
num$="19"
call record_to_text_file("temp\"&num$&".st","temp\"&num$&".txt")
end
EXTERNAL
!*********
! Summary (72-ut)
!*********
!    SUB read_index_file(in_file$)
! SUB auto_read_lp_index(name$,start,end)
! SUB auto_read_pt_index(name$,start,end)
!    SUB read_cut_index_file(file_ctx$)
! SUB auto_read_ct_index(name$,start,end)
! SUB cut_report(object$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
!          & xmax,ymax,zmax,xh,yh,decimal,skip,slice,start,end)
! SUB read_cut_report(name$)
!*****************************
! Subroutine: read_index_file
!*****************************
SUB read_index_file(file_in$)    ! i

   ! Read *.lpx,*.ptx index file and display contents
   ! file_in$  : *.lpx, or *.ptx index record file

 CALL open_file(#1,file_in$,error)
 IF error=1 THEN exit sub

 ASK #1:Filesize num_record

 IF num_record=0 THEN
   PRINT "No record!!  blank record file!"
   EXIT SUB
 END IF
```

```
  RESET #1:Begin
  PRINT "File name: ";file_in$
  PRINT "start/end","low(x/y)","high(x/y)","left(x/y)","right(x/y)"
  PRINT "-------------------------------------------------------------------------"

  FOR i=1 TO num_record/10
     READ #1:r1,r2,r3,r4,r5,r6,r7,r8,r9,r10
     PRINT i;")";r1,r3,r5,r7,r9
     PRINT i;")":r2,r4,r6,r8,r10
  PRINT "-------------------------------------------------------------------------"
  NEXT i
  CLOSE #1
  END SUB
  !*******************************
  ! Subroutine: auto_read_lp_index
  !*******************************
  SUB auto_read_lp_index(name$,start,end)  ! i(3)
     ! display *.lpx file from start.lpx to end.lpx
     ! name$ : object name
     ! start,end : start and end file number for *.lpx
  FOR i=start TO end
     file_in$="object\" & name$ & "\" & str$(i) & ".lpx"
     CLEAR
     CALL read_index_file(file_in$)
     GET KEY getkey
  NEXT i
  END SUB
  !*******************************
  ! Subroutine: auto_read_pt_index
  !*******************************
  SUB auto_read_pt_index(name$,start,end)  ! i(3)
     ! display *.ptx file from start.ptx to end.ptx
     ! name$ : object name
     ! start,end : start and end file number for *.ptx
  FOR i=start TO end
     file_in$="object\" & name$ & "\" & str$(i) & ".ptx"
     CLEAR
     CALL read_index_file(file_in$)
     GET KEY getkey
  NEXT i
  END SUB
  !*******************************
  ! Subroutine: read_cut_index_file
  !*******************************
  SUB read_cut_index_file(file_ctx$)  ! i
     ! Read *.ctx index file and display contents
     ! file_ctx$  : *.ctx index record file
  CALL open_file(#1,file_ctx$,error)
     IF error=1 THEN exit sub
  ASK #1:Filesize num_record
  IF num_record=0 THEN
    PRINT "No record!!  blank record file!"
    EXIT SUB
  END IF

  RESET #1:Begin
  PRINT "File name: ";file_ctx$
```

```
PRINT "start/end","start(xb/yb)","start(xt/yt)","xmin/xmax","ymin/ymax"
PRINT "index(b/t)","angle","xsize","ysize"
PRINT "-------------------------------------------------------------------------"

FOR i=1 TO (num_record-5)/13
  READ #1:r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13
  PRINT i;")";r1,r3,r5,r7,r9
  PRINT i;")";r2,r4,r6,r8,r10
  PRINT
  PRINT i;")";r11,r13,r8-r7,r10-r9
  PRINT i;")";r12
PRINT "-------------------------------------------------------------------------"
NEXT i
CLOSE #1
END SUB

!*********************************
! Subroutine: auto_read_ct_index
!*********************************
SUB auto_read_ct_index(name$,start,end)  ! i(3)
  ! display *.ctx file from start.ctx to end.ctx
  ! name$ : object name
  ! start,end : start and end file number for *.ctx
FOR i=start TO end
   file_in$="object\" & name$ & "\" & str$(i) & ".ctx"
   CLEAR
   CALL read_cut_index_file(file_in$)
   GET KEY getkey
NEXT i
END SUB
!***********
! cut_report
!***********
SUB cut_report(object$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
        & xmax,ymax,zmax,xh,yh,decimal,skip,slice,start,end)
                                  ! all input
   ! make cut report (ctreport.txt) in object directory
   ! object$, num_tri : object name. number of triangles
   ! dh, angle       : foam thickness. maximum cut angle
   ! space_t, space_b : space bw control point and top or bot surface
   ! d_loop, d_point  : min distance bw loops.  discrete point distance
   ! x,y,zmax : object size
   ! xh,yh    : home position in x and y direction
   ! decimal, skip : round decimal. number of skip points.
   ! slice, start,end : num of slices. start, end num of slice to report

 cutreport$="object\" & object$ & "\ctreport.txt"
 OPEN #5:name cutreport$,access outin, create newold,org text
 ERASE #5
 PRINT #5:"object  :  ";object$
 PRINT #5:"triangles: ";num_tri
 PRINT #5:"slices  : ";slice
 PRINT #5:"foam. angle      : ";dh,angle
 PRINT #5:"space (t/b)      : ";space_t,space_b
 PRINT #5:"loop dis. pt res  : ";d_loop,d_point
 PRINT #5:"skip. decimal    : ";skip,decimal
 PRINT #5:"home (xh/yh)     : ";xh,yh
```

```
PRINT #5:"size    : ";"x =";xmax;"  y =";ymax;"  z =";zmax
PRINT #5:"-----------------------------------------------------------------------"
PRINT #5:"slice #"," dx"," dy"," angle"," loops"
PRINT #5:"-----------------------------------------------------------------------"

count=0
FOR i=start TO end
   file_ctx$="object\" & object$ & "\" & Str$(i) & ".ctx"
   OPEN #1:name file_ctx$,access input
   ASK #1:filesize fsize
   loops=(fsize-5)/13
   RESET #1:record fsize-4
   READ #1:xmin,xmax,ymin,ymax,max_angle
   PRINT #5:"#";i,xmax-xmin,ymax-ymin,max_angle,loops
   CLOSE #1

   count=count+1
   IF count>=4 THEN
     PRINT #5:"-----------------------------------------------------------------------"
     count=0
   END IF
NEXT i

PRINT #5:"-----------------------------------------------------------------------"
PRINT #5:"end of record"
PRINT "cutreport.txt for ";object$;" is made. "
CLOSE #5
END SUB

!*****************************
! Subroutine: read_cut_report
!*****************************

SUB read_cut_report(name$)  ! o
   ! display contents of cut report file
   ! name$ : object name

CLEAR
cutreport$="object\" & name$ & "\ctreport.txt"
CALL open_file(#1,cutreport$,error)
IF error=1 THEN exit sub
count=0
DO WHILE MORE #1
   INPUT #1:text$
   PRINT text$
   count=count+1
   IF mod(count,24)=0 THEN
     get key key
     CLEAR
   END IF
LOOP
CLOSE #1
END SUB
!*************
! End of File
!*************
```

```
!***********************************
! Written by Cheol H.Lee
! File: 71-main.tru
! Date: 6/96
! RP Shape Maker 2 control program
!***********************************
! MEMO
! - run ixtsr m:10000 for small memory.  default is 52K
! - *.stl, *.vtx files are in "object\" directory
! - *.ln, *.lp, *.pt, *.lpx, *.ptx files are in "temp\" directory
! - setup.rp is in program directory
! - index file (*.ptx, *.lpx): num_start,num_end,xd,yd,xu,yu,xl,yl,xr,yr,..
!                    10 record for each loop
! - cut file (*.ct) : xmin,xmax,ymin,ymax,max_angle
!                5 records at the end of file
! PROS
! - maintain status file, and loaded and updated
! - menu driven, ststus list
! - window size and object size are different option.
!   when change object, window and object size are updated
!*************************************************
LIBRARY "72-1-vtx.tru"
LIBRARY "72-2-ln1.tru"
LIBRARY "72-2-ln2.tru"
LIBRARY "72-3-lp2.tru"
LIBRARY "72-3-lp1.tru"
LIBRARY "72-4-pt.tru"
LIBRARY "72-5-ct2.tru"
LIBRARY "72-5-ct1.tru"
LIBRARY "72-6-st.tru"
LIBRARY "72-ut.tru"
LIBRARY "72-gvt.tru"
LIBRARY "72-gln.tru"
LIBRARY "72-gct.tru"
LIBRARY "0-menu.tru"
!LIBRARY "72-menu.trc"
LIBRARY "3dlib.trc"
LIBRARY "utillib2.tru"
LIBRARY "utillib1.tru"
LIBRARY "shell.tru"
!LIBRARY "72-1-vtx.trc"
!LIBRARY "72-2-ln1.trc"
!LIBRARY "72-2-ln2.trc"
!LIBRARY "72-3-lp2.trc"
!LIBRARY "72-3-lp1.trc"
!LIBRARY "72-4-pt.trc"
!LIBRARY "72-5-ct2.trc"
!LIBRARY "72-5-ct1.trc"
!LIBRARY "72-6-st.trc"
!LIBRARY "72-ut.trc"
!LIBRARY "72-gvt.trc"
!LIBRARY "72-gln.trc"
!LIBRARY "72-gct.trc"
!LIBRARY "72-menu.tru"
!LIBRARY "3dlib.trc"
!LIBRARY "utillib2.trc"
!LIBRARY "utillib1.trc"
!LIBRARY "shell.tru"
```

```
DIM a(0,0),b(0,0),index(10,10)

x1s,x2s,y2s=800
y1s=120.63
!num_var=26
!***************
! Main Program
!***************

SET MODE "vga"

CALL read_setup (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip)
!---------------
! menu job loop

DO

MAT REDIM a(0,0),b(0,0)        ! save memory

!------
! menu

rwin=.8
IF xmax*rwin>ymax THEN  ! window size
  xwin=xmax
  ywin=xmax*rwin
  zwin=zmax
ELSE
  xwin=ymax/rwin
  ywin=ymax
  zwin=zmax
END IF

CALL status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip)
CALL main_menu (job)
CLEAR
SET COLOR "white"

!----------------------
! main job description

SELECT CASE job

CASE 0          ! quit
  STOP

CASE 1          !
  DO
!exit do ! remove dos shell for memory----------
    CLEAR
    PRINT "Press (0) to return to main menu."
    INPUT PROMPT "DOS command: ":com$
    IF com$="0" THEN
      EXIT DO
    ELSE
```

```
    CALL shell(com$)
   END IF
   GET KEY getkey
  LOOP

 !-----------
 ! job setup

CASE 2              ! change job setup

  DO

  !-------------------------------------------------------------
  ! setup menu

  CALL status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip)

  CALL setup_menu(job_setup)
  CLEAR
  SET COLOR "white"

  SELECT CASE job_setup

    CASE 0          ! return to main menu
     EXIT DO

    CASE 1          ! foam thickness
     PRINT "current foam thickness: ";dh
     INPUT PROMPT "Foam thickness (cm)?  ":dh

    CASE 2          ! max cutter angle
     PRINT "current cutter angle: ";angle
     INPUT PROMPT "Maximum cutter angle (degree)?  ":angle

    CASE 3          ! bot space
     PRINT "current bottom space: ";space_b
     INPUT PROMPT "Bottom space from the foam (cm)?  ":space_b

    CASE 4          ! top space
     PRINT "current top space: ";space_t
     INPUT PROMPT "Top space from the foam (cm)?  ":space_t

    CASE 11         ! loop distance
     PRINT "current loop distance: ";d_loop
     INPUT PROMPT "Minimum distance between loops (cm)?  ":d_loop

    CASE 12         ! point distance
     PRINT "current point distance: ";d_point
     INPUT PROMPT "Maximum distance between points (cm)?  ":d_point

    CASE 13         ! skip
     PRINT "current skip points: ";skip
     INPUT PROMPT "Skip points?  ":skip

    CASE 14         ! decimal
     PRINT "current rount decimal: ";decimal
     INPUT PROMPT "Round decimal?  ":decimal
```

```
      CASE 21          ! home position
        PRINT "current home position : ";xh;yh
        INPUT PROMPT "Home position x,y (cm)?  ":xh,yh

      CASE 31          ! save setup
        CALL save_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
          & xmax,ymax,zmax,xh,yh,decimal,skip)
        EXIT DO

      CASE ELSE
        PRINT "Job is not available. Select again "
        GET KEY key
        EXIT SELECT
    END SELECT
    LOOP

!-------------------------------------------------------------
! object

CASE 3              ! combine STL files
    CALL shell("dir/w/o:n object\*.stl")
    PRINT "Press (0,0,0) to cancel."
    INPUT PROMPT "stl_in1, stl_in2,Ostl_in3?  ":in1$,in2$,out$
    IF in1$="0" and in2$="0" and out$="0" THEN EXIT select
    PRINT "Wait..."
    CALL combine_stl(in1$,in2$,out$)

CASE 4              ! append STL files
    CALL shell("dir/w/o:n object\*.stl")
    PRINT "Press (0,0) to cancel."
    INPUT PROMPT "stl_inout, stl_in.":inout$,in$
    IF inout$="0" and in$="0" THEN EXIT select
    PRINT "Wait..."
    CALL append_stl(inout$,in$)

CASE 5              ! stl to object
    CALL shell("dir/w/o:n object\*.stl")
    PRINT "Press (0) to cancel."
    INPUT PROMPT "Object name?  ":name$
    IF name$="0" THEN exit select
    com$="md object\" & name$
    CALL shell(com$)
    CLEAR
    CALL stl_to_vtx(name$,num_tri,xmax,ymax,zmax)
    CLEAR
    CALL draw_vtx_3d(name$,xmax,ymax,zmax)
    GET KEY key

CASE 6              ! select object
    CALL shell("dir/w/o:n object\*.vtx")
    temp$=name$
    PRINT "Press (0) to cancel."
    INPUT PROMPT "Object name?  ":name$
    IF name$="0" THEN
      name$=temp$
      exit select
    END IF
```

```
    CALL object_size(name$,xmin,xmax,ymin,ymax,zmin,zmax,num_tri)

CASE 7          ! rotate object
    PRINT "Press (0,0,0) to cancel."
    PRINT "Current object is ";name$
    PRINT
    INPUT PROMPT "Rotated object name, axes, angle?  ":name_out$,rot_axes$,rot_angle
    IF rot_angle=0 THEN exit select

    CALL rotate_vtx(name$,rot_axes$,rot_angle,xmax,ymax,zmax,name_out$)
    name$=name_out$
    CLEAR
    CALL draw_vtx_3d(name$,xmax,ymax,zmax)
    GET KEY key

CASE 8          ! resize object
    PRINT "Press (0,0,0,0) to cancel."
    PRINT "To convert INCH into CM, scale=2.54."
    PRINT "Current object is ";name$
    PRINT
    INPUT PROMPT "Scaled object name, ratio(x,y,z)?  ":name_out$,xr,yr,zr
    IF xr=1 and yr=1 and zr=1 THEN exit select
    IF xr=0 and yr=0 and zr=0 THEN exit select

    CALL scale_vtx_file(name$,xr,yr,zr,xmax,ymax,zmax,name_out$)
    name$=name_out$

CASE 9          ! registration
    INPUT PROMPT "Start, end file number?: ":start,end
    CLEAR
    CALL registration(name$,start,end,xwin,ywin)

!---------
! a slice

CASE 11         ! object to one line
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "Line file number, cut height?  ":file_ln,slice_h
    IF file_ln=0 and slice_h=0 THEN exit select

    CALL vtx_to_line_h(name$,slice_h,decimal,file_ln)
    CLEAR
    CALL draw_line(name$,file_ln,0,xmax,ymax)
    GET KEY getkey

CASE 12         ! object to lines
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "start, end line files?  ":start,end
    IF start=0 THEN Exit select
    CALL vtx_to_line(name$,start,end,dh,decimal)
    GET KEY getkey

CASE 13         ! lines to loops
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:":int(zmax/dh)+1
```

```
    INPUT PROMPT "start, end loop files? ":start,end
    IF end=0 THEN exit select
    CALL line_to_loop(name$,start,end,d_loop)
    GET KEY getkey

  CASE 14        ! loop to point
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1
    INPUT PROMPT "start, end point files? ":start,end
    IF end=0 THEN exit select
    CALL loop_to_point(name$,start,end,d_point,decimal,1)
    GET KEY key

  CASE 15        ! point to cut
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
    INPUT PROMPT "start, end cut files? ":start,end
    IF start=0 and end=0 THEN exit select
    reverse=0
    INPUT PROMPT "reverse(0/1)? ":reverse

    CALL auto_cut(name$,start,end,reverse,dh,angle,decimal,skip)
    GET KEY getkey

  CASE 16        ! cuts to steps
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
    INPUT PROMPT "start, end step files? ":start,end
    IF start=0 and end=0 THEN exit select
    INPUT PROMPT "number of holes(0-n)? ":num_hole

    CALL auto_step(name$,start,end,num_hole,dh,space_b,space_t,&
      & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)
    GET KEY getkey

  CASE 21        ! remove loop
    PRINT "Press (0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
    INPUT PROMPT "loop file number? ":file_lp
    IF file_lp=0 THEN exit select
    CLEAR
    CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)
    SET COLOR "yellow"
    PRINT
    PRINT " Press (0) to cancel."
    INPUT PROMPT " loop number to be removed? ":lp_num
    IF lp_num=0 THEN exit select
    CALL remove_loop(name$,file_lp,lp_num)
    CLEAR
    CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)

  CASE 22        ! reverse loop
    PRINT "Press (0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
    INPUT PROMPT "loop file number? ":file_lp
    IF file_lp=0 THEN exit select

    CLEAR
```

```
    CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)
    SET COLOR "yellow"
    PRINT "Press (0) to cancel."
    INPUT PROMPT "loop number to be reversed? ":lp_num
    IF lp_num=0 THEN exit select
    CALL reverse_loop(name$,file_lp,lp_num)
    CLEAR
    CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)

CASE 23         ! point to cut (SM1)
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
    INPUT PROMPT "start, end cut files? ":start,end
    IF start=0 and end=0 THEN exit select

    CALL auto_cut_sm1(name$,start,end,decimal)
    GET KEY getkey

!------
! Read

CASE 41         ! read loop index file
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1

    INPUT PROMPT "start, end number? ":start,end
    IF start=0 and end=0 THEN exit select
    CALL auto_read_lp_index(name$,start,end)

CASE 42         ! read point index file
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1

    INPUT PROMPT "start, end number? ":start,end
    IF start=0 and end=0 THEN exit select
    CALL auto_read_pt_index(name$,start,end)

CASE 43         ! read ct index file
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1

    INPUT PROMPT "start, end number? ":start,end
    IF start=0 and end=0 THEN exit select
    CALL auto_read_ct_index(name$,start,end)

CASE 44         ! read coord file
    PRINT "Press (0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "Coordinate file name? ":file_name$
    IF file_name$="0" THEN exit select

    file_name$="object\" & name$ & "\" & file_name$
    CALL read_record_file(file_name$,4)
    GET KEY getkey

CASE 45         ! cut report
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
```

```
    INPUT PROMPT "File number start, end?  ":start,end
    slice=int(zmax/dh)
    CALL cut_report(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip,slice,start,end)

 CASE 46          ! read report
    CALL read_cut_report(name$)
    PRINT
    INPUT PROMPT "print? (1/0)?  ":temp
    PRINT
    IF temp=1 THEN
      com$="print object\" & name$ & "\ctreport.txt"
      CALL shell(com$)
    END IF

!----------------
! automatic view

 CASE 51          ! view object - wire
    CLEAR
    CALL draw_vtx_3d(name$,xmax,ymax,zmax)
    GET KEY getkey

 CASE 52          ! view lines 3D
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    h_bot=(start-1)*dh
    h_top=(end-1)*dh
    CLEAR
    CALL draw_auto_line_3d(name$,h_bot,h_top,dh,xmax,ymax,zmax)
    GET KEY getkey

 CASE 53          ! view lines 2D
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foarm thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    CLEAR
    CALL draw_auto_line_2d(name$,start,end,0,xwin,ywin)

 CASE 54          ! view loops
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    CLEAR
    CALL draw_auto_loop_2d(name$,start,end,0,xwin,ywin)

 CASE 55          ! view points
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    CLEAR
    CALL draw_auto_point_2d(name$,start,end,0,xwin,ywin)
```

```
CASE 56          ! view cuts
   PRINT "Press (0,0) to cancel."
   PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
   INPUT PROMPT "start, end number?  ":start,end
   IF start=0 and end=0 THEN exit select
   CALL draw_auto_cut_2d(name$,start,end,dh,xwin,ywin)

CASE 57          ! view steps
   PRINT "Press (0,0) to cancel."
   PRINT "object height:";zmax;",  foam thickness:";dh;", number of step file:";int(zmax/dh)
   INPUT PROMPT "start, end number?  ":start,end
   IF start=0 and end=0 THEN exit select
   FOR i=start TO end
     CLEAR
     CALL draw_st_3d(name$,i,dh,space_b,space_t,xwin,ywin,x1s,x2s,y1s,y2s,xh,yh)
     GET KEY getkey
   NEXT i

CASE 91          ! prepare
   com$="copy object\" & name$ & "\*.st object\temp"
   CALL shell(com$)
   com$="copy object\" & name$ & "\*.txt object\temp"
   CALL shell(com$)

CASE 92          ! read report
   CALL read_cut_report("temp")
   GET KEY getkey

CASE 100         ! prototype
   CALL shell("c:\-clee\bc31\bin\rp-mt4")

CASE ELSE
   PRINT "Job is not available. Select again "
   GET KEY key
   EXIT SELECT

END SELECT
LOOP
END

!*************
! End of file
!*************
```

```
!************************************************
! Written by Cheol H. Lee
!! File: 71-menu.tru
! Date: 6/96
! Library file for menu
!************************************************
set mode "vga"
clear
a=1
if a=1 then

call main_menu (job)
else
call setup_menu(job)
end if
end

EXTERNAL
!*********
! Summary (71-menu)
!*********
! SUB status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
!         & xmax,ymax,zmax,xh,yh,decimal,skip)
! SUB main_menu(job)
! SUB setup_menu(job)
! SUB read_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
!         & xmax,ymax,zmax,xh,yh,decimal,skip)
! SUB save_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
!         & xmax,ymax,zmax,xh,yh,decimal,skip)

!************************
! Subroutine: status
!************************
SUB status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
        & xmax,ymax,zmax,xh,yh,decimal,skip)
ASK free memory mem
memory=round(mem/1000,1)
IF dh<>0 THEN num_slice=int(zmax/dh)

OPEN #9:SCREEN 0.66,1,0,1
  WINDOW #9
  CLEAR

  SET COLOR "white"
!  SET COLOR "red"
  PRINT "JOB SETUP "
  PRINT "*********************************"
  PRINT " Object ";name$;",";num_tri;"tri"
  PRINT "--------------------------------"
  PRINT " Slices     :";num_slice;"slices"
  PRINT " Free memory :";memory;"Kb"
  PRINT
  PRINT " Loop gab   :";d_loop;"cm"
```

```
  PRINT " Point res  :";d_point;"cm"
  PRINT " Point skip  :";skip;"pts"
  PRINT " Decimal    :";decimal;"place"
  PRINT
  PRINT " Cut angle  :";angle;"degree"
  PRINT " Foam       :";dh;"cm"
  PRINT " Bottom space:";space_b;"cm"
  PRINT " Top space  :";space_t;"cm"
  PRINT
  PRINT " home x,y   :";xh;yh
  PRINT
  PRINT " object x:";xmax;"cm"
  PRINT "        y:";ymax;"cm"
  PRINT "        z:";zmax;"cm"
END SUB

!**********************
! Subroutine: main_menu
!**********************
SUB main_menu (job)

   ! main menu for job selection

SET BACK "black"
SET COLOR "white"
OPEN #1:SCREEN 0,0.33,0.1,1
OPEN #2:SCREEN 0.33,0.66,0.1,1
OPEN #3:SCREEN 0,1,0,0.1

!-- 1st column 1111111111111111111111111111111111111111111111111

  WINDOW #1
  CLEAR
SET COLOR "white"
!  SET COLOR "yellow"
  PRINT "RP CONTROL SOFTWARE R72"
  PRINT "by Cheol Lee "
  PRINT

!  SET COLOR "red"
  PRINT "JOB LIST"
  PRINT "*******************************"

!  SET COLOR "white"
  PRINT " 0) Quit"
  PRINT " 1) DOS"
  PRINT " 2) Job setup"
  PRINT " 3) Combine STL"
  PRINT " 4) Append STL"
  PRINT " 5) STL to object"
  PRINT " 6) Select object"
  PRINT " 7) Rotate object"
  PRINT " 8) Resize object"
```

```
   PRINT " 9) Registration"
 !  SET COLOR "green"
   PRINT "SLICE "
 !  SET COLOR "white"
   PRINT " 11) Object to a line "
   PRINT " 12) Object to lines "
   PRINT " 13) Lines to loops "
   PRINT " 14) Loops to points "
   PRINT " 15) Points to cuts"
   PRINT " 16) Cuts to steps"
   PRINT
   PRINT " 21) Remove loop "
   PRINT " 22) Reverse loop "
   PRINT " 23) Points to cuts(SM1) "
!-- 2nd column 22222222222222222222222222222222222222222222222222
   WINDOW #2
   CLEAR
 SET COLOR "white"
 !  SET COLOR "magenta"
   PRINT " Manufacturing Lab "
   PRINT "   Univ. of Utah "
   PRINT " "
   BOX LINES 0,0.8,0.93,1
   PRINT
   PRINT
 !  SET COLOR "green"
   PRINT "READ "
 !  SET COLOR "white"
   PRINT " 41) Loop index"
   PRINT " 42) Point index"
   PRINT " 43) Cut index"
   PRINT " 44) Coord file"
   PRINT " 45) Create cut report"
   PRINT " 46) Read cut report"

 !  SET COLOR "green"
   PRINT "AUTOMATIC VIEW"
 !  SET COLOR "white"
   PRINT " 51) Object"
   PRINT " 52) Lines-3D"
   PRINT " 53) Lines-2D"
   PRINT " 54) Loops"
   PRINT " 55) Points"
   PRINT " 56) Cuts"
   PRINT " 57) Steps"
 !  SET COLOR "green"
   PRINT "SHAPE MAKER 2"
 !  SET COLOR "white"
   PRINT "  91) prepare "
   PRINT "  92) check report"
   PRINT " 100) prototype"
!-- question window
```

```
  WINDOW #3
  CLEAR
SET COLOR "white"
!  SET COLOR "white"
  INPUT PROMPT "Select a job:  ":job
CLEAR
END SUB

!*************************
! Subroutine: setup_menu
!*************************
SUB setup_menu(job)

SET BACK "black"

OPEN #1:SCREEN 0,0.33,0.1,1
OPEN #2:SCREEN 0.33,0.66,0.1,1
OPEN #3:SCREEN 0,1,0,0.1

!-- 1st column 1111111111111111111111111111111111111111111111111

  WINDOW #1
  CLEAR
!  SET COLOR "yellow"
  PRINT "RP CONTROL SOFTWARE R67"
  PRINT "by Cheol Lee "
  PRINT

!  SET COLOR "red"
  PRINT "JOB LIST"
  PRINT "*******************************"

!  SET COLOR "white"
  PRINT " 0) Return to main menu"
  PRINT
  PRINT " 1) Foam thickness"
  PRINT " 2) Max Cut angle  "
  PRINT " 3) Bottom space  "
  PRINT " 4) Top space     "
  PRINT
  PRINT " 11) Min loop gab "
  PRINT " 12) Point resolution"
  PRINT " 13) Skip points "
  PRINT " 14) Decimal point "
  PRINT
  PRINT " 21) Home"
  PRINT
  PRINT " 31) Save setup"

!-- 2nd column 22222222222222222222222222222222222222222222222222
  WINDOW #2
  CLEAR
```

```
 !  SET COLOR "magenta"
   PRINT " Manufacturing Lab "
   PRINT "  Univ. of Utah "
   PRINT " "
   BOX LINES 0,0.8,0.93,1
 !  SET COLOR "white"
 !-- question window
   WINDOW #3
   CLEAR
 !  SET COLOR "white"
   INPUT PROMPT "Select a job:  ":job

 CLEAR
 END SUB

 !************************
 ! Subroutine: read_setup
 !************************

 SUB read_setup (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
        & xmax,ymax,zmax,xh,yh,decimal,skip)
 OPEN #1:name "object\setup.rp",create newold,access outin, org record
 RESET #1:Begin
 WHEN error IN
    READ #1:name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point
    READ #1:xmax,ymax,zmax
    READ #1:xh,yh,decimal,skip
 USE
    exit sub
 END WHEN
 CLOSE #1
 END SUB

 !************************
 ! Subroutine: save_setup
 !************************

 SUB save_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
        & xmax,ymax,zmax,xh,yh,decimal,skip)

 OPEN #1:name "object\setup.rp",create newold,access outin, org record
 ERASE #1
 SET #1:RECSIZE 10
 RESET #1:Begin
 WRITE #1:name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point
 WRITE #1:xmax,ymax,zmax
 WRITE #1:xh,yh,decimal,skip
 END SUB

 !*************
 ! End of File
 !*************
```

```
!*************************************************************
! Written by Cheol H. Lee
! University of Utah
! File : 72-1-vtx.tru
! Date : 6/96
! Object : make object vertex file by reading a stl file
! Input  : *.stl file
! Output : *.vtx file
!*************************************************************
! MEMO
! - *.stl and *.vtx are in "object\" directory
! - All other files are in "object\[name]\" directory
! - *.stl is a text or a binary file made by a CAD s/w
! - *.vtx is a record file having coord of triangles.
!  Each triangle has 9 coord (x1,y1,z1,x2,y2,z2,x3,y3,z3)
!  The min x,y,z in *.vtx are always 0 by shifting coord
!*************************************************************
library "71-gvt.tru"
library "3dlib.trc"
library "71-ut.tru"
library "utillib.tru"
name$="temp"
!name$="head"

!call get_vtx(name$,num_tri)
!    SUB object_size(name$,xmin,xmax,ymin,ymax,zmin,zmax,num_tri)
!    SUB shift_vtx_file(name$,xmin,ymin,zmin)
!call stl_to_vtx(name$,num_tri,xmax,ymax,zmax)
!call scale_vtx_file("s1",2,83,76,50,"temp")
!call combine_stl("d1",0,3)
call rotate_vtx(name$,"x",90,xmax,ymax,zmax,"temp")
!call object_size("temp",xmin,xmax,ymin,ymax,zmin,zmax,num_tri)
print xmin,xmax
print ymin,ymax
print zmin,zmax

!call rotate_vtx(name$,"z",90,xmax,ymax,zmax)
!call draw_vtx_3d(name$,xmax,ymax,zmax)
end

EXTERNAL
!*********
! Summary (72-1-vtx)
!*********
! SUB combine_stl(name_in1$,name_in2$,name_out$)
! SUB append_stl(name_inout$,name_in$)
!       SUB vtx_to_coord(vtx$,#2)
!       SUB text_to_coord(text$,rec_num,#2)
!    SUB get_vtx(name$,num_tri)
!    SUB object_size(name$,xmin,xmax,ymin,ymax,zmin,zmax,num_tri)
!    SUB shift_vtx_file(name$,xmin,ymin,zmin)
! SUB stl_to_vtx(name$,num_tri,xmax,ymax,zmax)

! SUB scale_vtx_file(name_in$,xr,yr,zr,xmax,ymax,zmax,name_out$)
```

```
! SUB rotate_vtx(name_in$,axes$,angle,xmax,ymax,zmax,name_out$)
!************************
! Subroutine: combine_stl
!************************
SUB combine_stl(name_in1$,name_in2$,name_out$) ! i,i,o
   ! Merge two STL files.  Useful for SilverScreen stl files
   ! name_in1$+name_in2$=name_out$

in1$="object\" & name_in1$ & ".stl"   ! input file name
in2$="object\" & name_in2$ & ".stl"
out$="object\" & name_out$ & ".stl"   ! output file name

OPEN #1:Name in1$,Access input,Create old,Org byte
OPEN #2:Name in2$,Access input,Create old,Org byte
OPEN #5:Name out$,Access outin,Create newold,Org byte
ERASE #5

DO WHILE More #1              ! copy contents of first STL file
  READ #1,bytes 1000:rec$
  WRITE #5:rec$
LOOP

DO WHILE More #2              ! append contents of second STL file
  READ #2,bytes 1000:rec$
  WRITE #5:rec$
LOOP

CLOSE #1
CLOSE #2
CLOSE #5
END SUB
!************************
! Subroutine: append_stl
!************************
SUB append_stl(name_inout$,name_in$)  ! i/o,i

   ! Append STL files. useful for SilverScreen stl files
   ! inout$+in$=inout$

inout$="object\" & name_inout$ & ".stl"       ! file names
in$="object\" & name_in$ & ".stl"
OPEN #1:Name inout$,Access outin,Create newold,Org byte
OPEN #2:Name in$,Access input,Create old,Org byte
RESET #1:End
DO WHILE More #2              ! append second file to first file
  READ #2,bytes 1000:rec$
  WRITE #1:rec$
LOOP
CLOSE #1
CLOSE #2
END SUB
```

```
!*************************
! Subroutine: vtx_to_coord
!*************************
SUB vtx_to_coord(vtx$,#2)          ! i,o
   ! Get x, y, z coordinates out of "vertex x y z " text in stl file
   ! vtx$ : "vertex x y z" text in stl file
   ! #2   : *.vtx file channel
valid$="1234567890-+.e"            ! valid char for coord
length=Len(vtx$)
x1=CPos(vtx$,valid$,1)             ! start and end positions of x, y, z
x2=NCPos(vtx$,valid$,x1+1)-1
y1=CPos(vtx$,valid$,x2+1)
y2=NCPos(vtx$,valid$,y1+1)-1
z1=CPos(vtx$,valid$,y2+1)
z2=NCPos(vtx$,valid$,z1+1)-1
x$=vtx$[x1:x2]                     ! x,y,z string
y$=vtx$[y1:y2]
z$=vtx$[z1:z2]
x=Val(x$)                          ! x,y,z coord
y=Val(y$)
z=Val(z$)
WRITE #2:x,y,z                     ! save three vertex coords
END SUB

!*****************************
! Subroutine: text_to_coord
!*****************************
SUB text_to_coord(text$,rec_num,#2)   ! i,o,o
   ! Get x, y, z coordinates from text$ of stl
   ! text$ : one reading text of stl
   ! rec_num : last record number in text$
   ! #2   : *.vtx, output record file channel
rec_num=1    ! initialize

DO
  outer=Pos(text$,"outer",rec_num)   ! pos of start of a triangle
  endloop=Pos(text$,"endloop",outer) ! pos of end of a triangle
  IF outer=0 OR endloop=0 THEN Exit Sub
  triangle$=text$[outer:endloop+6]       ! string for one triangle
  rec_num=endloop+6
  length=Len(triangle$)
  v1=Pos(triangle$,"vertex",1)           ! 1st vertex
  v2=Pos(triangle$,"vertex",v1+6)        ! 2nd vertex
  v3=Pos(triangle$,"vertex",v2+6)        ! 3rd vertex

  v1$=triangle$[v1+6:v2-1]
  v2$=triangle$[v2+6:v3-1]
  v3$=triangle$[v3+6:length]
  CALL vtx_to_coord(v1$,#2)              ! get x,y,z
  CALL vtx_to_coord(v2$,#2)
  CALL vtx_to_coord(v3$,#2)
LOOP
END SUB
```

```
!************************
! Subroutine: get_vtx
!************************
SUB get_vtx(name$,num_tri)              ! i/o,o
   ! Make *.vtx file from *.stl file
   ! name$ : STL and vertex file name without extension
   ! num_tri : number of triangles
file_stl$="object\" & name$ & ".stl"   ! file names
file_vtx$="object\" & name$ & ".vtx"
OPEN #1:name file_stl$,org byte
ASK #1:filesize fsize                  ! get number of chars in STL file
OPEN #2:name file_vtx$,access outin,create newold, org record
ERASE #2
SET #2:recsize 10
rec_pos=1                              ! start from beginning of STL file
count=0                                ! number of iteration in DO LOOP
DO WHILE more #1
   count=count+1                       ! update number of iteration
   SET #1:record rec_pos               ! read from next record
   READ #1,bytes 30000:text$           ! read 30000 chars at a time
   text$=Lcase$(text$)                 ! make all string lower case
   CALL text_to_coord(text$,rec_num,#2)  ! find vertex coords
   rec_pos=rec_pos+rec_num             ! update position of record.
LOOP
ASK #2:filesize num_rec
num_tri=num_rec/9                      ! calculate number of triangles
CLOSE #1
CLOSE #2
END SUB

!*************************
! Subroutine: object_size
!*************************
SUB object_size(name$,xmin,xmax,ymin,ymax,zmin,zmax,num_tri) ! i,o,..
   ! Find object size reading vertex coord record file
   ! name$ : vertex file name
   ! x,y,zmin,max : min and max of x, y, and z. object size.
   ! num_tri : number of triangles

file_vtx$="object\" & name$ & ".vtx"   ! file_name
CALL open_file(#1,file_vtx$,error)     ! open record file
   IF error=1 THEN exit sub
ASK #1:filesize num_rec
num_tri=num_rec/9                      ! number of triangles and vertexes.
num_vtx=num_rec/3
RESET #1:Begin
xmin,ymin,zmin=1000                    ! initialize min,max of x,y,z
xmax,ymax,zmax=-1000
FOR i=1 TO num_vtx                     ! compare all x,y,z to find min and max
   READ #1:x,y,z
   IF x<xmin THEN xmin=x               ! update min,max value
   IF x>xmax THEN xmax=x
   IF y<ymin THEN ymin=y
```

```
   IF y>ymax THEN ymax=y
   IF z<zmin THEN zmin=z
   IF z>zmax THEN zmax=z
NEXT i
CLOSE #1
END SUB
!****************************
! Subroutine: shift_vtx_file
!****************************
SUB shift_vtx_file(name$,xmin,ymin,zmin)  ! i/o,i,i,i
   ! Shift coords of vtx file to make xmin,ymin,zmin=0
   ! name$: object name, name$.vtx
   ! xmin, ymin, zmin : min and max of z, y, z. original object size.
file_vtx$="object\" & name$ & ".vtx"    ! file name
OPEN #1:Name file_vtx$,Access Outin,Create Newold,Org Record
ASK #1:filesize num_rec
num_vtx=num_rec/3                  ! number of vertexes
RESET #1:Begin
FOR i=1 TO num_vtx                 ! shift all coords
   RESET #1:Record (i-1)*3+1
   READ #1:x,y,z
   RESET #1:Record (i-1)*3+1
   WRITE #1:x-xmin,y-ymin,z-zmin
NEXT i
CLOSE #1
END SUB
!****************************
! Subroutine: stl_to_vtx
!****************************
SUB stl_to_vtx(name$,num_tri,xmax,ymax,zmax)  ! i/o,o,o,o,o
   ! Make *.vtx coord file reading and shifting *.stl file
   ! name$ : *.stl, *.vtx. object name
   ! num_tri,x,y,zmax : num of triangle in vertex file, and object size
  ! make vertex record file
CALL get_vtx(name$,num_tri)
                  ! find object size
CALL object_size(name$,xmin,xmax,ymin,ymax,zmin,zmax,num_tri)
                  ! if min=0, then skip shifting
IF xmin=0 and ymin=0 and zmin=0 THEN
  EXIT SUB
ELSE                  ! if min<>0, then shift
  CALL shift_vtx_file(name$,xmin,ymin,zmin)
  xmax=xmax-xmin
  ymax=ymax-ymin
  zmax=zmax-zmin
END IF
END SUB
!****************************
! Subroutine: scale_vtx_file
!****************************
SUB scale_vtx_file(name_in$,xr,yr,zr,xmax,ymax,zmax,name_out$)
                  ! i,i,i,i,i/o,i/o,i/o,o
   ! Enlarge object by scaling vertex coords
```

```
    ! name_in$,_out$: original vertex coord file and scaled vertex file
    ! xr,yr,zr : conversion ratio.  old_coord*ratio=new_coord
    ! x,y,zmax : new object size after scaling
  file_old$="object\" & name_in$ & ".vtx"          ! file names
  file_new$="object\" & name_out$ &".vtx"
  OPEN #1:Name file_old$,Access Outin,Create Newold,Org Record
  IF file_old$<>file_new$ THEN
    OPEN #2:Name file_new$,Access Outin,Create Newold,Org Record
    ERASE #2
    SET #2:Recsize 10
  END IF
  RESET #1:Begin
  ASK #1:filesize num_rec
  num_vtx=num_rec/3                    ! get number of vertex
  FOR i=1 TO num_vtx                   ! scale all vertex
    RESET #1:Record (i-1)*3+1
    READ #1:x,y,z
    x=x*xr                       ! scale x,y,z
    y=y*yr
    z=z*zr
    IF file_old$<>file_new$ THEN
      WRITE #2:x,y,z
    ELSE
      RESET #1:Record (i-1)*3+1
      WRITE #1:x,y,z
    END IF
  NEXT i
  xmax=xmax*xr                     ! update object size
  ymax=ymax*yr
  zmax=zmax*zr
  CLOSE #1
  IF file_old$<>file_new$ THEN
    CLOSE #2
  END IF
  END SUB
  !************************
  ! Subroutine: rotate_vtx
  !************************
  SUB rotate_vtx(name_in$,axes$,angle,xmax,ymax,zmax,name_out$)
                              ! i/o,i,i,o,o,o,o
    ! Rotate vertex file about x, y, or z axes.
    ! name_in$, _out$ : *.vtx. object name
    ! axes$ : rotation axes. "x", "y" or "z"
    ! angle : rotation angle in degree.
    ! xmax,ymax, zmax : new object size after rotation
  IF axes$<>"x" and axes$<>"y" and axes$<>"z" THEN Exit Sub
  IF angle=0 THEN Exit Sub
  file_old$="object\" & name_in$ & ".vtx"          ! file names
  file_new$="object\" & name_out$ &".vtx"
  OPEN #1:Name file_old$,Access Outin,Create Newold,Org Record
  IF file_old$<>file_new$ THEN
    OPEN #2:Name file_new$,Access Outin,Create Newold,Org Record
    ERASE #2
```

```
  SET #2:Recsize 10
END IF

ASK #1:filesize num_rec
num_vtx=num_rec/3
angle=angle*pi/180          ! convert degree into radian

FOR i=1 TO num_vtx      ! rotate vtx
  RESET #1:Record i*3-2
  READ #1:x,y,z
  IF axes$="x" THEN                ! rotate about x
    xnew=x
    ynew=y*cos(angle)-z*sin(angle)
    znew=y*sin(angle)+z*cos(angle)
  ELSE IF axes$="y" THEN           ! rotate about y
    xnew=x*cos(angle)+z*sin(angle)
    ynew=y
    znew=-x*sin(angle)+z*cos(angle)
  ELSE                             ! rotate about z
    xnew=x*cos(angle)-y*sin(angle)
    ynew=x*sin(angle)+y*cos(angle)
    znew=z
  END IF
  IF file_old$<>file_new$ THEN
    WRITE #2:xnew,ynew,znew
  ELSE
    RESET #1:Record (i-1)*3+1
    WRITE #1:xnew,ynew,znew
  END IF
NEXT i
CLOSE #1
IF file_old$<>file_new$ THEN
  CLOSE #2
END IF
                          ! new object size
CALL object_size(name_out$,xmin,xmax,ymin,ymax,zmin,zmax,num_tri)
CALL shift_vtx_file(name_out$,xmin,ymin,zmin) ! shift to make min 0

xmax=xmax-xmin              ! find new object size
ymax=ymax-ymin
zmax=zmax-zmin
END SUB

!*************
! End of File
!*************
```

```
!*********************************************************************
! Written by Cheol H. Lee
! University of Utah
!
! File : 72-2-ln1.tru
! Date : 6/96
!
! Object : slice an object (*.vtx), and make slice files (*.ln)
! Input  : *.vtx file
! Output : *.ln file
!*********************************************************************
! MEMO
! - *.ln files are in "object\[name]\" directory
!  In *.ln are line coords. Each line has 4 coords (x1,y1,x2,y2)
!*********************************************************************
library "69-ut"
library "69-gln"
name$="s1"
start=1
end=2
dh,decimal=2
!call vtx_to_line(name$,start,end,dh,decimal)
call draw_line(name$,2,0,100,100)
end
EXTERNAL
!*********
! Summary (72-2-ln1)
!*********
!   SUB get_line(name$,#1,array,a(,),num_tri,slice_h,decimal,file_ln,num_ln)
! SUB vtx_to_line_h(name$,slice_h,decimal,file_ln)
! SUB vtx_to_line(name$,start,end,dh,decimal)

!**********************
! Subroutine: get_line
!**********************

SUB get_line(name$,#1,array,a(,),num_tri,slice_h,decimal,file_ln,num_ln)

  ! Make line segment coord record file (*.ln) of a slice    i,..,o,o
  !    of the object at a certain height
  ! name$ : object name.  *.vtx
  ! #1    : *.vtx file channel
  ! array : array option.
  !       1=use array to load vtx file, 0=do not use array
  !       If memory allows, use array
  ! a(,)  : load vtx file to array.
  !       array size=a(num_tri,9). (x1,y1,z1,x2,y2,z2,x3,y3,z3)
  ! num_tri : number of triangle in vertex file
  ! slice_h : height of a slice
  ! decimal : round decimal of line coord
  ! file_ln : line coord record file name. make file_ln.ln
  !        in *.ln file (x1,y1,x2,y2)(x2,y2,x3,y3)..
  ! num_ln  : number of lines in *.ln file

file_ln$="object\" & name$ & "\" & str$(file_ln) & ".ln"     ! file name
OPEN #2:Name file_ln$,Access Outin,Create Newold,Org Record
ERASE #2
SET #2:RECSIZE 10
```

```
RESET #1:Begin

!-------------------------------------------------
! read *.vtx file and find line segment on a slice

FOR i=1 TO num_tri

  IF array=1 THEN              ! use array if there is enough memory
    x1=a(i,1)                  ! read triangle coord from array
    y1=a(i,2)
    z1=a(i,3)
    x2=a(i,4)
    y2=a(i,5)
    z2=a(i,6)
    x3=a(i,7)
    y3=a(i,8)
    z3=a(i,9)
  ELSE                         ! if we don't use array, read file directly
    READ #1:x1,y1,z1,x2,y2,z2,x3,y3,z3
  END IF

  z1=z1-slice_h              ! shift slice height to z=0
  z2=z2-slice_h
  z3=z3-slice_h

  found=0                    ! found=1 if side of triangle intersects
                             !   slice plane,
  check=1                    ! check intersection of triangle and slice plane
  IF z1>0 AND z2>0 AND z3>0 THEN check=0 ! three z points are above cut
  IF z1<0 AND z2<0 AND z3<0 THEN check=0 ! three z points are below cut
  IF z1=0 AND z2=0 AND z3=0 THEN check=0 ! three z points are on the cut
  !---------------
  ! when intersect
  IF check=1 THEN
  !-------------
  ! 1) z1=0 case
  IF z1=0 THEN                 ! at least one point is on the slice
    IF z2=0 AND z3<>0 THEN     ! z1=0 z2=0 z3<>0
      x1w=x1
      y1w=y1
      x2w=x2
      y2w=y2
      found=1
    ELSE IF z2>0 THEN
      IF z3<=0 THEN            ! z1=0 z2>0 z3<=0
        x1w=x1
        y1w=y1
        n2=Abs(z2)             ! n2:n3 ratio
        n3=Abs(z3)
        x2w=(n2*x3+n3*x2)/(n2+n3)
        y2w=(n2*y3+n3*y2)/(n2+n3)
        found=1
      END IF                   ! z1=0 z2>0 z3>0 not the case
    ELSE IF z2<0 THEN
      IF z3>=0 THEN            ! z1=0 z2<0 z3>=0
        x1w=x1
        y1w=y1
```

```
      n2=Abs(z2)          ! n2:n3 ratio
      n3=Abs(z3)
      x2w=(n2*x3+n3*x2)/(n2+n3)
      y2w=(n2*y3+n3*y2)/(n2+n3)
      found=1
    END IF                ! z1=0 z2<0 z3<0 not the case
  END IF
END IF
!-------------
! 2) z1>0 case
IF z1>0 THEN              ! at least one point is above slice plane
  IF z2=0 THEN
    IF z3<=0 THEN         ! z1>0 z2=0 z3<=0
      x1w=x2
      y1w=y2
      n1=Abs(z1)          ! n1:n3 ratio
      n3=Abs(z3)
      x2w=(n1*x3+n3*x1)/(n1+n3)
      y2w=(n1*y3+n3*y1)/(n1+n3)
      found=1
    END IF                ! z1>0 z2=0 z3>0 not the case
  ELSE IF z2>0 THEN
    IF z3<0 THEN          ! z1>0 z2>0 z3<0
      n2=Abs(z2)          ! n2:n3 ratio
      n3=Abs(z3)
      x1w=(n2*x3+n3*x2)/(n2+n3)
      y1w=(n2*y3+n3*y2)/(n2+n3)
      n1=Abs(z1)          ! n1:n3 ratio
      n3=Abs(z3)
      x2w=(n1*x3+n3*x1)/(n1+n3)
      y2w=(n1*y3+n3*y1)/(n1+n3)
      found=1
    END IF                ! else not the case
  ELSE IF z2<0 THEN
    IF z3=0 THEN          ! z1>0 z2<0 z3=0
      x1w=x3
      y1w=y3
      n1=Abs(z1)          ! n1:n2 ratio
      n2=Abs(z2)
      x2w=(n1*x2+n2*x1)/(n1+n2)
      y2w=(n1*y2+n2*y2)/(n1+n2)
      found=1
    ELSE IF z3>0 THEN     ! z1>0 z2<0 z3>0
      n1=Abs(z1)          ! n1:n2 ratio
      n2=Abs(z2)
      x1w=(n1*x2+n2*x1)/(n1+n2)
      y1w=(n1*y2+n2*y1)/(n1+n2)
      n2=Abs(z2)          ! n2:n3 ratio
      n3=Abs(z3)
      x2w=(n2*x3+n3*x2)/(n2+n3)
      y2w=(n2*y3+n3*y2)/(n2+n3)
      found=1
    ELSE IF z3<0 THEN     ! z1>0 z2<0 z3<0
      n1=Abs(z1)          ! n1:n2 ratio
      n2=Abs(z2)
      x1w=(n1*x2+n2*x1)/(n1+n2)
```

```
      y1w=(n1*y2+n2*y1)/(n1+n2)
      n1=Abs(z1)          ! n1:n3 ratio
      n3=Abs(z3)
      x2w=(n1*x3+n3*x1)/(n1+n3)
      y2w=(n1*y3+n3*y1)/(n1+n3)
      found=1
    END IF
  END IF
END IF
!-------------
! 3) z1<0 case
IF z1<0 THEN                ! at least one point is below slice plane
  IF z2=0 THEN
    IF z3>=0 THEN           ! z1<0 z2=0 z3>=0
      x1w=x2
      y1w=y2
      n1=Abs(z1)          ! n1:n3 ratio
      n3=Abs(z3)
      x2w=(n1*x3+n3*x1)/(n1+n3)
      y2w=(n1*y3+n3*y1)/(n1+n3)
      found=1
    END IF
  ELSE IF z2>0 THEN
    IF z3=0 THEN            ! z1<0 z2>0 z3=0
      x1w=x3
      y1w=y3
      n1=Abs(z1)          ! n1:n2 ratio
      n2=Abs(z2)
      x2w=(n1*x2+n2*x1)/(n1+n2)
      y2w=(n1*y2+n2*y1)/(n1+n2)
      found=1
    ELSE IF z3>0 THEN       ! z1<0 z2>0 z3>0
      n1=Abs(z1)          ! n1:n2 ratio
      n2=Abs(z2)
      x1w=(n1*x2+n2*x1)/(n1+n2)
      y1w=(n1*y2+n2*y1)/(n1+n2)
      n1=Abs(z1)          ! n1:n3 ratio
      n3=Abs(z3)
      x2w=(n1*x3+n3*x1)/(n1+n3)
      y2w=(n1*y3+n3*y1)/(n1+n3)
      found=1
    ELSE                    ! z1<0 z2>0 z3<0
      n1=Abs(z1)          ! n1:n2 ratio
      n2=Abs(z2)
      x1w=(n1*x2+n2*x1)/(n1+n2)
      y1w=(n1*y2+n2*y1)/(n1+n2)
      n2=Abs(z2)          ! n2:n3 ratio
      n3=Abs(z3)
      x2w=(n2*x3+n3*x2)/(n2+n3)
      y2w=(n2*y3+n3*y2)/(n2+n3)
      found=1
    END IF
  ELSE IF z2<0 THEN
    IF z3>0 THEN            ! z1<0 z2<0 z3>0
      n1=Abs(z1)          ! n1:n3 ratio
      n3=Abs(z3)
```

```
        x1w=(n1*x3+n3*x1)/(n1+n3)
        y1w=(n1*y3+n3*y1)/(n1+n3)
        n2=Abs(z2)          ! n2:n3 ratio
        n3=Abs(z3)
        x2w=(n2*x3+n3*x2)/(n2+n3)
        y2w=(n2*y3+n3*y2)/(n2+n3)
        found=1
      END IF
    END IF
  END IF
  END IF
  IF found=1 THEN
    x1r=Round(x1w,decimal)      ! round intersecting line coord
    y1r=Round(y1w,decimal)
    x2r=Round(x2w,decimal)
    y2r=Round(y2w,decimal)
    WRITE #2:x1r,y1r,x2r,y2r    ! save rounded line coord
  END IF
NEXT i
ASK #2:Filesize num_rec
num_ln=num_rec/4                ! number of lines in *.ln file
CLOSE #2
END SUB
!*****************************
! Subroutine: vtx_to_line_h
!*****************************
SUB vtx_to_line_h(name$,slice_h,decimal,file_ln) ! i,i,i,o
  ! Cut an object at a certain height and make line files
  ! name$ : object name, *.vtx
  ! slice_h : cut height
  ! decimal : round line coords
  ! file_ln : line record file name. *.ln
DIM a(0,0)                      ! initialize
com$="md object\" & name$       ! if no object directory, create it
CALL dos(com$)
CLEAR
file_vtx$="object\" & name$ & ".vtx"   ! file name
CALL open_file(#1,file_vtx$,error)     ! open vertex coord file
  IF error=1 THEN Exit sub
ASK #1:filesize num_rec
num_tri=num_rec/9
PRINT " Making file ":str$(file_ln);".ln.  Wait..."
                                ! make a line file
CALL get_line(name$,#1,0,a(,),num_tri,slice_h,decimal,file_ln,num_line)
CLOSE #1
END SUB
!*************************
! Subroutine: vtx_to_line
!*************************
SUB vtx_to_line(name$,start,end,dh,decimal)  ! i,i,i,i,i

  ! Automatically cut an object with dh thickness, and make line files
  ! name$ : object name. *.vtx
  ! start,end : start and end number of line files to make
  !     1.ln [h=0], start.ln [h=(start-1)*dh]
  ! dh,decimal: layer thickness. round line coord.
```

```
DIM a(0,0)                    ! initialize

com$="md object\" & name$          ! if no object directory, create it
CALL dos(com$)
CLEAR

file_vtx$="object\" & name$ & ".vtx"    ! file_name
CALL open_file(#1,file_vtx$,error)      ! open vertex file
   IF error=1 THEN Exit sub

ASK #1:filesize num_rec
num_tri=num_rec/9

array=1                       ! check memory for array usuage
WHEN error in
   MAT REDIM a(num_tri,9)
   CALL file_to_array(file_vtx$,a(,),num_tri,9)
USE
   array=0
END WHEN

!--------------------------------
! making line files for cuts

no_line_file=0              ! number of empty line files
num_file=end-start+1          ! number of line file

FOR i=start TO end            ! create line files

   slice_h=(i-1)*dh          ! slice height
   PRINT
   PRINT "Making file ";str$(i);".ln";" (";i;"/";end;"). Wait.."

                  ! get one line file at certain height
   CALL get_line(name$,#1,array,a(,),num_tri,slice_h,decimal,i,num_line)
                  ! if line file is empty, update counter.
   IF num_line<3 THEN no_line_file=no_line_file+1
NEXT i

PRINT " *** OUTPUT ***"
PRINT
PRINT num_file;"files are made. "
PRINT no_line_file;"files have less than 3 lines."

CLOSE #1
END SUB

!*************
! End of File
!*************
```

```
!************************************************************
! Written by Cheol H. Lee
! University of Utah
! File: 72-2-ln2.tru
! Date: 2/96
! Object : scale and shift line coord in line file, and find
!        min and max x,y of line coord
! Input  : *.ln file, old line file
! Output : *.ln file, new line file
!************************************************************
library "69-gln.tru"
library "3dlib.trc"
library "69-ut.tru"
name_in=105
name_out=105
scale=1
xshift=10
yshift=5
call line_size(name_in,xmin,xmax,ymin,ymax,num_line)
print name_in,xmin,xmax,ymin,ymax
call scale_shift_line(name_in,scale,xshift,yshift,xmin,xmax,ymin,ymax,name_out)
call line_size(name_out,xmin,xmax,ymin,ymax,num_line)
print name_out,xmin,xmax,ymin,ymax
end
EXTERNAL
!*********
! Summary (72-2-ln2)
!*********
! SUB line_size(name$,name,xmin,xmax,ymin,ymax,num_line)
! SUB scale_shift_line(name_in,scale,xshift,yshift,xmin,xmax,ymin,ymax,name_out)
!************************
! Subroutine: line_size
!************************
SUB line_size(name$,name,xmin,xmax,ymin,ymax,num_line)  ! i,o,i,i,i,i,i
  ! Fine min and max x,y in a line file
  ! name$ : object name. *.vtx
  ! name : line file num. name.ln
  ! x,ymin,max : min and max of x, y
  ! num_line : number of lines in *.ln
file_ln$="object\" & name$ & "\" & Str$(name) & ".ln"  ! file_name
CALL open_file(#1,file_ln$,error)    ! open line file
  IF error=1 THEN exit sub
ASK #1:filesize num_rec
num_line=num_rec/4                  ! number of lines
RESET #1:Begin
xmin,ymin=1000                ! initialize min,max of x,y,z
xmax,ymax=-1000
FOR i=1 TO num_line           ! find min and max x,y of all lines
  READ #1:x1,y1,x2,y2
  CALL min3(x1,x2,xmin,xmin)
  CALL min3(y1,y2,ymin,ymin)
  CALL max3(x1,x2,xmax,xmax)
  CALL max3(y1,y2,ymax,ymax)
NEXT i
CLOSE #1
END SUB
```

```
!*****************************
! Subroutine: scale_shift_line
!*****************************

SUB scale_shift_line(name$,name_in,scale,xshift,yshift,xmin,xmax,ymin,ymax,name_out)
                    ! i(5),o(5)
  ! Enlarge and shift line coord by scaling x,y coords
  ! name$ : object name. *.vtx
  ! name_in,_out: original and scaled line files. *.ln
  ! scale    : conversion ratio. old_coord*scale=new_coord
  ! x,yshift  : shift distance. x_new=x_old+xshift, y_new=y_old+yshift
  ! x,ymin,max : new min and max x,y of line file after scaling

file_in$="object\" & name$ & "\" & str$(name_in) & ".ln"  ! input file
file_out$="object\" & name$ & "\" & str$(name_out) & ".ln" ! output file

OPEN #1:Name file_in$,Access Outin,Create Newold,Org Record
IF file_in$<>file_out$ THEN
  OPEN #2:Name file_out$,Access Outin,Create Newold,Org Record
  ERASE #2
  SET #2:Recsize 10
END IF

ASK #1:filesize num_rec
num_line=num_rec/4          ! number of lines in *.ln
RESET #1:Begin

FOR i=1 TO num_line          ! scale and shift all lines
  RESET #1:Record (i-1)*4+1
  READ #1:x1,y1,x2,y2

  x1=x1*scale+xshift        ! scale and shift
  x2=x2*scale+xshift
  y1=y1*scale+yshift
  y2=y2*scale+yshift

  IF file_in$<>file_out$ THEN
    WRITE #2:x1,y1,x2,y2
  ELSE
    RESET #1:Record (i-1)*4+1
    WRITE #1:x1,y1,x2,y2
  END IF
NEXT i

CALL line_size(name_in,xmin,xmax,ymin,ymax,num_line) ! find min max x,y

CLOSE #1
IF file_in$<>file_out$ THEN
  CLOSE #2
END IF
END SUB

!*************
! End of File
!*************
```

```
!*************************************************************
! Written by Cheol H. Lee
! University of Utah
!
! File: 72-3-lp1.tru
! Date: 1/96
!
! Object : make complete loops, and save coord info in array
! Input  : a(num of lines,4)
! Output : b(num of lines,4), index(num of loops,2)
!*************************************************************
! MEMO
! - coord in *.ln file are checked, verified, and sorted to make
!   complete loops
! - eliminate lines which are points or duplicated lines, and fill
!   gaps or add missing lines, if any
! - a(num of lines,4) from *.ln : (x1,y1,x2,y2)(x3,y3,x4,y4),..
! - b(sorted num of lines,4) : (x1,y1,x2,y2)(x2,y2,x3,y3),..
! - index(num of loops,2) :
!       (start dim_b,end dim_b)(start dim_b, end dim_b),..
!*************************************************************
EXTERNAL

!*********
! Summary (72-3-lp1)
!*********

! SUB refine_line(a(,),dim_a1,b(,),dim_b1)
! SUB find_connected_line(a(,),dim_a1,x1,y1,x2,y2,&
!                 & x1c,y1c,x2c,y2c,found,count_line)
! SUB find_closest_line(a(,),dim_a1,x_ref,y_ref, &
!                 & x1d,y1d,x2d,y2d,d_min,count_line)
! SUB make_loop(a(,),dim_a1,d_loop,b(,),dim_b1,index(,),dim_index1)

!*************************
! Subroutine: refine_line
!*************************

SUB refine_line(a(,),dim_a1,b(,),dim_b1)  ! i,i,o,o

  ! Eliminate lines which are not lines but points in line file.
  ! a(,)  : line coord record array. a(dim_a1,4)
  ! dim_a1 : number of lines
  ! b(,)  : updated line coord array. b(dim_b1,4)
  ! dim_b1 : updated array size. new number of lines

MAT REDIM b(dim_a1,4)          ! declare array size
dim_b1=0

FOR i=1 TO dim_a1              ! check all lines for points
 IF a(i,1)<>a(i,3) Or a(i,2)<>a(i,4) THEN   ! if a line is not a point
   dim_b1=dim_b1+1
   b(dim_b1,1)=a(i,1)                 ! save line in new array
   b(dim_b1,2)=a(i,2)
   b(dim_b1,3)=a(i,3)
   b(dim_b1,4)=a(i,4)
 END IF
```

```
NEXT i

IF dim_b1=0 THEN                        ! if new line file is empty,
  PRINT "Problem, no line.  Press any key"    ! print error message
  GET KEY getkey
  EXIT SUB
ELSE
  PRINT dim_b1;"lines revised. ";dim_a1-dim_b1;"points removed."
END IF
END SUB

!*********************************
! Subroutine: find_connected_line
!*********************************
SUB find_connected_line(a(,),dim_a1,x1,y1,x2,y2,&     ! i,i,o,..,i/o
              & x1c,y1c,x2c,y2c,found,count_line)

  ! Find a line starting at x2,y2, and remove duplicated lines.
  ! a(,), dim_a1   : line array and size.  a(dim_a1,4)
  ! x1,y1,x2,y2    : a line coord ending at x2,y2
  ! x1c,y1c,x2c,y2c: a line coord starting at x2,y2
  ! found         : 0=if not found, 1=if found connected line at x2,y2
  ! count_line     : number of lines checked
found=0                     ! initialize

FOR i=1 TO dim_a1     ! find a line starting x2,y2 by searching all lines

  ! -- check lines which are not checked yet

  IF a(i,1)>=0 THEN
    x11=a(i,1)
    y11=a(i,2)
    x22=a(i,3)
    y22=a(i,4)

    !-- check connection at starting point to x2,y2

    IF x11=x2 And y11=y2 THEN
      a(i,1)=-10              ! mark when checked
      count_line=count_line+1     ! update count_line
      IF x22<>x1 Or y22<>y1 THEN   ! if not duplicated line, keep it
        x1c=x11
        y1c=y11
        x2c=x22
        y2c=y22
        found=1
      END IF
    !-- check connection at end point to x2,y2
    ELSE IF x22=x2 And y22=y2 THEN
      a(i,1)=-10              ! mark when checked
      count_line=count_line+1
      IF x11<>x1 Or y11<>y1 THEN   ! if not duplicated line, keep it
        x1c=x22
        y1c=y22
        x2c=x11
        y2c=y11
```

```
          found=1
        END IF
      END IF
    END IF
  NEXT i
  END SUB

  !**********************************
  ! Subroutine: find_closest_line
  !**********************************
  SUB find_closest_line(a(,),dim_a1,x_ref,y_ref, &   ! i(4),o(5),i/o
              & x1d,y1d,x2d,y2d,d_min,count_line)

    ! If a connected line was not found, find closest line from x_ref,y_ref
    ! a(,), dim_a1: line coord array and its size. a(dim_a1,4)
    ! x_ref,y_ref : reference point to find closest line from
    ! x1d,y1d,x2d,y2d : closest line from reference point (x_ref,y_ref)
    ! d_min      : gab between ref point and closest line
    ! count_line  : number of lines checked

 d_min=100000                  ! initialize
 line_pos=0

 !-- check all lines
 FOR i=1 TO dim_a1

   !-- check line which are not checked yet

   IF a(i,1)>=0 THEN
     x1=a(i,1)
     y1=a(i,2)
     x2=a(i,3)
     y2=a(i,4)

     d=((x_ref-x1)^2+(y_ref-y1)^2)^0.5
                  ! distance between x1,y1 and reference point
     IF d<d_min THEN        ! update distance
       x1d=x1
       y1d=y1
       x2d=x2
       y2d=y2
       d_min=d
       line_pos=i
     END IF

     d=((x_ref-x2)^2+(y_ref-y2)^2)^0.5
                  ! distance between x2,y2 and reference point
     IF d<d_min THEN        ! update distance
       x1d=x2
       y1d=y2
       x2d=x1
       y2d=y1
       d_min=d
       line_pos=i
     END IF
   END IF
 NEXT i
```

```
IF line_pos=0 THEN          ! if the closest line is not found, error
  PRINT "Problem!!! Can not find closest line."
ELSE
  a(line_pos,1)=-10         ! mark if checked
  count_line=count_line+1    ! update counter
END IF
END SUB

!************************
! Subroutine: make_loop
!************************
SUB make_loop(a(,),dim_a1,d_loop,b(,),dim_b1,index(,),dim_index1)!i(3),o(4)

   ! Make closed loops by connecting all lines and adding missing lines
   ! Call find_closest_line, find_connected_line
   ! a(,), dim_a1: line coord array and its size. a(dim_a1,4)
   !           (x1,y1,x2,y2)(x3,y3,x4,y4)(x5,y5,x6,y6),..
   ! d_loop     : max gab allowed for missing line
   !           if gab>d_loop, new loop exists
   ! b(,),dim_b1 : loop line coord array and its size.  b(dim_b1,4)
   !           (x1,y1,x2,y2)(x2,y2,x3,y3)(x3,y3,x4,y4),..
   ! index(,) : loop line file index and its size.  index(dim_index1,2)
   !         (start dim_b,end dim_b)(start dim_b,end dim_b),..
   ! dim_index1 : number of loops
MAT REDIM b(2*dim_a1,4)        ! declare array size
count_line=0                ! initialize count lines considered
dim_b1=0                    ! initialize number of connected line
count_closest_line=0
!-- first line of first loop
 dim_index1=1                  ! number of loops
 index(dim_index1,1)=1            ! first loop start index
 x1=a(1,1)                   ! start line
 y1=a(1,2)
 x2=a(1,3)
 y2=a(1,4)
 dim_b1=dim_b1+1                ! keep the first line
 b(dim_b1,1)=x1
 b(dim_b1,2)=y1
 b(dim_b1,3)=x2
 b(dim_b1,4)=y2
 a(1,1)=-10                  ! mark if considered
 count_line=count_line+1

 x_start=x1                  ! start a loop from x1,y1
 y_start=y1

!-- make closed loops
DO
  found=0
                          ! find connected line from x1,y1
  CALL find_connected_line(a(,),dim_a1,x1,y1,x2,y2,&
              & x1c,y1c.x2c,y2c,found,count_line)
  !-- if found connected line, keep the line
  IF found=1 THEN
    dim_b1=dim_b1+1
    b(dim_b1,1)=x1c
    b(dim_b1,2)=y1c
```

```
    b(dim_b1,3)=x2c
    b(dim_b1,4)=y2c
  END IF

  !-- if not found connected line, find closest line
  IF found=0 THEN


   CALL find_closest_line(a(,),dim_a1,x2,y2, &
            & x1c,y1c,x2c,y2c,d_min,count_line)
   count_closest_line=count_closest_line+1


   !-- add missing line by filling gab
   IF d_min<d_loop THEN

     IF x2=x_start And y2=y_start THEN   ! check d_loop
       PRINT "Problem!!!  Minimum loop distance is too small."
       GET KEY key
     END IF
     dim_b1=dim_b1-1        ! add missing line
     b(dim_b1,1)=x2
     b(dim_b1,2)=y2
     b(dim_b1,3)=x1c
     b(dim_b1,4)=y1c
     dim_b1=dim_b1-1
     b(dim_b1,1)=x1c
     b(dim_b1,2)=y1c
     b(dim_b1,3)=x2c
     b(dim_b1,4)=y2c

   !-- new loop
   ELSE

     !-- close loop
     IF x_start<>x2 OR y_start<>y2 THEN
       dim_b1=dim_b1+1
       b(dim_b1,1)=x2
       b(dim_b1,2)=y2
       b(dim_b1,3)=x_start
       b(dim_b1,4)=y_start
     END IF

     index(dim_index1,2)=dim_b1        ! end loop index

     dim_b1=dim_b1-1                ! new start line
     b(dim_b1,1)=x1c
     b(dim_b1,2)=y1c
     b(dim_b1,3)=x2c
     b(dim_b1,4)=y2c

     dim_index1=dim_index1+1
     index(dim_index1,1)=dim_b1        ! start loop index
     x_start=x1c
     y_start=y1c
   END IF
  END IF
```

```
  x1=x1c                    ! new line to be connected
  y1=y1c
  x2=x2c
  y2=y2c
LOOP UNTIL count_line>=dim_a1

!-- close last loop
IF x_start<>x2 OR y_start<>y2 THEN
  dim_b1=dim_b1+1
  b(dim_b1,1)=x2
  b(dim_b1,2)=y2
  b(dim_b1,3)=x_start
  b(dim_b1,4)=y_start
END IF
index(dim_index1,2)=dim_b1      ! end loop index

PRINT dim_index1;"loops, ";dim_b1;"lines, ";count_closest_line;"gabs exist."
END SUB

!*************
! End of File
!*************
```

```
!*************************************************************************
! Written by Cheol H.Lee
!
! File: 72-4-pt.tru
! Date: 1/96
!
! Object : Read loop file (*.lp, *.lpx), and make point file (*.pt, *.ptx)
! Input  : loop record file (*.lp, *.lpx)
! Output : point record file (*.pt, *.ptx)
!*************************************************************************
! MEMO
! - *.lp : ccw loop record file. (x1,y1,x2,y2)(x2,y2,x3,y3),..
! - *.lpx: loop index record file.  each loop has 10 records
!       (start line num, end line num, xd,yd, xu,yu, xl,yl, xr,yr)
!       where xd,yd=ymin pt. xu,yu=ymax pt. xl,yl=xmin pt, xr,yr=xmax pt
! - *.pt  : point coord record file (x1,y1)(x2,y2)(x3,y3),..
! - *.ptx : point index record file. each loop has 10 records
!         (num of start pt, num of end pt, xd,yd,xu,yu,xl,yl,xr,yr)
! - two methods are available in making point file
!       0=include all start and end points of lines
!       1=even distance points, ignore start and end points of lines
!*************************************************************************
library "71-3-lp1"
library "71-gln"
library "71-ut"
library "utillib1"
name$="fordcar"
start=1
end=2
step=.2
decimal=2
method=1
call read_record_file("object\fordcar\6.lp",4)
!call loop_to_point(name$,start,end,step,decimal,method)
!call draw_point(name$,1,0,100,100)
end
EXTERNAL

!*********
! Summary (72-4-pt)
!*********
!    SUB find_point_by_vtx(x1,y1,x2,y2,step,decimal,#1,count)
!    SUB find_point_by_dis(x_end,y_end,x2,y2,step,decimal,#1,count)
! SUB one_loop_to_point(name$,file_name$,step,decimal,num_point,method)
! SUB loop_to_point(name$,start,end,step,decimal,method)
! SUB scale_point(name$,in_num,x_scale,y_scale,decimal,out_num)
!*******************************
! Subroutine: find_point_by_vtx
!*******************************
SUB find_point_by_vtx(x1,y1,x2,y2,step,decimal,#1,count) ! i(6),i/o,i/o
  ! Find discrete point coords including start and end points of a line
  ! x1,y1,x2,y2 : line coord to discretize
  ! step, decimal : distance bw points.  round decimal point
  ! #1 : point file channel (*.pt)
  ! count : number of total points saved in point file
dx=x2-x1
dy=y2-y1
```

```
d=(dx^2+dy^2)^0.5      ! length of a line
num_pt=int(d/step)     ! number of points on a line

IF num_pt<1 THEN       ! if length of a line is shorter then one step,
                !      save end point
  WRITE #1:x2,y2
  count=count+1
ELSE                   ! if length of a line is longer then one step,
                !      discretize
  FOR i=1 TO num_pt
    ratio=i*step/d
    x=x1+ratio*dx
    y=y1+ratio*dy
    x=round(x,decimal)
    y=round(y,decimal)
    WRITE #1:x,y
    count=count+1
  NEXT i
  IF x<>x2 or y<>y2 THEN    ! check if end point is included
    WRITE #1:x2,y2
    count=count+1
  END IF
END IF
END SUB

!*******************************
! Subroutine: find_point_by_dis
!*******************************
SUB find_point_by_dis(x_end,y_end,x2,y2,step,decimal,#1,count)
                              ! i/o,i/o,i(4),i/o,i/o
   ! Find point coords with even distance on a line and make point file
   ! x_end,y_end : last point in a pt file. previous point
   ! x2,y2  : end point of a line
   ! step, decimal : distance bw points.  round decimal point
   ! #1 : point file channel (*.pt)
   ! count : number of total points saved in point file
dx=x2-x_end
dy=y2-y_end
d=(dx^2+dy^2)^0.5
num_pt=int(d/step)     ! number of point in a file

IF num_pt<1 THEN       ! if length of a line is shorter then one step, skip
  Exit sub

ELSE                   ! if length of a line is longer then one step,
                !      discretize a line with even step size
  FOR i=1 TO num_pt
    ratio=i*step/d
    x=x_end+ratio*dx
    y=y_end+ratio*dy
    x=round(x,decimal)
    y=round(y,decimal)
    WRITE #1:x,y
    count=count+1
  NEXT i
```

```
  x_end=x                    ! keep end point coordinates
  y_end=y
END IF
END SUB
!***********************************
! Subroutine: one_loop_to_point
!***********************************
SUB one_loop_to_point(name$,file_num,step,decimal,num_point,method)
                                  ! i,i/o,i,i,o,i
  ! Read a loop file, and make a point file
  ! name$ : object name
  ! file_num : file number for *.lp, *.lpx, *.pt, *.ptx files
  ! *.pt  : point coord record file (x1,y1)(x2,y2)(x3,y3),..
  ! *.ptx : point index record file. each loop has 10 records
  !       (num of start pt, num of end pt, xd,yd,xu,yu,xl,yl,xr,yr)
  ! step, decimal : distance bw points.  round decimal point
  ! num_point  : num of points in *.pt
  ! method : 0=include all start and end points of lines
  !        1=even distance points, ignore start and end points of lines
file_loop$="object\" & name$ & "\" & str$(file_num) & ".lp"      ! file names
file_point$="object\" & name$ & "\" & str$(file_num) & ".pt"
file_index_lp$="object\" & name$ & "\" & str$(file_num) & ".lpx"
file_index_pt$="object\" & name$ & "\" & str$(file_num) & ".ptx"

CALL open_file(#1,file_loop$,error)
IF error=1 THEN Exit sub

CALL open_file(#2,file_index_lp$,error)
IF error=1 THEN Exit sub

OPEN #3:name file_point$,access outin,create newold,organization record
OPEN #4:name file_index_pt$,access outin,create newold,organization record
ERASE #3
ERASE #4
SET #3:RECSIZE 10          ! record size of point file
SET #4:RECSIZE 10          ! record size of loop index file
RESET #1:Begin
RESET #2:Begin
ASK #2:FILESIZE num_record   ! get number of records of loop index file
IF num_record=0 THEN         ! if loop index file is blank, exit
  EXIT SUB
END IF

LET num_loop=num_record/10   ! number of loops in loop file
count_pt=0

!-----------------------------
! get point coord for all loops

FOR loop=1 TO num_loop

  READ #2:ln_start,ln_end,xd,yd,xu,yu,xl,yl,xr,yr ! get loop index
  RESET #1:Record (ln_start-1)*4+1
  READ #1:xstart,ystart,x2,y2      ! start line
  WRITE #3:xstart,ystart           ! start point
  count_pt=count_pt+1
  WRITE #4:count_pt                ! start point number of a loop
```

```
                    ! discretize first line
  IF method=0 THEN              ! passing all vertex method
    CALL find_point_by_vtx(xstart,ystart,x2,y2,step,decimal,#3,count_pt)
  ELSE                          ! even point distance method
    x_end=xstart
    y_end=ystart
    CALL find_point_by_dis(x_end,y_end,x2,y2,step,decimal,#3,count_pt)
  END IF

  !---------------------------------------
  ! compute points for all lines of a loop

  FOR i=ln_start+1 TO ln_end

    READ #1:x1,y1,x2,y2      ! get line coordinate

    IF method=0 THEN         ! passing all vertex method
      CALL find_point_by_vtx(x1,y1,x2,y2,step,decimal,#3,count_pt)
    ELSE                     ! even point distance method
      CALL find_point_by_dis(x_end,y_end,x2,y2,step,decimal,#3,count_pt)
    END IF
  NEXT i

  WRITE #3:xstart,ystart     ! return start point
  count_pt=count_pt+1

  WRITE #4:count_pt,xd,yd,xu,yu,xl,yl,xr,yr ! save in pt index file
NEXT loop

PRINT count_pt;"points in ";str$(file_num);".pt";", ";num_loop;"loops in ";str$(file_num);".ptx"
num_point=count_pt

CLOSE #1
CLOSE #2
CLOSE #3
CLOSE #4
END SUB

!*****************************
! Subroutine: loop_to_point
!*****************************

SUB loop_to_point(name$,start,end,step,decimal,method) ! i,i/o,i/o,i(3)

  ! Read loop file, and make point file
  ! start, end: start and end number of file for loop, point, index file
  ! step, decimal : distance bw points.  round decimal point
  ! method : 0=include all start and end points of lines
  !          1=even distance points, ignore start and end points of lines

FOR i=start TO end             ! create all point files

  PRINT
  PRINT "Making file ";str$(i);".pt";" (";i;"/";end;").  Wait.."

  CALL one_loop_to_point(name$,i,step,decimal,num_point,method)
NEXT i
```

```
PRINT end-start+1;"files are made. "
END SUB

!*************************
! Subroutine: scale_point
!*************************

SUB scale_point(name$,in_num,x_scale,y_scale,decimal,out_num) ! i(5),o

   ! Scale coords in point files
   ! in_num : input point file num to scale *.pt
   ! x_scale,y_scale: scale factor. new_x=old_x*x_scale,new_y=old_y*y_scale
   ! decimal  : round decimal
   ! out_num : output point file number *.pt

file_in$="object\" & name$ & "\" & str$(in_num) & ".pt"
file_out$="object\" & name$ & "\" & str$(out_num) & ".pt"

CALL open_file(#1,file_in$,error)
   IF error=1 THEN Exit sub

OPEN #5:name file_out$,access outin,create newold,organization record
ERASE #5
SET #5:RECSIZE 10          ! record size of output file

ASK #1:FILESIZE num_rec      ! number of records of loop index file
num_pt=num_rec/2

FOR i=1 TO num_pt          ! scale all point
   READ #1:x,y
   xnew=round(x*x_scale,decimal)
   ynew=round(y*y_scale,decimal)
   WRITE #5:xnew,ynew
NEXT i
END SUB

!*************
! End of File
!*************
```

```
! Written by Cheol Lee
!
! File: 72-5-ct1.tru
! Date: 6/96
!
! Object : match bot and top loops reading point file
! Input  : *.pt, *.ptx
! Output : match loop array match(,)
!*****************************************************************
! MEMO
!
! deal with arbiturary num of loops
! ignore second layer of children loops
! a(,) : a(num_lp,12) start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child
! index : 100 < 110 120 130 .. ;  200 < 210 220 230.. , 000
!      mother  children    mother  children     empty
! match(,) : (num_lp*2,12)
!    0 index for blank file   ex) match(1,0) for top blank
!    match bot loop (a1) and top loop (a2) and store match info
!    match bot index and top index in ascending order
!    match 1st lp(bot)-2nd lp(top), 3rd lo(bot)-4th lp(top), etc
!    (start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child)
!*****************************************************************
library "utillib.tru"
name$="sink"
file_num=17
clear

dim match(0,0)
call ptx_to_match(name$,file_num,match(,))  ! ,o
mat print match
end
EXTERNAL
!*********
! Summary (72-5-ct1)
!*********
!    SUB find_mother_loop(a(,),index,done,i_mom)  ! i/o,i,o,o
!    SUB find_child_loop(a(,),index,i_mom)  ! i/o,i,o
! SUB relate_loop(name$,file_num,a(,),num_mother)    ! i,i,o
!    SUB make_b(a(,),index,b(,))      ! i,i,o
!    SUB sort_b(b(,))      ! i/o
!    SUB screen_b(b(,),num_2nd)      ! i/o,o,o
!    SUB update_a(b(,).num_2nd,index,a(,))      ! i,i,i,o
!    SUB sort_a(a(,),num_mom)      ! i/o,i
! SUB screen_child(a(,),num_mother)      ! i/o,i
! SUB match_2_loop(a1(,),a2(,),num_mom1,num_mom2,match(,))  ! i,i,i,i,o
! SUB ptx_to_match(name$,file_num,match(,))  ! ,o

!*****************************
! Subroutine: find_mother_loop
!*****************************

SUB find_mother_loop(a(,),index,done,i_mom)  ! i/o,i,o,o

   ! Find main mother loops and give index number.
   ! mother loop has the smallest y among loops
   ! a(,) : pt index loop array.  a(num_lp,12)
```

```
    !       (start pt,end pt,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child)
    ! index : mother index number, ex) 100, 200, 300..
    ! done  : 1=no more loop not considered, finish relate loop process
    !         0=more loops to be considered.
    ! i_mom : index number of mother loop in array a(,)
  num_loop=ubound(a,1)
  min_yd=1000                ! init min yd
  i_mom=0                    ! init index number i
  FOR i=1 TO num_loop        ! find lowest (smallest yd) loop not considered
    IF a(i,11)=0 THEN        ! if not considered
      IF a(i,4)<min_yd THEN  ! if found lower loop, check yd
        min_yd=a(i,4)        ! update min_yd and min_i
        i_mom=i
      END IF
    END IF
  NEXT i

  IF i_mom=0 THEN            ! if not found, done=1
    done=1
  ELSE
    a(i_mom,11)=index        ! if found, mark index
  END IF
  END SUB

  !*******************************
  ! Subroutine: find_child_loop
  !*******************************
  SUB find_child_loop(a(,),index,i_mom)     ! i/o,i,i

    ! find child loop and give index number
    ! child loops have larger yd,xl and smaller yu,xr than mother
    ! a(,) : pt index loop info array. a(num_lp,12)
    !       (start pt,end pt,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child)
    ! index : mother index number, ex) 100, 200, 300..
    !         child index num will be 110,120,130,..
    !         100 < 110 120 130 .. ;  200 < 210 220 230.. , 000
    !         mother children    mother children    empty
    ! i_mom : index number of mother in array a(,)
  num_loop=ubound(a,1)
  xdm=a(i_mom,3)             ! mother boundary
  ydm=a(i_mom,4)
  xum=a(i_mom,5)
  yum=a(i_mom,6)
  xlm=a(i_mom,7)
  ylm=a(i_mom,8)
  xrm=a(i_mom,9)
  yrm=a(i_mom,10)
  count=0                    ! child count
  FOR i=1 TO num_loop        ! find lowest (smallest yd) loop not considered
    IF a(i,11)=0 THEN        ! if not considered
      IF a(i,4)>ydm and a(i,6)<yum and a(i,7)>xlm and a(i,9)<xrm THEN
        index=index+10       ! update child index
        count=count+1
        a(i,11)=index
      END IF
    END IF
  NEXT i
```

```
a(i_mom,12)=count          ! save num of child at mom
END SUB

!**************************
! Subroutine: relate_loop
!**************************

SUB relate_loop(name$,file_num,a(,),num_mother)      ! i,i,o,o

  ! Find loop relationship reading index file. if blank file a(0,12)
  ! mother loop has smallest yd
  ! children loops have larger yd,xl and smaller yu,xr than mother
  ! name$ : object name
  ! file_num : point file and index file number. *.pt, *.ptx
  ! a(,) : a(num_lp,12) start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child
  ! index : 100 < 110 120 130 .. ; 200 < 210 220 230.. , 000
  !      mother children    mother children    empty
  ! num_mother : num of mother loops

file_ptx$="object\" & name$ & "\" & str$(file_num)&".ptx"  ! index file name

CALL open_file(#1,file_ptx$,error)       ! save loop index in array
   IF error=1 THEN exit sub
ASK #1:filesize num_rec
num_loop=num_rec/10              ! num of loop
MAT REDIM a(num_loop,12)           ! prepare array
IF num_loop=0 THEN exit sub        ! if blank file a(0,12)
FOR i=1 TO num_loop              ! store loop info
  FOR j=1 TO 10
    READ #1:x
    a(i,j)=x
  NEXT j
  a(i,11)=0                  ! init
  a(i,12)=0                  ! init
NEXT i
index=0                  ! init mother index
count=0
DO                       ! check all loops, find mother and children
  index=index+100            ! update mother index
  CALL find_mother_loop(a(,),index,done,i_mom)
  IF done=1 THEN exit do       ! if no more mom then exit
  count=count+1
  index_child=index
  CALL find_child_loop(a(,),index_child,i_mom)
LOOP
num_mother=count
!print num_mother
!mat print a
CLOSE #1
END SUB
!********************
! Subroutine: make_b
!********************

SUB make_b(a(,),index,b(,))       ! i,i,o
```

```
    ! get children loop array b(,) of a mother loop
    ! a(,) : a(num_lp,12) start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child
    ! index : 100 < 110 120 130 .. ; 200 < 210 220 230.. , 000
    !      mother children    mother children    empty
    ! b(,)  : children loop array. b(num of children,10)
    !       (xd,yd,xu,yu,xl,yl,xr,yr,index of a(,),flag)
    ! flag : sorted order of a child loop
  num_loop=ubound(a,1)
  FOR j=1 TO num_loop              ! find num_child
    IF a(j,11)=index THEN num_child=a(j,12)
  NEXT j
  MAT REDIM b(num_child,10)        ! b(index of a(),yd,sort num)
  FOR j=1 TO num_child             ! make b()
    index_child=index+10*j
    FOR k=1 TO num_loop
      IF a(k,11)=index_child THEN
        b(j,1)=a(k,3)
        b(j,2)=a(k,4)
        b(j,3)=a(k,5)
        b(j,4)=a(k,6)
        b(j,5)=a(k,7)
        b(j,6)=a(k,8)
        b(j,7)=a(k,9)
        b(j,8)=a(k,10)
        b(j,9)=k
        b(j,10)=0          ! init
      END IF
    NEXT k
  NEXT j
  END SUB
  !*********************
  ! Subroutine: sort_b
  !*********************
  SUB sort_b(b(,))      ! i/o
    ! sort children loops in b(,) from smallest yd to largest
    ! b(,)  : children loop array. b(num of children,10)
    !       (xd,yd,xu,yu,xl,yl,xr,yr,index of a(,),flag)
    ! flag : sorted order of a child loop
  num_lp=ubound(b,1)
  FOR i=1 TO num_lp        ! sort all loop
    ymin=10000          ! init
    index=0
    FOR j=i TO num_lp     ! find min yd
      IF b(j,2)<=ymin THEN
        ymin=b(j,2)
        index=j
      END IF
    NEXT j
    FOR j=1 TO 10         ! switch to sort
      temp=b(i,j)
      b(i,j)=b(index,j)
      b(index,j)=temp
      b(i,10)=i         ! order
    NEXT j
  NEXT i
  END SUB
```

```
!**********************
! Subroutine: screen_b
!**********************

SUB screen_b(b(,),num_2nd)      ! i/o,o

  ! remove 2nd child loop
  ! b(,)  : children loop array. b(num of children,10)
  !       (xd,yd,xu,yu,xl,yl,xr,yr,index of a(,),flag)
  ! flag : sorted order of a child loop. -1 if 2nd child loop
  ! num_2nd : num of 2nd children loops
num_lp=ubound(b,1)
num_2nd=0
FOR i=2 TO num_lp       ! compare loops and find 2nd child
  xd=b(i,1)
  yd=b(i,2)
  xu=b(i,3)
  yu=b(i,4)
  xl=b(i,5)
  yl=b(i,6)
  xr=b(i,7)
  yr=b(i,8)
  FOR j=1 TO i-1        ! compare loops with smaller yd
    IF b(j,10)<>-1 and b(j,2)<yd and b(j,4)>yu and b(j,5)<xl and b(j,7)>xr THEN
      b(i,10)=-1
      num_2nd=num_2nd+1
      EXIT FOR
    END IF
  NEXT j
NEXT i
END SUB
!**********************
! Subroutine: update_a
!**********************
SUB update_a(b(,),num_2nd,index,a(,))      ! i,i,i,o
  ! sort index of child in ascendind order of yd, and update a(,).
  ! make index of 2nd children 0.  update num_child=num_child-num_2nd
  ! a(,) : a(num_lp,12) start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child
  ! index : 100 < 110 120 130 .. ;  200 < 210 220 230.. , 000
  !      mother  children    mother  children    empty
  ! b(,)  : children loop array. b(num of children,10)
  !       (xd,yd,xu,yu,xl,yl,xr,yr,index of a(,),flag)
  ! flag : sorted order of a child loop.  -1 if 2nd child
num_a=ubound(a,1)
num_b=ubound(b,1)
index_child=index       ! init
FOR i=1 TO num_b        ! update a(,) with b(,)
  flag=b(i,10)
  index_a=b(i,9)
  IF flag=-1 THEN
    a(index_a,11)=0
  ELSE
    index_child=index+10
    a(index_a,11)=index_child
  END IF
NEXT i
```

```
 IF num_2nd>0 THEN          ! update num_child
   FOR i=1 TO num_a
     IF a(i,11)=index THEN
       num_child=a(i,12)
       a(i,12)=num_child-num_2nd
     END IF
   NEXT i
 END IF
 END SUB
 !********************
 ! Subroutine: sort_a
 !********************
 SUB sort_a(a(,),num_mom)       ! i/o,i
   ! sort a(,) mom first and child second in ascending order of yd
   ! ex) 100,200,300,110,120,210,220,310..
   ! a(,) : a(num_lp,12) start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child
   ! num_mom : num of mother loops in a(,)
 DIM c(0,0)
 num_lp=ubound(a,1)
 MAT REDIM c(num_lp,12)
 MAT c=a                ! temp work space
 count=num_mom          ! init, num of mom and 1st child
 FOR i=1 TO num_mom       ! find num of mom and 1st loop
   index=i*100
   FOR j=1 TO num_lp            ! find num_child
     IF c(j,11)=index THEN
       num_child=c(j,12)
       count=count+num_child
     END IF
   NEXT j
 NEXT i
 MAT REDIM a(count,12)
 FOR i=1 TO num_mom       ! save mom info first
   index=i*100
   FOR j=1 TO num_lp            ! find num_child
     IF c(j,11)=index THEN
       FOR k=1 TO 12
         a(i,k)=c(j,k)
       NEXT k
     END IF
   NEXT j
 NEXT i
 count_a=num_mom
 FOR i=1 TO num_mom       ! save child info next
   index=i*100
   num_child=a(i,12)
   FOR j=1 TO num_child
     index_child=index+j*10
     FOR k=1 TO num_lp
       IF c(k,11)=index_child THEN
         count_a=count_a+1
         FOR n=1 TO 12
           a(count_a,n)=c(k,n)
         NEXT n
       END IF
     NEXT k
   NEXT j
```

```
NEXT i
END SUB

!*************************
! Subroutine: screen_child
!*************************

SUB screen_child(a(,),num_mother)       ! i/o,i

   ! sort children index with from smallest yd to largest yd
   ! remove 2nd child making index 0 and update num_child
   ! a(,) : a(num_lp,12) start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child
   ! num_mom : num of mother loops in a(,)
DIM b(0,0)
num_loop=ubound(a,1)
FOR i=1 TO num_mother                ! check all children of all mothers
   index=num_mother*100
   FOR j=1 TO num_loop               ! find num_child
      IF a(j,11)=index THEN num_child=a(j,12)
   NEXT j

   MAT REDIM b(num_child,10)         ! b(index of a(),yd,sort num)
   CALL make_b(a(,),index,b(,))      ! i,i,o
   CALL sort_b(b(,))       ! i/o
   CALL screen_b(b(,),num_2nd)       ! i/o,o
   CALL update_a(b(,),num_2nd,index,a(,))       ! i,i,i,o
NEXT i
!print num_mother
!mat print a
CALL sort_a(a(,),num_mother)
END SUB

!*************************
! Subroutine: match_2_loop
!*************************

SUB match_2_loop(a1(,),a2(,),num_mom1,num_mom2,match(,))   ! i,i,i,i,o

   ! match bot and top loops.
   ! a1(,),a2(,) : bot and top loop array. a1(num_lp,12)
   !      (start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child)
   ! num_mom1,num_mom2 : num of mother loops in a1(,),a2(,)
   ! match(,) : (num_lp*2,12)
   !    0 index for blank file   ex) match(1,0) for top blank
   !    match bot loop (a1) and top loop (a2) and store match info
   !    match bot index and top index in ascending order
   !    match 1st lp(bot)-2nd lp(top), 3rd lo(bot)-4th lp(top), etc
   !    (start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child)

DIM temp(0,0)

num_a1=ubound(a1,1)
num_a2=ubound(a2,1)
                    ! one of top and bot is blank
IF num_a1=0 OR num_a2=0 THEN              ! blank file exists
   PRINT "Problem. blank file exist."
```

```
    GET KEY key
    Exit sub
                        ! one of top and bot has 1 loop
  ELSE IF num_a1=1 OR num_a2=1 THEN ! 1 loop in top or bot
    MAT REDIM match(1,2)
    match(1,1)=1
    match(1,2)=1
                        ! both have at least 2 loops
  ELSE IF num_mom1=1 AND num_mom2=1 THEN ! 1 mom 1 child
    MAT REDIM match(2,2)
    match(1,1)=1
    match(1,2)=1
    match(2,1)=2
    match(2,2)=2
  ELSE IF num_mom1=1 AND num_mom2=2 THEN ! b=1 mom 1 child, t=2 mom
    MAT REDIM match(1,2)
    match(1,1)=1
    match(1,2)=1
  ELSE IF num_mom1=2 AND num_mom2=1 THEN ! b=2 mom, t=1 mom 1 child
    MAT REDIM match(1,2)
    match(1,1)=1
    match(1,2)=1
  ELSE IF num_mom1=2 AND num_mom2=2 THEN ! b=2 mom, t=2 mom
    MAT REDIM match(2,2)
    match(1,1)=1
    match(1,2)=1
    match(2,1)=2
    match(2,2)=2
  END IF
  n=ubound(match,1)
  MAT REDIM temp(n,2)
  MAT temp=match
  MAT REDIM match(2*n,12)
  count=0
  FOR i=n TO 1 STEP -1          ! make match(2*n,12)
    bot=temp(i,1)
    top=temp(i,2)
    count=count+1
    FOR j=1 TO 12
      match(count,j)=a1(bot,j)
    NEXT j
    count=count+1
    FOR j=1 TO 12
      match(count,j)=a2(top,j)
    NEXT j
  NEXT i
  END SUB

  !*************************
  ! Subroutine: ptx_to_match
  !*************************

  SUB ptx_to_match(name$,file_num,match(,))  ! i,i,o

    ! match bot and top loop reading ptx file
    ! name$ : object name
```

```
! file_num : point file num. (file_num).pt for bot (file_num+1).pt for top
! match(,) : (num_lp*2,12)
!    0 index for blank file   ex) match(1,0) for top blank
!    match bot loop (a1) and top loop (a2) and store match info
!    match bot index and top index in ascending order
!    match 1st lp(bot)-2nd lp(top), 3rd lo(bot)-4th lp(top), etc
!    (start,end,xd,yd,xu,yu,xl,yl,xr,yl,index,num_child)

DIM bot(0,0),top(0,0)

CALL relate_loop(name$,file_num,bot(,),num_mom1)     ! i,i,o
CALL relate_loop(name$,file_num+1,top(,),num_mom2)     ! i,i,o

CALL screen_child(bot(,),num_mom1)       ! i/o,i
CALL screen_child(top(,),num_mom2)       ! i/o,i

CALL match_2_loop(bot(,),top(,),num_mom1,num_mom2,match(,))   ! i,i,i,i,o
END SUB

!**********************
! End of File
!**********************
```

```
!*********************************************************************
! Written by Cheol Lee
! File: 72-5-ct2.tru
! Date: 6/96
! Object : Make cut file reading point file
! Input  : *.pt, *.ptx
! Output : *.ct, *.ctx
!*********************************************************************
! MENO
! - *.ctx : 13 rec for each match loop, and 5 rec at the end
!   (start num of cut,end num of cut,xbstart,ybstart,xtstart,ytstart,
!    xmin,xmax,ymin,ymax,index_b,index_t,max_angle)
!       ... (xmin,xmax,ymin,ymax,max_angle for all match loops)
! - can keep top cross section by reverse_cut_file for better looking slice
!*********************************************************************
library "71-gct.tru"
library "71-ut.tru"
library "71-5-ct1.tru"
library "utillib1.tru"
library "3dlib.trc"
d_point=.4
cut_angle=40
decimal=4
dh=1.3

name$="fordcar1"
skip=0
reverse=0
start,end=7

clear
!call cut_report(name$,19,1,40,2,2,2,2,12,12,12,1,1,2,19)
!call read_cut_report(name$)
call auto_cut(name$,start,end,reverse,dh,cut_angle,decimal,skip)
end

EXTERNAL

!*********
! Summary (72-5-ct2)
!*********

!   SUB point_to_array(#1,start,end,a(,)) ! .,o
!   SUB find_closest_pt(a(,),xi,yi,index,xo,yo) ! .,o,o,o
!   SUB resort_array(a(,),index) ! i/o,i
!   SUB cutter_position(d_allowed,decimal,x1,y1,x2,y2,x2_cut,y2_cut) !.,o,o
! SUB cut_a_loop(#1,#2,skip,start_b,end_b,start_t,end_t,&
!         & x1s,y1s,x2s,y2s,dh,cut_angle,decimal,#5,#6)  ! .,,o
!   SUB reverse_cut_file(#1,#2)      ! i,i
! SUB point_to_cut(name$,file_num,reverse,dh,cut_angle,decimal,skip,&
!         & xsize,ysize,max_angle) ! i/o,i.,o,o,o
! SUB auto_cut(name$.start,end,reverse,dh,cut_angle,decimal,skip)

! SUB cut_a_loop_sm1(#1,start_b,end_b,index_b,decimal,#5,#6)
! SUB point_to_cut_sm1(name$,file_num,decimal)
! SUB auto_cut_sm1(name$,start,end,decimal)
```

```
!*****************************
! Subroutine: point_to_array
!*****************************

SUB point_to_array(#1,start,end,a(,)) ! i,i,i,o

   ! copy a loop pt coord in array.
   ! start,end : start and end pt num for a loop
   !          first pt and last pt are same bc it is a loop
   ! a(,) : point array for a loop, a(num_pt,2{x,y})

num_pt=end-start+1
MAT REDIM a(num_pt,2)

rec_start=start*2-1
RESET #1:record rec_start
FOR i=1 TO num_pt
   READ #1:x,y
   a(i,1)=x        ! shift
   a(i,2)=y
NEXT i
END SUB
!*****************************
! Subroutine: find_closest_pt
!*****************************
SUB find_closest_pt(a(,),xi,yi,index,xo,yo) ! i(3),o(3)
   ! find closest pt in a(,) from (xi,yi)
   ! a(,) : point array for a loop, a(num_pt,2{x,y})
   ! xi,yi : a point to calculate distance from
   ! xo,yo : closest point in a(,) from xi,yi
   ! index : index in a(,) for (xo,yo)
num_pt=ubound(a,1)
d_min=1000000            ! init
index=0

FOR i=1 TO num_pt
   x=a(i,1)
   y=a(i,2)
   d=(xi-x)^2+(yi-y)^2
   IF d<=d_min THEN
     index=i
     xo=x
     yo=y
     d_min=d
   END IF
NEXT i
END SUB

!*****************************
! Subroutine: resort_array
!*****************************

SUB resort_array(a(,),index) ! i/o,i

   ! sort a(,) to have index pt comes first. copy 1st pt at last.
   ! this is cut order from 1 to end
   ! a(,) : point array for a loop, a(num_pt,2{x,y})
```

```
    ! index : start point index in a(,)

  IF index=1 THEN Exit sub      ! if index=1, original array

  DIM temp(0,0)
  num_pt=ubound(a,1)
  MAT REDIM temp(num_pt,2)

  count=0
  FOR i=index TO num_pt-1       ! remove last pt. last pt= first pt
    count=count+1
    temp(count,1)=a(i,1)
    temp(count,2)=a(i,2)
  NEXT i
  FOR i=1 TO index
    count=count+1
    temp(count,1)=a(i,1)
    temp(count,2)=a(i,2)
  NEXT i
  MAT a=temp
  END SUB

  !*****************************
  ! Subroutine: cutter_position
  !*****************************

  SUB cutter_position(d_allowed,decimal,x1,y1,x2,y2,x2_cut,y2_cut) !i(6),o(2)

    ! Find actual cutter position satisfying cutter length allowed
    ! If distance is over max, keep bot point and move top at max distance
    ! d_allowed   : distance allowed bw top and bot in top view
    ! decimal     : round decimal for cut point        (i,..,o,o)
    ! x1,y1,x2,y2 : bot, top matching point, start points from point file
    ! x2_cut,y2_cut: top point satisfying distance requirement

  d=((x1-x2)^2+(y1-y2)^2)^0.5   ! distance of calculated point

                ! find actual cutter position
  IF d>d_allowed THEN
    m=d_allowed
    n=d-d_allowed
    x2_cut=(m*x2+n*x1)/d     ! m:n ratio intesection of x1 and x2
    y2_cut=(m*y2+n*y1)/d
    x2_cut=round(x2_cut,decimal+1)
    y2_cut=round(y2_cut,decimal+1)
  ELSE
    x2_cut=round(x2,decimal+1)
    y2_cut=round(y2,decimal+1)
  END IF
  END SUB

  !*************************
  ! Subroutine: cut_a_loop
  !*************************

  SUB cut_a_loop(#1,#2,skip,start_b,end_b,start_t,end_t,index_b,index_t,&
        & x1s,y1s,x2s,y2s,dh,cut_angle,decimal,#5,#6)  ! i,..,o,o
```

```
! find cut path for a match loop.
! #1,start_b,end_b : bot pt file channel, rec num of a loop
! #2,start_t,end_t,: top pt file channel, rec num of a loop
! #5,#6 : output cut file, *.ct, *.ctx
! skip  : num of pt to skip
! index_b,index_t : index for bot and top match loop
! x1s,y1s,x2s,y2s : bot and top first point of a match loop
! dh,cut_angle,decimal : foam thickness. max cut angle. round decimal
! *.ctx : 13 rec for each match loop, and 5 rec at the end
!   (start num of cut,end num of cut,xbstart,ybstart,xtstart,ytstart,
!    xmin,xmax,ymin,ymax,index_b,index_t,max_angle)
!     ... (xmin,xmax,ymin,ymax,max_angle for all match loops)

DIM b(0,0),t(0,0)

CALL point_to_array(#1,start_b,end_b,b(,)) ! .,o
CALL point_to_array(#2,start_t,end_t,t(,)) ! .,o
xb=b(1,1)                          ! bot start pt
yb=b(1,2)
CALL find_closest_pt(t(,),xb,yb,index,xt,yt) ! .,o,o,o
CALL resort_array(t(,),index) ! i/o,i  cut path is from 1 to end
d_allowed=dh*tan(cut_angle*pi/180) ! distance allowed bw top and bot in top view
num_b=ubound(b,1)              ! num of pt
num_t=ubound(t,1)
x1s=b(1,1)                 ! first pt
y1s=b(1,2)
x2=t(1,1)
y2=t(1,2)
CALL cutter_position(d_allowed,decimal,x1s,y1s,x2,y2,x2s,y2s)
RESET #5:end
ASK #5:record rec_start
WRITE #6:rec_start
WRITE #5:x1s,y1s,x2s,y2s  ! save start cut pos
bn=1                ! current counter
tn=1
DO                  ! cut all pt on a loop
  FOR i=1 TO skip             ! skip n points
    IF bn<num_b THEN              ! read bot
      x1=b(bn+1,1)
      y1=b(bn+1,2)
      bn=bn+1
    END IF
    IF tn<num_t THEN              ! read bot
      x2=t(tn+1,1)
      y2=t(tn+1,2)
      tn=tn+1
    END IF
  NEXT i
  IF bn>=num_b THEN             ! read bot
    x1n=b(num_b,1)
    y1n=b(num_b,2)
  ELSE
    x1n=b(bn+1,1)
    y1n=b(bn+1,2)
  END IF
```

```
    IF tn>=num_t THEN            ! read top
      x2n=t(num_t,1)
      y2n=t(num_t,2)
    ELSE
      x2n=t(tn+1,1)
      y2n=t(tn+1,2)
    END IF
    !-- compare distance
    d_bot=(x1n-x2)^2+(y1n-y2)^2    ! if move bot point
    d_top=(x2n-x1)^2+(y2n-y1)^2    ! if move top point
    d_both=(x1n-x2n)^2+(y1n-y2n)^2 ! if move both point
    CALL min3(d_bot,d_top,d_both,d_min)
    IF tn>=num_t and bn>=num_b THEN
      d_min=d_both
    ELSE IF tn>=num_t THEN
      d_min=d_bot
    ELSE IF bn>=num_b THEN
      d_min=d_top
    END IF
    IF d_min=d_both THEN         ! move both
      x1=x1n
      y1=y1n
      x2=x2n
      y2=y2n
      bn=bn+1
      tn=tn+1
    ELSE IF d_min=d_top THEN     ! move top
      x2=x2n
      y2=y2n
      tn=tn+1
    ELSE                         ! move bot
      x1=x1n                     ! update
      y1=y1n
      bn=bn+1
    END IF
    CALL cutter_position(d_allowed,decimal,x1,y1,x2,y2,x2_cut,y2_cut)
    WRITE #5:x1,y1,x2_cut,y2_cut
  LOOP UNTIL bn>=num_b AND tn>=num_t
  WRITE #5:x1s,y1s,x2s,y2s   !copy start pos at last

  RESET #5:end               ! end rec num of a lp
  ASK #5:record rec_new_start
  rec_end=rec_new_start-1
  WRITE #6:rec_end

  num_ct=(rec_end-rec_start+1)/4
  xmin,ymin=1000             ! init
  xmax,ymax,dmax=-1000
  RESET #5:record rec_start

  FOR i=1 TO num_ct
    READ #5:x1,y1,x2,y2
  if x1>1000 or y1>1000 or x2>1000 or y2>1000 then
    print x1,y1,x2,y2
    print num_ct,i
  end if
```

```
   IF x1<xmin THEN xmin=x1
   IF y1<ymin THEN ymin=y1
   IF x2<xmin THEN xmin=x2
   IF y2<ymin THEN ymin=y2
   IF x1>xmax THEN xmax=x1
   IF y1>ymax THEN ymax=y1
   IF x2>xmax THEN xmax=x2
   IF y2>ymax THEN ymax=y2
   d=((x1-x2)^2+(y1-y2)^2)^0.5
   IF d>dmax THEN dmax=d
NEXT i

!print xmin,xmax,ymin,ymax

max_angle=atn(dmax/dh)*180/pi
WRITE #6:x1s,y1s,x2s,y2s
WRITE #6:xmin,xmax,ymin,ymax
WRITE #6:index_b,index_t,max_angle
END SUB

!******************
! reverse_cut_file
!******************

SUB reverse_cut_file(#1,#2)      ! i/o,i/o

   ! change top and bot record in *.ct, and *.ctx
   ! keep top shape and cut bot at max cut angle
   ! #1, #2 : *.ct, *.ctx channel

ASK #1: filesize num_rec_ct
ASK #2: filesize num_rec_ctx

num_ct=num_rec_ct/4
num_match=(num_rec_ctx-5)/13

FOR i=1 TO num_ct
   rec=(i-1)*4+1
   RESET #1:record rec
   READ #1:x1,y1,x2,y2
   RESET #1:record rec
   WRITE #1:x2,y2,x1,y1
NEXT i

FOR i=1 TO num_match
   rec=(i-1)*13+1
   RESET #2:record rec
   READ #2:start,end,x1s,y1s,x2s,y2s,x1,x2,y1,y2,index1,index2,angle
   RESET #2:record rec
   WRITE #2:start,end,x2s,y2s,x1s,y1s,x1,x2,y1,y2,index2,index1,angle
NEXT i

END SUB

!**************************
! Subroutine: point_to_cut
!**************************
```

```
SUB point_to_cut(name$,file_num,reverse,dh,cut_angle,decimal,skip,&
      & xsize,ysize,max_angle) ! i,..,o(3)

  ! Find multi loop cut path by moving point to point
  ! name$ : object name
  ! reverse : 0=no reverse, defaule, keep bot cross section
  !         1=reverse, keep top cross section
  ! dh,cut_angle,decimal : foam thickness, max cut angle, round decimal
  ! skip : num of pt to skip to go to next pt to cut
  ! x,ysize,max_angle : max x and y and angle for a layer(all loops)

DIM match(0,0)

file_ct$="object\" & name$ & "\" & str$(file_num) & ".ct"
file_ctx$="object\" & name$ & "\" & str$(file_num) & ".ctx"

IF reverse=0 THEN
  file_pt_b$="object\" & name$ & "\" & str$(file_num) & ".pt"
  file_pt_t$="object\" & name$ & "\" & str$(file_num+1) & ".pt"
ELSE
  file_pt_b$="object\" & name$ & "\" & str$(file_num+1) & ".pt"
  file_pt_t$="object\" & name$ & "\" & str$(file_num) & ".pt"
END IF

CALL open_file(#1,file_pt_b$,error)
   IF error=1 THEN exit sub
CALL open_file(#2,file_pt_t$,error)
   IF error=1 THEN exit sub
OPEN #5:name file_ct$,access Outin,create Newold,org record
ERASE #5
SET #5:Recsize 10
OPEN #6:name file_ctx$,access Outin,create Newold,org record
ERASE #6
SET #6:Recsize 10
CALL ptx_to_match(name$,file_num,match(,))  ! ,o
num_match=ubound(match,1)/2
FOR i=1 TO num_match

  IF reverse=0 THEN
    start_b=match(i*2-1,1)
    end_b=match(i*2-1,2)
    start_t=match(i*2,1)
    end_t=match(i*2,2)
    index_b=match(i*2-1,11)
    index_t=match(i*2,11)

  ELSE
    start_t=match(i*2-1,1)
    end_t=match(i*2-1,2)
    start_b=match(i*2,1)
    end_b=match(i*2,2)
    index_t=match(i*2-1,11)
    index_b=match(i*2,11)
  END IF

  CALL cut_a_loop(#1,#2,skip,start_b,end_b,start_t,end_t,index_b,index_t,&
```

```
      & x1s,y1s,x2s,y2s,dh,cut_angle,decimal,#5,#6) ! .o,o
NEXT i

ASK #6:filesize fsize
num_match=fsize/13
xmin,ymin=1000
xmax,ymax,max_angle=-1000
RESET #6:begin
FOR i=1 TO num_match              ! find slice size and max angle
  READ #6:start,end,x1s,y1s,x2s,y2s,x1,x2,y1,y2,index1,index2,angle
  IF x1<xmin THEN xmin=x1
  IF y1<ymin THEN ymin=y1
  IF x2<xmin THEN xmin=x2
  IF y2<ymin THEN ymin=y2
  IF x1>xmax THEN xmax=x1
  IF y1>ymax THEN ymax=y1
  IF x2>xmax THEN xmax=x2
  IF y2>ymax THEN ymax=y2
  IF angle>max_angle THEN max_angle=angle
NEXT i

ASK #5:filesize fsize
num_cut=fsize/4
max_angle=round(max_angle,1)
WRITE #6:xmin,xmax,ymin,ymax,max_angle
IF reverse<>0 THEN CALL reverse_cut_file(#5,#6)      ! i,i
xsize=xmax-xmin
ysize=ymax-ymin
PRINT num_cut;"cuts in ";str$(file_num);".ct"
PRINT "Slice size x/y: ";xsize;"/";ysize
PRINT "Max angle is";max_angle;"degree."
CLOSE #1
CLOSE #2
CLOSE #5
CLOSE #6
END SUB

!**********
! auto_cut
!**********

SUB auto_cut(name$,start,end,reverse,dh,cut_angle,decimal,skip) ! i,..

  ! cut from start tod end cut file num
  ! name$ : object name
  ! start,end : start and end file num *.ct
  ! reverse : 0=no reverse, defaule, keep bot cross section
  !         1=reverse, keep top cross section
  ! dh,cut_angle,decimal : foam thickness, max cut angle, round decimal
  ! skip : num of pt to skip to go to next pt to cut
FOR i=start TO end            ! create all loop files
  PRINT
  PRINT "Making file ";str$(i);".ct";" (";i;"/";end;").  Wait.."
  CALL point_to_cut(name$,i,reverse,dh,cut_angle,decimal,skip,&
       & xsize,ysize,max_angle) ! i/o,i.,o,o,o
NEXT i
```

```
PRINT end-start+1;"files are made. "
END SUB

!*****************************
! Subroutine: cut_a_loop_sm1
!*****************************
SUB cut_a_loop_sm1(#1,start_b,end_b,index_b,decimal,#5,#6)  ! i,..,o(2)
   ! find cut path for a loop for sm1, vertical cylindrical shape
   ! #1,start_b,end_b : bot channel, rec num of a loop
   ! index_b,decimal : index of bot. round decimal
   ! #5,#6 : *.ct, *.ctx
   ! *.ctx : 13 rec for each match loop, and 5 rec at the end
   !   (start num of cut,end num of cut,xbstart,ybstart,xtstart,ytstart,
   !    xmin,xmax,ymin,ymax,index_b,index_t,max_angle)
   !       ... (xmin,xmax,ymin,ymax,max_angle for all match loops)
DIM b(0,0)
CALL point_to_array(#1,start_b,end_b,b(,))  ! .,o
num_b=ubound(b,1)                ! num of pt
RESET #5:end
ASK #5:record rec_start
WRITE #6:rec_start
FOR i=1 TO num_b
   x1=b(i,1)
   y1=b(i,2)
   WRITE #5:x1,y1,x1,y1  ! save start cut pos
NEXT i
x1s=b(1,1)
y1s=b(1,2)
WRITE #5:x1s,y1s,x1s,y1s  ! save start cut pos

RESET #5:end             ! end rec num of a lp
ASK #5:record rec_new_start
rec_end=rec_new_start-1
WRITE #6:rec_end
num_ct=(rec_end-rec_start+1)/4
xmin,ymin=1000                ! init
xmax,ymax,dmax=-1000
RESET #5:record rec_start
FOR i=1 TO num_ct
   READ #5:x1,y1,x2,y2
   IF x1<xmin THEN xmin=x1
   IF y1<ymin THEN ymin=y1
   IF x2<xmin THEN xmin=x2
   IF y2<ymin THEN ymin=y2
   IF x1>xmax THEN xmax=x1
   IF y1>ymax THEN ymax=y1
   IF x2>xmax THEN xmax=x2
   IF y2>ymax THEN ymax=y2
   d=((x1-x2)^2+(y1-y2)^2)^0.5
   IF d>dmax THEN dmax=d
NEXT i
WRITE #6:x1s,y1s,x1s,y1s
WRITE #6:xmin,xmax,ymin,ymax
WRITE #6:index_b,index_b,max_angle
END SUB
```

```
!*******************************
! Subroutine: point_to_cut_sm1
!*******************************

SUB point_to_cut_sm1(name$,file_num,decimal)  ! i,i,i

   ! make cut file for sm1
   ! name$ : object name
   ! file_num : cut file num
   ! decimal : round decimal for *.ct

DIM match(0,0)

file_ct$="object\" & name$ & "\" & str$(file_num) & ".ct"
file_ctx$="object\" & name$ & "\" & str$(file_num) & ".ctx"

file_pt$="object\" & name$ & "\" & str$(file_num) & ".pt"

CALL open_file(#1,file_pt$,error)
   IF error=1 THEN exit sub
OPEN #5:name file_ct$,access Outin,create Newold,org record
ERASE #5
SET #5:Recsize 10
OPEN #6:name file_ctx$,access Outin,create Newold,org record
ERASE #6
SET #6:Recsize 10

CALL ptx_to_match(name$,file_num,match(,))   ! ,o

num_match=ubound(match,1)/2

FOR i=1 TO num_match

   start_b=match(i*2-1,1)
   end_b=match(i*2-1,2)
   start_t=match(i*2,1)
   end_t=match(i*2,2)
   index_b=match(i*2-1,11)
   index_t=match(i*2,11)

   CALL cut_a_loop_sm1(#1,start_b,end_b,index_b,decimal,#5,#6)
NEXT i

ASK #6:filesize fsize
num_match=fsize/13
xmin,ymin=1000
xmax,ymax,max_angle=-1000
RESET #6:begin

FOR i=1 TO num_match                ! find slice size and max angle
   READ #6:start,end,x1s,y1s,x2s,y2s,x1,x2,y1,y2,index1,index2,angle
   IF x1<xmin THEN xmin=x1
   IF y1<ymin THEN ymin=y1
   IF x2<xmin THEN xmin=x2
   IF y2<ymin THEN ymin=y2
   IF x1>xmax THEN xmax=x1
   IF y1>ymax THEN ymax=y1
```

```
   IF x2>xmax THEN xmax=x2
   IF y2>ymax THEN ymax=y2
   IF angle>max_angle THEN max_angle=angle
NEXT i

ASK #5:filesize fsize
num_cut=fsize/4
max_angle=round(max_angle,1)
WRITE #6:xmin,xmax,ymin,ymax,max_angle

xsize=xmax-xmin
ysize=ymax-ymin

PRINT num_cut;"cuts in ";str$(file_num);".ct"
PRINT "Slice size x/y: ";xsize;"/";ysize
PRINT "Max angle is";max_angle;"degree."

CLOSE #1
CLOSE #5
CLOSE #6
END SUB

!**************
! auto_cut_sm1
!**************

SUB auto_cut_sm1(name$,start,end,decimal)  ! i(4)

   ! make cut file from start to end
   ! name$ : object name
   ! start,end : start and end cut file num
   ! decimal : round decimal

FOR i=start TO end          ! create all loop files
   PRINT
   PRINT "Making file ";str$(i);".ct";" (";i;"/";end;").  Wait.."

   CALL point_to_cut_sm1(name$,i,decimal)
NEXT i

PRINT end-start+1;"files are made. "
END SUB

!**************
! End of File
!**************
```

```
!*********************************************************************
! Written by Cheol H. Lee
! File: 72-6-st.tru
! Date: 6/96
! Object : make step file reading cut file
! Input  : *.ct, *.ctx
! Output : *.st text file
!*********************************************************************
! MEMO
! - *.st file is text file to be read by C program
! - *.st file includes registration holes, and access and return path
!   to reg holes.
!*********************************************************************
library "71-ut.tru"
library "71-gct.tru"
library "utillib.tru"
library "3dlib.trc"
name$="carbody"
start,end=6
num_hole=2
dh=2.5
space_b=1.3
space_t=1.2
d_point=0.1
xh=0
yh=2
x1s,x2s,y2s=800
y1s=120.63
skip=2

x2w,y2w=130
!clear
call auto_step(name$,start,end,num_hole,dh,space_b,space_t,&
        & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)
call draw_auto_st_3d(name$,start,end,dh,space_b,space_t,x2w,y2w,x1s,x2s,y1s,y2s,xh,yh)
end
EXTERNAL
!*********
! Summary (72-6-st)
!*********
!    SUB slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,x1,x2,y1,y2,#5)   ! ..o
!    SUB access_path(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
!       & x1f,x2f,y1f,y2f,x1t,x2t,y1t,y2t,#5)     ! ..o
! SUB find_closest_cut_num(#1,x,y,choice,cut_num,x1c,y1c,x2c,y2c)
! SUB find_hole_start(#1,num_hole,hole(,))
! SUB hole_circle(hole(,),hole_num,r_bot,r_top,d_point,x1s,x2s,y1s,y2s,&
!         & x1f,x2f,y1f,y2f,xs,ys,#5)
! SUB cut_to_step(name$,file_num,num_hole,hole(,),dh,space_b,space_t,&
!        & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)
! SUB auto_step(name$,start,end,num_hole,dh,space_b,space_t,&
!          & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)

!************************
! Subroutine: slide_pos
!************************
```

```
SUB slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,x1,x2,y1,y2,#5) !i,..,o

   ! convert pos of cut file into abs step pos of slide with shift
   ! r_bot=space_b/dh, r_top=space_t/dh
   ! xs,ys  : shift xnew=xold+xs.
   ! x1,x2,y1,y2  : coord in ct file
   ! x1s,y1s : num steps for bot / cm
   ! x2s,y2s : num steps for top / cm
   ! #5  : *.st channel, abs step pos
 x1a=x1-(x2-x1)*r_bot     ! slide pos w/o shift
 x2a=x2+(x2-x1)*r_top
 y1a=y1-(y2-y1)*r_bot
 y2a=y2+(y2-y1)*r_top
 x1a=x1a+xs              ! shift
 x2a=x2a+xs
 y1a=y1a+ys
 y2a=y2a+ys
 x1sa=int(x1a*x1s)       ! slide abs coord in step
 x2sa=int(x2a*x2s)
 y1sa=int(y1a*y1s)
 y2sa=int((y2a-y1a)*y2s)
 PRINT #5:x1sa
 PRINT #5:y1sa
 PRINT #5:x2sa
 PRINT #5:y2sa
 END SUB

 !*****************************
 ! Subroutine: slide_pos_point
 !*****************************

 SUB slide_pos_point(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
      & x1f,x2f,y1f,y2f,x1t,x2t,y1t,y2t,xs,ys,#5)     ! i,..,o

   ! divide a line with d_point and skip n point
   ! r_bot=space_b/dh, r_top=space_t/dh
   ! d_point,skip : distance bw points, num of points to skip
   ! x1s,y1s,x2s,y2s : bot and top ratio for motors steps/cm
   ! x1f,x2f,y1f,y2f : abs coord of start point of a line in cm
   ! x1t,x2t,y1t,y2t : abs coord of end point of a line in cm
   ! xs,ys  : shift xnew=xold+xs.
   ! #5 : step file. *.st
dx1=x1t-x1f              ! dis bw each coord
dx2=x2t-x2f
dy1=y1t-y1f
dy2=y2t-y2f
d_bot=(dx1^2+dy1^2)^0.5
d_top=(dx2^2+dy2^2)^0.5
d=max(d_bot,d_top)       ! find max dis bw top and bot

IF d=0 THEN exit sub

d_p=d_point*(skip+1)     ! new d_point with skip
num_pt=int(d/d_p)        ! num of pt on a line

IF num_pt <>0 THEN       ! dis increment in cm
  ddx1=dx1/num_pt
```

```
  ddx2=dx2/num_pt
  ddy1=dy1/num_pt
  ddy2=dy2/num_pt

  FOR i=1 TO num_pt          ! cal step size and save
     x1=x1f+ddx1*i          ! coord to go in cm
     x2=x2f+ddx2*i
     y1=y1f+ddy1*i
     y2=y2f+ddy2*i

     CALL slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,x1,x2,y1,y2,#5) ! ..o
  NEXT i
END IF
CALL slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,x1t,x2t,y1t,y2t,#5)   ! ..o
END SUB
!************************
! Subroutine: access_path
!************************
SUB access_path(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
      & x1f,x2f,y1f,y2f,x1t,x2t,y1t,y2t,#5)     ! i,..,o
   ! access path from xf to xt
   ! r_bot=space_b/dh, r_top=space_t/dh
   ! d_point,skip : distance bw points, num of points to skip
   ! x1s,y1s,x2s,y2s : bot and top ratio for motors steps/cm
   ! x1f,x2f,y1f,y2f : abs coord of start point of a line in cm
   ! x1t,x2t,y1t,y2t : abs coord of end point of a line in cm
   ! #5 : step file. *.st
xs,ys=0
CALL slide_pos_point(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
      & x1f,x2f,y1f,y2f,x1f,x1f,0,0,xs,ys,#5)     ! ..o
CALL slide_pos_point(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
      & x1f,x1f,0,0,x1t,x1t,0,0,xs,ys,#5)     ! ..o
CALL slide_pos_point(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
      & x1t,x1t,0,0,x1t,x2t,y1t,y2t,xs,ys,#5)     ! ..o
END SUB

!*********************************
! Subroutine: find_closest_cut_num
!*********************************
SUB find_closest_cut_num(#1,x,y,choice,cut_num,x1c,y1c,x2c,y2c) ! i(4),o(5)

   ! find closest cut num from a point (x,y)
   ! #1: cut file. *.ct
   ! x,y: point to find crosest point from
   ! choice : access direction
   !        1=0 deg, 2=90 deg, 3=180 deg,4=270 deg, 0=min dis
   ! cut_num: cut num in cut file.
   ! x1c,y1c,x2c,y2c : cut record
 ASK #1:filesize num_rec
 num_cut=(num_rec-5)/4
 cut_num=0
 dmin=10000
 RESET #1:Begin
 READ #1:x1old,y1old,x2old,y2old

 FOR i=2 TO num_cut
    READ #1:x1,y1,x2,y2
```

```
    xsign=(x-x1)*(x-x1old)
    ysign=(y-y1)*(y-y1old)
    d=(x-x1)^2+(y-y1)^2
    dy=abs(y1-y)
    dx=abs(x1-x)

    IF choice=0 THEN
      IF d<dmin THEN
        dmin=d
        cut_num=i
      END IF
    ELSE IF choice=1 THEN
      IF ysign<=0 and x<=x1 and dx<dmin THEN
        dmin=d
        cut_num=i
      END IF
    ELSE IF choice=2 THEN
      IF xsign<=0 and y<=y1 and dy<dmin THEN
        dmin=d
        cut_num=i
      END IF
    ELSE IF choice=3 THEN
      IF ysign<=0 and x>=x1 and dx<dmin THEN
        dmin=d
        cut_num=i
      END IF
    ELSE IF choice=4 THEN
      IF xsign<=0 and y>=y1 and dy<dmin THEN
        dmin=d
        cut_num=i
      END IF
    END IF
    IF cut_num=i THEN
      x1c=x1
      x2c=x2
      y1c=y1
      y2c=y2
    END IF
    x1old=x1
    x2old=x2
    y1old=y1
    y2old=y2
  NEXT i
  END SUB

  !************************************
  ! Subroutine: find_hole_start
  !************************************

  SUB find_hole_start(#1,num_hole,hole(,))  ! i,i,i/o

    ! find cut_num, rad_start for all holes.
    ! #1: cut file, *.ct.
    ! num_hole : num of registration holes
    ! hole(,) : reg hole info array. hole(num of hole,7)
    !       (choice,xc,yc,radi,cut_num,rad_start,resolution)
    ! choice : access direction
```

```
!       1=0 deg, 2=90 deg, 3=180 deg,4=270 deg, 0=min dis
! xc,yc  : center of reg hole
! radi   : radius of reg hole
! cut_num : cut num of access pos in cut file
! rad_start : access radian to start to cut a hole
! resolution : num of points for a reg hole

ASK #1:filesize num_rec
num_cut=(num_rec-5)/4
FOR i=1 TO num_hole
  choice=hole(i,1)
  xc=hole(i,2)
  yc=hole(i,3)
  radi=hole(i,4)
  IF choice=1 THEN        ! start 0 deg
    rad_start=0
    x=xc+radi
    y=yc
    CALL find_closest_cut_num(#1,x,y,choice,cut_num,x1c,y1c,x2c,y2c)
  ELSE IF choice=2 THEN       ! start 90 deg
    rad_start=pi/2
    x=xc
    y=yc+radi
    CALL find_closest_cut_num(#1,x,y,choice,cut_num,x1c,y1c,x2c,y2c)
  ELSE IF choice=3 THEN       ! start 180 deg
    rad_start=pi
    x=xc-radi
    y=yc
    CALL find_closest_cut_num(#1,x,y,choice,cut_num,x1c,y1c,x2c,y2c)
  ELSE IF choice=4 THEN       ! start 270 deg
    rad_start=pi*1.5
    x=xc
    y=yc-radi
    CALL find_closest_cut_num(#1,x,y,choice,cut_num,x1c,y1c,x2c,y2c)
  ELSE IF choice=0 THEN       ! closest from xc,yc
    x=xc
    y=yc
    CALL find_closest_cut_num(#1,x,y,choice,cut_num,x1c,y1c,x2c,y2c)
    dx=x1c-x1
    dy=y1c-x2
    IF dx=0 THEN
      PRINT "Hole is outbound."
      GET KEY key
      Exit sub
    ELSE
      rads=atn(abs(dy/dx))
    END IF
    IF dx>=0 and dy>=0 THEN
      rad_start=rads
    ELSE IF dx<=0 and dy>=0 THEN
      rad_start=pi-rads
    ELSE IF dx<=0 and dy<=0 THEN
      rad_start=pi+rads
    ELSE IF dx>=0 and dy<=0 THEN
      rad_start=2*pi-rads
    END IF
  END IF
```

```
    hole(i,5)=cut_num          ! save info
    hole(i,6)=rad_start
  NEXT i
  END SUB

  !************************
  ! Subroutine: hole_circle
  !************************
  SUB hole_circle(hole(,),hole_num,r_bot,r_top,d_point,x1s,x2s,y1s,y2s,&
        & x1f,x2f,y1f,y2f,xs,ys,#5)  ! i,..,o
    ! calculate all pt and step for a hole and save in *.st file
    ! hole(,) : reg hole info array. hole(num of hole,7)
    !       (choice,xc,yc,radi,cut_num,rad_start,resolution)
    ! hole_num : hole num in hole(,)
    ! r_bot=space_b/dh, r_top=space_t/dh
    ! d_point : distance bw points
    ! x1s,y1s,x2s,y2s : bot and top ratio for motors steps/cm
    ! x1f,x2f,y1f,y2f : abs coord of start point of a line in cm
    ! xs,ys : shift in x,y direction
    ! #5 : step file *.st

  choice=hole(hole_num,1)
  xc=hole(hole_num,2)
  yc=hole(hole_num,3)
  radi=hole(hole_num,4)
  cut_num=hole(hole_num,5)
  rad_start=hole(hole_num,6)
  resolution=hole(hole_num,7)

  !-----------
  ! access hole
  x_start=xc+cos(rad_start)*radi  ! start pt on a hole
  y_start=yc+sin(rad_start)*radi
  CALL slide_pos_point(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
        & x1f,x2f,y1f,y2f,x_start,x_start,y_start,y_start,xs,ys,#5)
  !---------
  ! cut hole
  drad=2*pi/resolution
  n1=int((2*pi-rad_start)/drad) ! num of pt on a arc from rad_start to 2pi
  n2=int(rad_start/drad)        ! num of pt on a arc from 0 to rad_start
  FOR i=1 TO n1                 ! cut angle from rad_start to 2*pi
    rad=rad_start+i*drad
    xout=xc+cos(rad)*radi
    yout=yc+sin(rad)*radi
    CALL slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,&
          & xout,xout,yout,yout,#5)
  NEXT i
  xout=xc+radi                  ! path 0 deg pt
  CALL slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,xout,xout,yc,yc,#5)
  FOR i=1 TO n2                 ! cut angle from 0 to rad_start
    rad=i*drad
    xout=xc+cos(rad)*radi
    yout=yc+sin(rad)*radi
    CALL slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,xout,xout,yout,yout,#5)
  NEXT i
                  ! pass start point
  CALL slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,x_start,x_start,y_start,y_start,#5)
```

```
!----------------
! return to loop

CALL slide_pos_point(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
      & x_start,x_start,y_start,y_start,x1f,x2f,y1f,y2f,xs,ys,#5)
END SUB

!*************************
! Subroutine: cut_to_step
!*************************

SUB cut_to_step(name$,file_num,num_hole,hole(,),dh,space_b,space_t,&
      & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)  ! all input

   ! read cut file and make step file including access path, return path,
   ! and registration holes
   ! name$ : object name
   ! file_num : file num for *.ct, *.st
   ! num_hole : num of reg hole
   ! hole(,) : reg hole info array. hole(num of hole,7)
   !       (choice,xc,yc,radi,cut_num,rad_start,resolution)
   ! dh,d_point : foam thickness, distance bw points
   ! space_b,space_t : distance bw top and bot control point and the foam
   ! skip : num of point to skip
   ! xh,yh : x and y home position
   ! x1s,y1s,x2s,y2s : bot and top ratio for motors steps/cm

file_ct$="object\" & name$ & "\" & str$(file_num) & ".ct"
file_ctx$="object\" & name$ & "\" & str$(file_num) & ".ctx"
file_st$="object\" & name$ & "\" & str$(file_num) & ".st"

CALL open_file(#1,file_ct$,error)
   IF error=1 THEN exit sub
CALL open_file(#2,file_ctx$,error)
   IF error=1 THEN exit sub
OPEN #5:name file_st$,access outin, create newold, org text
ERASE #5
ASK #2:filesize num_rec
RESET #2:record num_rec-4
READ #2:xmin,xmax,ymin,ymax,angle
RESET #2:begin
num_match=(num_rec-5)/13
r_bot=space_b/dh
r_top=space_t/dh
xs=xh-xmin                    ! calc shift
ys=yh-ymin
FOR i=1 TO num_match          ! find step for all loops

   IF i=1 THEN                ! start point
     x1f,x2f=0  ! first start pos=0
     y1f,y2f=0
   ELSE
     x1f=x1t
     x2f=x2t
     y1f=y1t
     y2f=y2t
   END IF
```

```
    READ #2:start,end,xbs,ybs,xts,yts,xmin,xmax,ymin,ymax,index_b,index_t,angle
                          ! a slice info
      x1t=xbs+xs
      x2t=xts+xs
      y1t=ybs+ys
      y2t=yts+ys
    CALL access_path(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
          & x1f,x2f,y1f,y2f,x1t,x2t,y1t,y2t,#5)    ! ..o
    CALL find_hole_start(#1,num_hole,hole(,))
    num_ct=(end-start+1)/4
    RESET #1:record start
    FOR j=1 TO num_ct
      READ #1:x1,y1,x2,y2
      CALL slide_pos(r_bot,r_top,xs,ys,x1s,x2s,y1s,y2s,x1,x2,y1,y2,#5)   ! ..o
      FOR k=1 TO num_hole
        cut_num=hole(k,5)
        IF cut_num=j THEN
          CALL hole_circle(hole(,),k,r_bot,r_top,d_point,x1s,x2s,&
            & y1s,y2s,x1,x2,y1,y2,xs,ys,#5)
        END IF
      NEXT k
    NEXT j
  NEXT i

  CALL access_path(r_bot,r_top,d_point,skip,x1s,x2s,y1s,y2s, &
        & x1t,x2t,y1t,y2t,0,0,0,0,#5)    ! ..o
  CLOSE #1
  CLOSE #2
  CLOSE #5
  END SUB
  !************************
  ! Subroutine: auto_step
  !************************
  SUB auto_step(name$,start,end,num_hole,dh,space_b,space_t,&
        & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)
    ! make step file from start to end including access path, return path,
    ! and registration holes
    ! name$ : object name
    ! start,end : start and end file num for *.ct, *.st
    ! num_hole : num of reg hole
    ! dh,d_point : foam thickness, distance bw points
    ! space_b,space_t : distance bw top and bot control point and the foam
    ! skip : num of point to skip
    ! xh,yh : x and y home position
    ! x1s,y1s,x2s,y2s : bot and top ratio for motors steps/cm
  DIM hole(0,0)
  IF num_hole<>0 THEN        ! make hole(,)
    MAT REDIM hole(num_hole,7)
    INPUT PROMPT "radius(cm), resolution(n), alternative(1/0)? " &
          & :radi,resolution,alt
    PRINT
    PRINT "choice : 0=min dis, 1=0 deg, 2=90 deg, 3=180 deg, 4=270 deg"
    PRINT

    FOR i=1 TO num_hole
      INPUT PROMPT "Hole "&str$(i)&": choice(0-4),xc,yc? ":choice,xc,yc
      hole(i,1)=choice
```

```
    hole(i,2)=xc
    hole(i,3)=yc
    hole(i,4)=radi
    hole(i,7)=resolution
  NEXT i
END IF
FOR i=start TO end       ! make step files
  PRINT
  PRINT "Making file ";str$(i);".st";" (";i;"/";end;").  Wait.."
  CALL cut_to_step(name$,i,num_hole,hole(,),dh,space_b,space_t,&
      & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)
  IF alt=1 THEN                  ! alt reg hole
    FOR j=1 TO num_hole
      IF hole(j,1)=1 THEN
        hole(j,1)=3
      ELSE IF hole(j,1)=2 THEN
        hole(j,1)=4
      ELSE IF hole(j,1)=3 THEN
        hole(j,1)=1
      ELSE IF hole(j,1)=4 THEN
        hole(j,1)=2
      END IF
    NEXT j
  END IF

NEXT i

PRINT end-start+1;"files are made. "
END SUB

!*************
! End of file
!*************
```

```
!*********************************************************************
! Written by Cheol Lee
! File: 72-gct.tru
! Date: 10/95
! Object : plot *.ct, *.st and *.ln for registration holes
! Input  : *.ct, *.st, *.lp
! Output : 2D or 3D drawing for *.ct and *.st, and drawing for reg holes
!*********************************************************************
library "71-ut.tru"
library "71-gln.tru"
library "71-5-ct1.tru"
library "utillib1.tru"
library "3dlib.trc"
name$="fordcar"
start,end=6
num_hole=0
dh=2.54/2
space_b,space_t=1
d_point=0.2
xh=1
yh=1
x1s,x2s,y2s=800
y1s=150
x2w,y2w=50
!call draw_auto_st_3d(name$,start,end,dh,space_b,space_t,x2w,y2w,x1s,x2s,y1s,y2s,xh,yh)
!call registration(1,5,50,50)
!stop
!call draw_top_bot_2d(name$,file_num,x2w,y2w)
!stop
call draw_auto_cut_2d(name$,1,1,2.74/2,x2w,y2w)
!call registration(1,10,100,100)
end
EXTERNAL
!*********
! Summary (72-gct)
!*********
! SUB draw_top_bot_2d(file_num,x2w,y2w)
!    SUB draw_cut_3d(name$,file_num,dh,x2w,y2w)
! SUB draw_auto_cut_2d(name$,start,end,dh,x2w,y2w)
!    SUB draw_st_3d(file_num,dh,space_b,space_t,x2w,y2w,x1s,x2s,y1s,y2s,xh,yh)
! SUB draw_auto_st_3d(name$,start,end,dh,space_b,space_t,x2w,y2w,x1s,x2s,y1s,y2s,xh,yh)

! SUB registration(start,end,xmax,ymax)
!*****************************
! Subroutine: draw_top_bot_2d
!*****************************
SUB draw_top_bot_2d(name$,file_num,x2w,y2w)  ! i(4)

  ! Read *.ct and plot it in 2D
  ! name$ : object name
  ! file_num : cut coordinate file number, *.ct
  ! 0,x2w,0,y2w : window size

CALL draw_loop(name$,file_num,0,x2w,y2w)
CALL draw_loop(name$,file_num+1,0,x2w,y2w)
END SUB
```

```
!*************************
! Subroutine: draw_cut_3d
!*************************

SUB draw_cut_3d(name$,file_num,dh,x2w,y2w)  ! i(5)

   ! Read *.ct and plot the edges and cutter motion in 3D
   ! name$ : object name
   ! file_num : cut coordinate file number, *.ct
   ! dh  : slice thickness
   ! 0,x2w,0,y2w : window size
file_bot$="object\" & name$ & "\" & Str$(file_num) & ".pt"
file_top$="object\" & name$ & "\" & Str$(file_num+1) & ".pt"
file_cut$="object\" & name$ & "\" & Str$(file_num) & ".ct"
file_ctx$="object\" & name$ & "\" & Str$(file_num) & ".ctx"
CALL open_file(#1,file_bot$,error)
   IF error=1 THEN exit sub
CALL open_file(#2,file_top$,error)
   IF error=1 THEN exit sub
CALL open_file(#3,file_cut$,error)
   IF error=1 THEN exit sub
CALL open_file(#4,file_ctx$,error)
   IF error=1 THEN exit sub
ASK #1:FILESIZE num_rec_b
ASK #2:FILESIZE num_rec_t
ASK #3:FILESIZE num_rec_cut
ASK #4:FILESIZE num_rec_ctx
LET num_point_b=num_rec_b/2
LET num_point_t=num_rec_t/2
LET num_cut=num_rec_cut/4
LET num_match=(num_rec_ctx-5)/13
CLEAR
CALL PersWindow(0,x2w,0,y2w,0,dh)   ! define window size in 3d
SET BACK "red"
SET COLOR "white"
!CALL AskScale3(hor,ver)
!   IF hor<1.2 THEN CALL SetScale3(1.2,1.2)
CALL Axes3
CALL PlotText3(x2w,0,0,"X")
CALL PlotText3(0,y2w,0,"Y")
CALL PlotText3(0,0,dh,"Z")
CALL PlotText3(0,0,0,"O")

SET COLOR "yellow"
PRINT num_point_b;"points in ";str$(file_num);".pt"
PRINT num_point_t;"points in ";str$(file_num+1);".pt"
PRINT num_cut;"cut steps in ";str$(file_num);".ct"

RESET #4:record num_rec_ctx-4
READ #4:xmin,xmax,ymin,ymax,max_angle
xsize=xmax-xmin
ysize=ymax-ymin

PRINT
PRINT ""
PRINT " Slice size x/y  : ";xsize;"/";ysize
PRINT " Max angle:";max_angle
```

```
!-----------------------------
! draw top and bot point file

SET COLOR "blue"
FOR i=1 TO num_point_b       ! plot bot edge
  READ #1:x1,y1
  CALL PlotOff3(x1,y1,0)
NEXT i
!CALL PlotOff3(x1,y1,0)
SET COLOR "white"
FOR i=1 TO num_point_t       ! plot top edge
  READ #2:x2,y2
  CALL PlotOff3(x2,y2,dh)
NEXT i
!CALL PlotOff3(x2,y2,dh)
!-----------------
! plot cutter path
SET COLOR "yellow"
FOR i=1 TO num_match
  RESET #4:record (i-1)*13+1
  READ #4:rec_start,rec_end
  n_cut=(rec_end-rec_start+1)/4
  RESET #3:record rec_start
  FOR j=1 TO n_cut          ! plot cutter motion
    READ #3:x1,y1,x2,y2
    CALL PlotOff3(x1,y1,0)   ! draw top bot cutter points
    CALL PlotOff3(x2,y2,dh)
  NEXT j
NEXT i

CLOSE #1
CLOSE #2
CLOSE #3
CLOSE #4
END SUB
!*********************************
! Subroutine: draw_auto_cut_2d
!*********************************
SUB draw_auto_cut_2d(name$,start,end,dh,x2w,y2w) ! i(6)
  ! Read from start.ct to end.ct, and plot cutter motion in 3D
  ! name$ : object name
  ! start,end : start and end num of cut file, *.ct
  ! dh  : slice thickness
  ! 0,x2w,0,y2w : window size

FOR i=start TO end
  CLEAR
  CALL draw_cut_3d(name$,i,dh,x2w,y2w)
  GET KEY getkey
NEXT i
END SUB
!*************************
! Subroutine: draw_st_3d
!*************************
SUB draw_st_3d(name$,file_num,dh,space_b,space_t,x2w,y2w,x1s,x2s,y1s,y2s,xh,yh)
                              ! all input
  ! Read *.st and plot cutter motion in 3D
```

```
  ! name$ : object name
  ! file_num : file num of *.st
  ! dh  : slice thickness
  ! space_b,space_t : space bw top and bottom pivot point and form
  ! 0,x2w,0,y2w : window size
  ! x1s,x2s,y1s,y2s : step/cm conversion for x,y direction of top and bot
  ! xh,yh : home position from the orgin of a coord system

file_bot$="object\" & name$ & "\" & Str$(file_num) & ".pt"
file_top$="object\" & name$ & "\" & Str$(file_num+1) & ".pt"
file_ct$="object\" & name$ & "\" & Str$(file_num) & ".ct"
file_ctx$="object\" & name$ & "\" & Str$(file_num) & ".ctx"
file_st$="object\" & name$ & "\" & Str$(file_num) & ".st"

CALL open_file(#1,file_bot$,error)
   IF error=1 THEN exit sub
CALL open_file(#2,file_top$,error)
   IF error=1 THEN exit sub
CALL open_file(#3,file_ct$,error)
   IF error=1 THEN exit sub
CALL open_file(#4,file_ctx$,error)
   IF error=1 THEN exit sub
OPEN #5:name file_st$,access input, org text
ASK #1:FILESIZE num_rec_b
ASK #2:FILESIZE num_rec_t
ASK #3:FILESIZE num_rec_ct
ASK #4:FILESIZE num_rec_ctx
LET num_point_b=num_rec_b/2
LET num_point_t=num_rec_t/2
LET num_ct=(num_rec_ct-5)/4
RESET #4:record num_rec_ctx-4
READ #4:xmin,xmax,ymin,ymax,max_angle
CALL PersWindow(0,x2w,0,y2w,0,dh)  ! define window size in 3d
SET BACK "red"
SET COLOR "white"
!CALL AskScale3(hor,ver)
!   IF hor<1.2 THEN CALL SetScale3(1.2,1.2)
CALL Axes3
CALL PlotText3(x2w,0,0,"X")
CALL PlotText3(0,y2w,0,"Y")
CALL PlotText3(0,0,dh,"Z")
CALL PlotText3(0,0,0,"O")
SET COLOR "yellow"
PRINT num_point_b;"points in ";str$(file_num);".pt"
PRINT num_point_t;"points in ";str$(file_num+1);".pt"
PRINT num_ct;"cuts in ";str$(file_num);".ct"
PRINT
PRINT ""
PRINT " Slice size x/y  : ";xmin;xmax;"/";ymin;ymax
PRINT "          dx/dy: ";xmax-xmin;"/";ymax-ymin
PRINT " Max angle:";max_angle

!-----------------------------
! draw cut file

xs=xh-xmin
ys=yh-ymin
```

```
  SET COLOR "blue"
  RESET #3:begin

  FOR i=1 TO num_ct      ! plot bot edge
    READ #3:x1,y1,x2,y2

    x1=x1+xs
    x2=x2+xs
    y1=y1+ys
    y2=y2+ys

    CALL PlotOff3(x1,y1,0)
    CALL PlotOff3(x2,y2,dh)
  NEXT i
  !------------------
  ! plot cutter path
  SET COLOR "yellow"
  h=space_b+space_t+dh
  ratio_b=(space_b)/h
  ratio_t=(space_b+dh)/h
  x1o,x2o,y1o,y2o=0
  RESET #5:Begin
  DO WHILE MORE #5          ! plot cutter motion
    INPUT #5:x1$
    INPUT #5:y1$
    INPUT #5:x2$
    INPUT #5:y2$

    x1=Val(x1$)
    x2=Val(x2$)
    y1=Val(y1$)
    y2=Val(y2$)

    x1=x1/x1s             ! abs coord of slide in cm
    x2=x2/x2s
    y1=y1/y1s
    y2=y1+y2/y2s
    xb=x1+(x2-x1)*ratio_b     ! abs coord on the foam in cm
    xt=x1+(x2-x1)*ratio_t
    yb=y1+(y2-y1)*ratio_b
    yt=y1+(y2-y1)*ratio_t
  set color "yellow"
    CALL lineOff3(x1o,y1o,0,xb,yb,0)    ! draw top bot cutter points
    CALL lineOff3(x2o,y2o,dh,xt,yt,dh)
  set color "blue"
    CALL lineOff3(x1o,y1o,0,xb,yb,0)    ! draw top bot cutter points
    CALL lineOff3(x2o,y2o,dh,xt,yt,dh)
    x1o=xb
    x2o=xt
    y1o=yb
    y2o=yt
  LOOP
  CLOSE #1
  CLOSE #2
  CLOSE #3
  CLOSE #4
  CLOSE #5
```

```
END SUB
!*********************************
! Subroutine: draw_auto_st_3d
!*********************************
SUB draw_auto_st_3d(name$,start,end,dh,space_b,space_t,x2w,y2w,x1s,x2s,y1s,y2s,xh,yh)
                                    ! all input
  ! Read from start.st to end.st, and plot cutter motion in 3D
  ! name$ : object name
  ! start,end: start and end file num, *.st
  ! dh  : slice thickness
  ! space_b,space_t : space bw top and bottom pivot point and form
  ! 0,x2w,0,y2w : window size
  ! x1s,x2s,y1s,y2s : step/cm conversion for x,y direction of top and bot
  ! xh,yh : home position from the orgin of a coord system
FOR i=start TO end
   CLEAR
   CALL draw_st_3d(name$,i,dh,space_b,space_t,x2w,y2w,x1s,x2s,y1s,y2s,xh,yh)
   GET KEY getkey
NEXT i
END SUB
!*****************************
! Subroutine: Registration
!*****************************
SUB registration(object$,start,end,xmax,ymax)  ! i(5)
  ! Read *.ln, and plot it in 2D for registration hole.
  ! start,end : start and end *.ln file num to decide reg holes
  ! 0,xmax,0,ymax : window size
OPEN #5:SCREEN 0,1,0,0.8
SET WINDOW 0,xmax,0,ymax     ! define window coordinate
SET COLOR "yellow/red"
OPEN #6:SCREEN 0,.5,0.8,1
SET COLOR "yellow/red"
OPEN #7:SCREEN 0.5,1,0.8,1
SET COLOR "yellow/red"
WINDOW #5
FOR i=start TO end
  file_ln$="object\" & object$ & "\" & str$(i) & ".ln"
  CALL open_file(#1,file_ln$,error)
     IF error=1 THEN exit sub
  ASK #1:FILESIZE record_num
  line_num=record_num/4    ! compute number of points
  FOR j=1 TO line_num
    READ #1:x1,y1,x2,y2
    PLOT x1,y1;x2,y2
  NEXT j
  CLOSE #1
NEXT i
DO
  WINDOW #6
  CLEAR
  PRINT " Select choice"
  PRINT " (0) exit."
  PRINT " (1) mouse."


  PRINT " (2) draw hole."
  PRINT " (3) redraw."
  INPUT PROMPT "choice  ":n
```

```
    IF n=0 THEN
      exit sub
    ELSE IF n=1 THEN
      WINDOW #6
      CLEAR
      PRINT " x / y"," dx / dy"
      PRINT
      PRINT "File:";start;"to";end
      PRINT "Press SHIFT-CLICK to quit."
      xold,yold=0
      DO
       WINDOW #5
       GET MOUSE x,y,ch
       IF ch=2 THEN      ! click
         x=round(x,1)
         y=round(y,1)
         dx=abs(x-xold)
         dy=abs(y-yold)
         WINDOW #7
         PRINT x;y,dx;dy
         xold=x
         yold=y
       END IF
      LOOP UNTIL ch=4
    ELSE IF n=2 THEN
      WINDOW #6
      CLEAR
      INPUT PROMPT "number of holes?  ":num_hole
      FOR i =1 TO num_hole
        WINDOW #6
        INPUT PROMPT "radius,xc,yc?  ":radi,xc,yc
        WINDOW #5
        SET COLOR "blue"
        BOX circle xc-radi,xc+radi,yc-radi,yc+radi
        SET COLOR "yellow"
      NEXT i
    ELSE IF n=3 THEN
      WINDOW #5
      CLEAR
      FOR i=start TO end
        file_ln$="object\" & object$ & "\" & str$(i) & ".ln"
        CALL open_file(#1,file_ln$,error)
          IF error=1 THEN exit sub
        ASK #1:FILESIZE record_num
        line_num=record_num/4     ! compute number of points
        FOR j=1 TO line_num
          READ #1:x1,y1,x2,y2
          PLOT x1,y1;x2,y2
        NEXT j
        CLOSE #1
      NEXT i
    END IF
  LOOP
  END SUB
  !*************
  ! End of File
  !*************
```

```
!*****************************************************************
! Written by Cheol Lee
! File: 72-gln.tru
! Date: 10/95
! Object : draw *.ln, *.lp, *.pt file
! Input  : *.ln, *.lp, *.pt
! Output : 2D or 3D drawing of files
!*****************************************************************
dim a(0,0),b(0,0)
library "69-1-vtx.tru"
library "69-ut.tru"
library "3dlib.trc"
xmin,ymix=0
xmax,ymax=100
!call draw_line(1,0,"yellow",xmin,xmax,ymin,ymax)
! call registration(1,50,100,100)
! call draw_auto_loop_2d(3,3,0,10,10)
end
EXTERNAL
!*********
! Summary (72-gln)
!*********
! SUB draw_line(name$,file_ln,sec,x2w,y2w)
! SUB draw_loop(name$,file_lp,sec,x2w,y2w)
! SUB draw_line_array(a(,),dim_a1,sec,x2w,y2w)
! SUB draw_point(name$,file_pt,sec,x2w,y2w)
! SUB draw_bounds(file_lpx,x2w,y2w)

! SUB draw_auto_line_2d(start,end,sec,x2w,y2w)
! SUB draw_auto_loop_2d(start,end,sec,x2w,y2w)
! SUB draw_auto_point_2d(start,end,sec,x2w,y2w)
! SUB draw_auto_line_3d(h_bot,h_top,dh,x2w,y2w,z2w)

!***********************
! Subroutine: draw_line
!***********************
SUB draw_line(name$,file_ln,sec,x2w,y2w)  ! i(5)
   ! Read *.ln and plot it in 2D
   ! name$ : object name
   ! file_ln : line coordinate file number, *.ln
   ! sec    : pause time between drawing lines
   ! 0,x2w,0,y2w : window size
file_ln$="object\" & name$ & "\" & str$(file_ln) & ".ln"
CALL open_file(#1,file_ln$,error)
   IF error=1 THEN exit sub
SET WINDOW 0,x2w,0,y2w    ! define window coordinate
SET COLOR "yellow"
SET BACK "red"

ASK #1:FILESIZE record_num
LET line_num=record_num/4        ! compute number of lines

FOR i=1 TO line_num
   READ #1:x1,y1,x2,y2
   PLOT x1,y1;x2,y2
   PAUSE sec
NEXT i
```

```
  CLOSE #1
  PRINT line_num;"lines in ";str$(file_ln);".ln"
  END SUB
  !************************
  ! Subroutine: draw_loop
  !************************
  SUB draw_loop(name$,file_lp,sec,x2w,y2w)  ! i(5)

    ! Read *.lp and plot it in 2D
    ! name$ : object name
    ! file_lp : loop coordinate file number, *.lp
    ! sec     : pause time between drawing lines
    ! 0,x2w,0,y2w : window size

  file_lp$="object\" & name$ & "\" & str$(file_lp) & ".lp"
  file_lpx$="object\" & name$ & "\" & str$(file_lp) & ".lpx"

  CALL open_file(#1,file_lp$,error)
     IF error=1 THEN exit sub
  CALL open_file(#2,file_lpx$,error)
     IF error=1 THEN exit sub

  ASK #1:filesize num_lp_rec
  ASK #2:filesize num_idx_rec
  num_line=num_lp_rec/4
  num_loop=num_idx_rec/10

  SET WINDOW 0,x2w,0,y2w     ! define window coordinate
  SET COLOR "yellow"
  SET BACK "red"

  FOR i=1 TO num_line
     READ #1:x1,y1,x2,y2
     PLOT x1,y1;x2,y2
     PAUSE sec
  NEXT i

  CLOSE #1
  PRINT num_line;" lines in ";str$(file_lp);".lp"
  FOR i=1 TO num_loop
     READ #2:start,end,xd,yd,xu,yu,xl,yl,xr,yr
     PRINT "#";i;": loop size = x1/x2,y1/y2 :";xl;"/";xr;", ";yd;"/";yu
  NEXT i
  CLOSE #1
  CLOSE #2
  END SUB
  !*****************************
  ! Subroutine: draw_line_array
  !*****************************
  SUB draw_line_array(a(,),dim_a1,sec,x2w,y2w) ! i(5)
     ! Read line coordinate array and plot it in 2D.
     ! a(,),dim_a1 : a(dim_a1,4). num of lines
     ! sec     : pause time between drawing lines
     ! 0,x2w,0,y2w : window size
  SET WINDOW 0,x2w,0,y2w     ! define window coordinate
  SET COLOR "yellow"
  SET BACK "red"
```

```
IF dim_a1=0 THEN
  PRINT "No line."
END IF

FOR i=1 TO dim_a1
   PLOT a(i,1),a(i,2);a(i,3),a(i,4)
   PAUSE sec
NEXT i
END SUB

!************************
! Subroutine: draw_point
!************************
SUB draw_point(name$,file_pt,sec,x2w,y2w)  ! i(5)
   ! Read *.pt and plot it in 2D
   ! name$ : object name
   ! file_pt : point coordinate file, *.pt
   ! sec     : pause time between drawing lines
   ! 0,x2w,0,y2w : window size
file_pt$="object\" & name$ & "\" & str$(file_pt) & ".pt"
CALL open_file(#1,file_pt$,error)
   IF error=1 THEN exit sub
SET WINDOW 0,x2w,0,y2w     ! define window coordinate
SET COLOR "yellow"
SET BACK "red"
ASK #1:FILESIZE record_num
LET point_num=record_num/2    ! compute number of points
FOR i=1 TO point_num
   PAUSE sec
   READ #1:x,y
   PLOT x,y
NEXT i
PRINT point_num;"points in ";str$(file_pt);".pt"
CLOSE #1
END SUB
!************************
! Subroutine: draw_bounds
!************************
SUB draw_bounds(name$,file_lpx,x2w,y2w)  ! i(4)
   ! Read loop index file, and draw all loop bounds with color
   ! name$     : object name
   ! file_lpx$ : loop index file, *.lpx
   ! 0,x2w,0,y2w : window size

file_lpx$="object\" & name$ & "\" & str$(file_lpx) & ".lpx"

CALL open_file(#1,file_lpx$,error)
   IF error=1 THEN exit sub
ASK #1:fILESIZE n
LET num_loop=n/10
SET COLOR "blue"
scale=0.05
dx=x2w*scale
dy=y2w*scale
FOR i=1 TO num_loop
   READ #1:start,end,xd,yd,xu,yu,xl,yl,xr,yr
   PLOT xl+dx,yl;xl,yl
```

```
   PLOT xr-dx,yr;xr,yr
   PLOT xd,yd+dy;xd,yd
   PLOT xu,yu-dy;xu,yu
 NEXT i
 END SUB
 !*******************************
 ! Subroutine: draw_auto_line_2d
 !*******************************
 SUB draw_auto_line_2d(name$,start,end,sec,x2w,y2w) ! i(6)
   ! Read from start.ln to end.ln, and plot it in 2D
   ! name$   : object name
   ! start,end: start and end num of line file, *.ln
   ! sec     : pause time between drawing lines
   ! 0,x2w,0,y2w : window size

 FOR i=start TO end
    CLEAR
    CALL draw_line(name$,i,sec,x2w,y2w)
    GET KEY getkey
 NEXT i
 END SUB
 !*******************************
 ! Subroutine: draw_auto_loop_2d
 !*******************************
 SUB draw_auto_loop_2d(name$,start,end,sec,x2w,y2w) ! i(6)

   ! Read from start.lp to end.lp, and plot it in 2D
   ! name$   : object name
   ! start,end: start and end num of loop file, *.lp
   ! sec     : pause time between drawing lines
   ! 0,x2w,0,y2w : window size
 FOR i=start TO end
    CLEAR
    CALL draw_loop(name$,i,sec,x2w,y2w)
    CALL draw_bounds(name$,i,x2w,y2w)
    GET KEY getkey
 NEXT i
 END SUB
 !*******************************
 ! Subroutine: draw_auto_point_2d
 !*******************************
 SUB draw_auto_point_2d(name$,start,end,sec,x2w,y2w) ! i(6)
   ! Read from start.pt to end.pt, and plot it in 2D
   ! name$   : object name
   ! start,end: start and end num of point file, *.pt
   ! sec     : pause time between drawing lines
   ! 0,x2w,0,y2w : window size
 FOR i=start TO end
    CLEAR
    CALL draw_point(name$,i,sec,x2w,y2w)
    GET KEY getkey
 NEXT i
 END SUB
```

```
!*******************************
! Subroutine: draw_auto_line_3d
!*******************************

SUB draw_auto_line_3d(name$,h_bot,h_top,dh,x2w,y2w,z2w)

  ! Read all line files (*.ln) and draw them in 3D
  ! h_bot,h_top: height of bottom and top of object
  ! dh       : slice thickness
  ! 0,x2w,0,y2w,0,z2w: 3D window size

CALL ParaWindow (0,x2w,0,y2w,0,z2w)   ! define window coordinate
SET BACK "red"
SET COLOR "white"
!CALL SetScale3(1.1,1.1)
CALL Axes3
CALL PlotText3(x2w,0,0,"X")
CALL PlotText3(0,y2w,0,"Y")
CALL PlotText3(0,0,z2w,"Z")
CALL PlotText3(0,0,0,"O")

SET COLOR "yellow"
LET num_slice=Int((h_top-h_bot)/dh)
LET num_file=num_slice+1     ! number of line files

FOR i=1 TO num_file          ! read a line file and draw line

  LET h=h_bot+(i-1)*dh     ! slice height
  LET file_line$="object\" & name$ & "\" & Str$(i) & ".ln"   ! line file name to read

  OPEN #1:name file_line$,access input    ! line file
  ASK #1:FILESIZE record_num

  LET line_num=record_num/4     ! compute number of lines

  FOR j=1 TO line_num      ! plot lines one at a time
    READ #1:x1,y1,x2,y2
    CALL LineOff3(x1,y1,h,x2,y2,h)
  NEXT j
  PAUSE pause_sec
  CLOSE #1
NEXT i
END SUB

!*************
! End of File
!*************
```

```
!************************************************************
! Written by Cheol Lee
!
! File: 72-gvt.tru
! Date: 10/95
!
! Object : draw *.vtx file
! Input  : vertex record file.  *.vtx
! Output : wireframe 3D drawing of an object
!************************************************************

library "67-1-vtx.tru"
library "67-ut.tru"
library "3dlib.trc"
end

EXTERNAL

!*********
! Summary (72-gvt)
!*********

! SUB draw_vtx_3d(name$,x2w,y2w,z2w)

!************************
! Subroutine: draw_vtx_3d
!************************

SUB draw_vtx_3d(name$,x2w,y2w,z2w)  ! i(4)

   ! Read vertex file and draw wireframe 3D plot
   ! name$ : object name  *.vtx
   ! 0,x2w,0,y2w,0,z2w: 3d window size

file_vtx$="object\" & name$ & ".vtx"

CALL open_file(#1,file_vtx$,error)
   IF error=1 THEN exit sub

ASK #1:FILESIZE num_rec
LET num_tri=num_rec/9

CALL ParaWindow(0,x2w,0,y2w,0,z2w)
SET BACK "red"

SET COLOR "white"
CALL Axes3
CALL PlotText3(x2w,0,0,"X")
CALL PlotText3(0,y2w,0,"Y")
CALL PlotText3(0,0,z2w,"Z")
CALL PlotText3(0,0,0,"O")

SET COLOR "yellow"

FOR i=1 to num_tri
  READ #1:x1,y1,z1,x2,y2,z2,x3,y3,z3
  CALL ploton3(x1,y1,z1)
```

```
   CALL ploton3(x2,y2,z2)
   CALL ploton3(x3,y3,z3)
   CALL plotoff3(x1,y1,z1)
NEXT i
PRINT num_tri;"triangles in ";name$;".vtx"
PRINT " object size x:";x2w;" y:";y2w;" z:";z2w
END SUB

!*************
! End of File
!*************
```

```
!*************************************************************************
! Written by Cheol H.Lee
! File: 72-main.tru
! Date: 6/96
! Object : main program to control SM2
! Input  : *.stl, job selection
! Output : driving machine
!*************************************************************************
! MEMO
! - *.stl, *.vtx files are in "object\" directory
! - *.ln,*.lp,*.lpx,*.pt,*.ptx,*.ct,*.ctx,*.st,ctreport.txt files are in
!   "object\[object_name]\" directory
! - setup.rp is in "object\" directory
! - rp-mt4.c is in "bc31\bin\" directory
! - rp-mt4.c will read *.st and ctreport.txt in "\object\temp\" directory
!*************************************************************************
LIBRARY "72-1-vtx.tru"
LIBRARY "72-2-ln1.tru"
LIBRARY "72-2-ln2.tru"
LIBRARY "72-3-lp2.tru"
LIBRARY "72-3-lp1.tru"
LIBRARY "72-4-pt.tru"
LIBRARY "72-5-ct2.tru"
LIBRARY "72-5-ct1.tru"
LIBRARY "72-6-st.tru"
LIBRARY "72-ut.tru"
LIBRARY "72-gvt.tru"
LIBRARY "72-gln.tru"
LIBRARY "72-gct.tru"
LIBRARY "72-menu.tru"
LIBRARY "3dlib.trc"
LIBRARY "utillib2.tru"
LIBRARY "utillib1.tru"
LIBRARY "shell.tru"
!LIBRARY "72-1-vtx.trc"
!LIBRARY "72-2-ln1.trc"
!LIBRARY "72-2-ln2.trc"
!LIBRARY "72-3-lp2.trc"
!LIBRARY "72-3-lp1.trc"
!LIBRARY "72-4-pt.trc"
!LIBRARY "72-5-ct2.trc"
!LIBRARY "72-5-ct1.trc"
!LIBRARY "72-6-st.trc"
!LIBRARY "72-ut.trc"
!LIBRARY "72-gvt.trc"
!LIBRARY "72-gln.trc"
!LIBRARY "72-gct.trc"
!LIBRARY "72-menu.tru"
!LIBRARY "3dlib.trc"
!LIBRARY "utillib2.trc"
!LIBRARY "utillib1.trc"
!LIBRARY "shell.tru"
DIM a(0,0),b(0,0),index(10,10)
x1s,x2s,y2s=800
y1s=120.63
!num_var=26
```

```
!**************
! Main Program
!**************
SET MODE "vga"
CALL read_setup (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip)
!---------------
! menu job loop
DO
MAT REDIM a(0,0),b(0,0)        ! save memory
!------
! menu
rwin=.8
IF xmax*rwin>ymax THEN  ! window size
  xwin=xmax
  ywin=xmax*rwin
  zwin=zmax
ELSE
  xwin=ymax/rwin
  ywin=ymax
  zwin=zmax
END IF
CALL status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip)
CALL main_menu (job)
CLEAR
SET COLOR "white"
!----------------------
! main job description
SELECT CASE job
CASE 0            ! quit
   STOP
CASE 1            !
   DO
!exit do ! remove dos shell for memory----------
     CLEAR
     PRINT "Press (0) to return to main menu."
     INPUT PROMPT "DOS command: ":com$
     IF com$="0" THEN
       EXIT DO
     ELSE
       CALL shell(com$)
     END IF
     GET KEY getkey
   LOOP
  !-----------
  ! job setup
CASE 2            ! change job setup
   DO
   !-----------------------------------------------------------
   ! setup menu
   CALL status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
       & xmax,ymax,zmax,xh,yh,decimal,skip)
   CALL setup_menu(job_setup)
   CLEAR
   SET COLOR "white"
```

```
SELECT CASE job_setup

  CASE 0          ! return to main menu
    EXIT DO

  CASE 1          ! foam thickness
    PRINT "current foam thickness: ";dh
    INPUT PROMPT "Foam thickness (cm)?  ":dh

  CASE 2          ! max cutter angle
    PRINT "current cutter angle: ";angle
    INPUT PROMPT "Maximum cutter angle (degree)?  ":angle

  CASE 3          ! bot space
    PRINT "current bottom space: ";space_b
    INPUT PROMPT "Bottom space from the foam (cm)?  ":space_b

  CASE 4          ! top space
    PRINT "current top space: ";space_t
    INPUT PROMPT "Top space from the foam (cm)?  ":space_t

  CASE 11         ! loop distance
    PRINT "current loop distance: ";d_loop
    INPUT PROMPT "Minimum distance between loops (cm)?  ":d_loop

  CASE 12         ! point distance
    PRINT "current point distance: ";d_point
    INPUT PROMPT "Maximum distance between points (cm)?  ":d_point

  CASE 13         ! skip
    PRINT "current skip points: ";skip
    INPUT PROMPT "Skip points?  ":skip

  CASE 14         ! decimal
    PRINT "current rount decimal: ";decimal
    INPUT PROMPT "Round decimal?  ":decimal

  CASE 21         ! home position
    PRINT "current home position : ";xh;yh
    INPUT PROMPT "Home position x,y (cm)?  ":xh,yh

  CASE 31         ! save setup
    CALL save_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip)
    EXIT DO

  CASE ELSE
    PRINT "Job is not available. Select again "
    GET KEY key
    EXIT SELECT
  END SELECT
  LOOP
!-------------------------------------------------------
! object

CASE 3            ! combine STL files
  CALL shell("dir/w/o:n object\*.stl")
```

```
    PRINT "Press (0,0,0) to cancel."
    INPUT PROMPT "stl_in1, stl_in2,Ostl_in3?  ":in1$,in2$,out$
    IF in1$="0" and in2$="0" and out$="0" THEN EXIT select
    PRINT "Wait..."
    CALL combine_stl(in1$,in2$,out$)

CASE 4              ! append STL files
    CALL shell("dir/w/o:n object\*.stl")
    PRINT "Press (0,0) to cancel."
    INPUT PROMPT "stl_inout, stl_in.":inout$,in$
    IF inout$="0" and in$="0" THEN EXIT select
    PRINT "Wait..."
    CALL append_stl(inout$,in$)

CASE 5              ! stl to object
    CALL shell("dir/w/o:n object\*.stl")
    PRINT "Press (0) to cancel."
    INPUT PROMPT "Object name?  ":name$
    IF name$="0" THEN exit select
    com$="md object\" & name$
    CALL shell(com$)
    CLEAR
    CALL stl_to_vtx(name$,num_tri,xmax,ymax,zmax)
    CLEAR
    CALL draw_vtx_3d(name$,xmax,ymax,zmax)
    GET KEY key

CASE 6              ! select object
    CALL shell("dir/w/o:n object\*.vtx")
    temp$=name$
    PRINT "Press (0) to cancel."
    INPUT PROMPT "Object name?  ":name$
    IF name$="0" THEN
      name$=temp$
      exit select
    END IF

    CALL object_size(name$,xmin,xmax,ymin,ymax,zmin,zmax,num_tri)

CASE 7          ! rotate object
    PRINT "Press (0,0,0) to cancel."
    PRINT "Current object is ";name$
    PRINT
    INPUT PROMPT "Rotated object name, axes, angle?  ":name_out$,rot_axes$,rot_angle
    IF rot_angle=0 THEN exit select

    CALL rotate_vtx(name$,rot_axes$,rot_angle,xmax,ymax,zmax,name_out$)
    name$=name_out$
    CLEAR
    CALL draw_vtx_3d(name$,xmax,ymax,zmax)
    GET KEY key

CASE 8          ! resize object
    PRINT "Press (0,0,0,0) to cancel."
    PRINT "To convert INCH into CM, scale=2.54."
    PRINT "Current object is ";name$
    PRINT
```

```
    INPUT PROMPT "Scaled object name, ratio(x,y,z)?  ":name_out$,xr,yr,zr
    IF xr=1 and yr=1 and zr=1 THEN exit select
    IF xr=0 and yr=0 and zr=0 THEN exit select

    CALL scale_vtx_file(name$,xr,yr,zr,xmax,ymax,zmax,name_out$)
    name$=name_out$

 CASE 9        ! registration
    INPUT PROMPT "Start, end file number?: ":start,end
    CLEAR
    CALL registration(name$,start,end,xwin,ywin)
 !---------
 ! a slice
 CASE 11       ! object to one line
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "Line file number, cut height?  ":file_ln,slice_h
    IF file_ln=0 and slice_h=0 THEN exit select
    CALL vtx_to_line_h(name$,slice_h,decimal,file_ln)
    CLEAR
    CALL draw_line(name$,file_ln,0,xmax,ymax)
    GET KEY getkey
 CASE 12       ! object to lines
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "start, end line files?  ":start,end
    IF start=0 THEN Exit select
    CALL vtx_to_line(name$,start,end,dh,decimal)
    GET KEY getkey
 CASE 13       ! lines to loops
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
    INPUT PROMPT "start, end loop files?  ":start,end
    IF end=0 THEN exit select
    CALL line_to_loop(name$,start,end,d_loop)
    GET KEY getkey

CASE 14       ! loop to point
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1
    INPUT PROMPT "start, end point files?  ":start,end
    IF end=0 THEN exit select
    CALL loop_to_point(name$,start,end,d_point,decimal,1)
    GET KEY key

CASE 15       ! point to cut
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
    INPUT PROMPT "start, end cut files?  ":start,end
    IF start=0 and end=0 THEN exit select
    reverse=0
    INPUT PROMPT "reverse(0/1)?  ":reverse

    CALL auto_cut(name$,start,end,reverse,dh,angle,decimal,skip)
    GET KEY getkey
```

```
CASE 16         ! cuts to steps
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
    INPUT PROMPT "start, end step files?  ":start,end
    IF start=0 and end=0 THEN exit select
    INPUT PROMPT "number of holes(0-n)?  ":num_hole

    CALL auto_step(name$,start,end,num_hole,dh,space_b,space_t,&
      & d_point,skip,xh,yh,x1s,x2s,y1s,y2s)
    GET KEY getkey

CASE 21         ! remove loop
   PRINT "Press (0) to cancel."
   PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
   INPUT PROMPT "loop file number?  ":file_lp
   IF file_lp=0 THEN exit select
   CLEAR
   CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)
   SET COLOR "yellow"
   PRINT
   PRINT " Press (0) to cancel."
   INPUT PROMPT " loop number to be removed?  ":lp_num
   IF lp_num=0 THEN exit select
   CALL remove_loop(name$,file_lp,lp_num)
   CLEAR
   CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)

CASE 22         ! reverse loop
   PRINT "Press (0) to cancel."
   PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
   INPUT PROMPT "loop file number?  ":file_lp
   IF file_lp=0 THEN exit select

   CLEAR
   CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)
   SET COLOR "yellow"
   PRINT "Press (0) to cancel."
   INPUT PROMPT "loop number to be reversed?  ":lp_num
   IF lp_num=0 THEN exit select
   CALL reverse_loop(name$,file_lp,lp_num)
   CLEAR
   CALL draw_auto_loop_2d(name$,file_lp,file_lp,0,xwin,ywin)

CASE 23         ! point to cut (SM1)
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
    INPUT PROMPT "start, end cut files?  ":start,end
    IF start=0 and end=0 THEN exit select
    CALL auto_cut_sm1(name$,start,end,decimal)
    GET KEY getkey
!------
! Read

CASE 41         ! read loop index file
   PRINT "Press (0,0) to cancel."
   PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
```

```
      INPUT PROMPT "start, end number?  ":start,end
      IF start=0 and end=0 THEN exit select
      CALL auto_read_lp_index(name$,start,end)

   CASE 42          ! read point index file
      PRINT "Press (0,0) to cancel."
      PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1

      INPUT PROMPT "start, end number?  ":start,end
      IF start=0 and end=0 THEN exit select
      CALL auto_read_pt_index(name$,start,end)

   CASE 43          ! read ct index file
      PRINT "Press (0,0) to cancel."
      PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1

      INPUT PROMPT "start, end number?  ":start,end
      IF start=0 and end=0 THEN exit select
      CALL auto_read_ct_index(name$,start,end)

   CASE 44          ! read coord file
      PRINT "Press (0) to cancel."
      PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
      INPUT PROMPT "Coordinate file name?  ":file_name$
      IF file_name$="0" THEN exit select

      file_name$="object\" & name$ & "\" & file_name$
      CALL read_record_file(file_name$,4)
      GET KEY getkey

   CASE 45          ! cut report
      PRINT "Press (0,0) to cancel."
      PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
      INPUT PROMPT "File number start, end?  ":start,end
      slice=int(zmax/dh)
      CALL cut_report(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
         & xmax,ymax,zmax,xh,yh,decimal,skip,slice,start,end)

   CASE 46          ! read report
      CALL read_cut_report(name$)
      PRINT
      INPUT PROMPT "print? (1/0)?  ":temp
      PRINT
      IF temp=1 THEN
        com$="print object\" & name$ & "\ctreport.txt"
        CALL shell(com$)
      END IF
!----------------
! automatic view
   CASE 51          ! view object - wire
      CLEAR
      CALL draw_vtx_3d(name$,xmax,ymax,zmax)
      GET KEY getkey
   CASE 52          ! view lines 3D
      PRINT "Press (0,0) to cancel."
      PRINT "object height:";zmax;",  foam thickness:";dh;", number of line file:";int(zmax/dh)+1
      INPUT PROMPT "start, end number?  ":start,end
```

```
    IF start=0 and end=0 THEN exit select
    h_bot=(start-1)*dh
    h_top=(end-1)*dh
    CLEAR
    CALL draw_auto_line_3d(name$,h_bot,h_top,dh,xmax,ymax,zmax)
    GET KEY getkey

CASE 53          ! view lines 2D
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foarm thickness:";dh;", number of line file:";int(zmax/dh)+1
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    CLEAR
    CALL draw_auto_line_2d(name$,start,end,0,xwin,ywin)

CASE 54          ! view loops
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of loop file:";int(zmax/dh)+1
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    CLEAR
    CALL draw_auto_loop_2d(name$,start,end,0,xwin,ywin)

CASE 55          ! view points
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of point file:";int(zmax/dh)+1
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    CLEAR
    CALL draw_auto_point_2d(name$,start,end,0,xwin,ywin)

CASE 56          ! view cuts
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of cut file:";int(zmax/dh)
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    CALL draw_auto_cut_2d(name$,start,end,dh,xwin,ywin)

CASE 57          ! view steps
    PRINT "Press (0,0) to cancel."
    PRINT "object height:";zmax;",  foam thickness:";dh;", number of step file:";int(zmax/dh)
    INPUT PROMPT "start, end number?  ":start,end
    IF start=0 and end=0 THEN exit select
    FOR i=start TO end
      CLEAR
      CALL draw_st_3d(name$,i,dh,space_b,space_t,xwin,ywin,x1s,x2s,y1s,y2s,xh,yh)
      GET KEY getkey
    NEXT i

CASE 91          ! prepare
    com$="copy object\" & name$ & "\*.st object\temp"
    CALL shell(com$)
    com$="copy object\" & name$ & "\*.txt object\temp"
    CALL shell(com$)

CASE 92          ! read report
    CALL read_cut_report("temp")
```

```
    GET KEY getkey

CASE 100          ! prototype
    CALL shell("c:\bc31\bin\rp-mt4")

CASE ELSE
    PRINT "Job is not available. Select again "
    GET KEY key
    EXIT SELECT

END SELECT
LOOP
END

!*************
! End of file
!*************
```

```
!*********************************************************************
! Written by Cheol H. Lee
! File: 72-menu.tru
! Date: 6/96
! Object : display menu, and read and save setup file
! Input  : setup.rp
! Output : dispaly menu, setup.rp
!*********************************************************************
! MENU
! - 15 jop setup parameters are save in object\setup.rp record file
!        (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point
!         xmax,ymax,zmax,xh,yh,decimal,skip)
! - Job setup parameters can be changed in setup menu.
!*********************************************************************
set mode "vga"
clear
a=1
if a=1 then
call main_menu (job)
else
call setup_menu(job)
end if
end
EXTERNAL

!*********
! Summary (72-menu)
!*********

! SUB status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
!         & xmax,ymax,zmax,xh,yh,decimal,skip)

! SUB main_menu(job)
! SUB setup_menu(job)

! SUB read_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
!         & xmax,ymax,zmax,xh,yh,decimal,skip)

! SUB save_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
!         & xmax,ymax,zmax,xh,yh,decimal,skip)

!*************************
! Subroutine: status
!*************************

SUB status (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
        & xmax,ymax,zmax,xh,yh,decimal,skip)   ! all input

   ! display status window in main menu window
   ! name$, num_tri  : object name. number of triangles
   ! dh, angle       : foam thickness. maximum cut angle
   ! space_t, space_b: space bw control point and top or bot surface
   ! d_loop, d_point : min distance bw loops.  discrete point distance
   ! x,y,zmax : object size
   ! xh,yh    : home position in x and y direction
   ! decimal, skip : round decimal. number of skip points.
```

```
ASK free memory mem
memory=round(mem/1000,1)

IF dh<>0 THEN num_slice=int(zmax/dh)

OPEN #9:SCREEN 0.66,1,0,1
  WINDOW #9
  CLEAR

  SET COLOR "red"
  PRINT "JOB SETUP "
  PRINT "**********************************"
  PRINT " Object ";name$;",";num_tri;"tri"
  PRINT "--------------------------------"
  PRINT " Slices     :";num_slice;"slices"
  PRINT " Free memory :";memory;"Kb"
  PRINT
  PRINT " Loop gab    :";d_loop;"cm"
  PRINT " Point res   :";d_point;"cm"
  PRINT " Point skip  :";skip;"pts"
  PRINT " Decimal     :";decimal;"place"
  PRINT
  PRINT " Cut angle   :";angle;"degree"
  PRINT " Foam        :";dh;"cm"
  PRINT " Bottom space:";space_b;"cm"
  PRINT " Top space   :";space_t;"cm"
  PRINT
  PRINT " home x,y    :";xh;yh
  PRINT
  PRINT " object x:";xmax;"cm"
  PRINT "        y:";ymax;"cm"
  PRINT "        z:";zmax;"cm"
END SUB

!************************
! Subroutine: main_menu
!************************

SUB main_menu (job)  ! i

  ! display main menu for job selection
  ! job : job selection

SET BACK "black"

OPEN #1:SCREEN 0,0.33,0.1,1
OPEN #2:SCREEN 0.33,0.66,0.1,1
OPEN #3:SCREEN 0,1,0,0.1
!-- 1st column 1111111111111111111111111111111111111111111111111
  WINDOW #1
  CLEAR
  SET COLOR "yellow"
  PRINT "RP CONTROL SOFTWARE R72"
  PRINT "by Cheol Lee "
  PRINT
```

```
SET COLOR "red"
PRINT "JOB LIST"
PRINT "*******************************"
SET COLOR "white"
PRINT " 0) Quit"
PRINT " 1) DOS"
PRINT " 2) Job setup"
PRINT " 3) Combine STL"
PRINT " 4) Append STL"
PRINT " 5) STL to object"
PRINT " 6) Select object"
PRINT " 7) Rotate object"
PRINT " 8) Resize object"
PRINT " 9) Registration"

SET COLOR "green"
PRINT "SLICE "
SET COLOR "white"
PRINT " 11) Object to a line "
PRINT " 12) Object to lines "
PRINT " 13) Lines to loops "
PRINT " 14) Loops to points "
PRINT " 15) Points to cuts"
PRINT " 16) Cuts to steps"
PRINT
PRINT " 21) Remove loop "
PRINT " 22) Reverse loop "
PRINT " 23) Points to cuts(SM1) "

!-- 2nd column

WINDOW #2
CLEAR
SET COLOR "magenta"
PRINT " Manufacturing Lab "
PRINT "  Univ. of Utah "
PRINT " "
BOX LINES 0,0.8,0.93,1

PRINT
PRINT
SET COLOR "green"
PRINT "READ "
SET COLOR "white"
PRINT " 41) Loop index"
PRINT " 42) Point index"
PRINT " 43) Cut index"
PRINT " 44) Coord file"
PRINT " 45) Create cut report"
PRINT " 46) Read cut report"

SET COLOR "green"
PRINT "AUTOMATIC VIEW"
SET COLOR "white"
PRINT " 51) Object"
PRINT " 52) Lines-3D"
PRINT " 53) Lines-2D"
```

```
  PRINT " 54) Loops"
  PRINT " 55) Points"
  PRINT " 56) Cuts"
  PRINT " 57) Steps"

  SET COLOR "green"
  PRINT "SHAPE MAKER 2"
  SET COLOR "white"
  PRINT "  91) prepare "
  PRINT "  92) check report"
  PRINT " 100) prototype"
!-- question window
  WINDOW #3
  CLEAR
  SET COLOR "white"
  INPUT PROMPT "Select a job:  ":job
CLEAR
END SUB

!*************************
! Subroutine: setup_menu
!*************************

SUB setup_menu(job)  ! i

  ! display setup sub menu
  ! job : job selection in setup sub menu

SET BACK "black"

OPEN #1:SCREEN 0,0.33,0.1,1
OPEN #2:SCREEN 0.33,0.66,0.1,1
OPEN #3:SCREEN 0,1,0,0.1

!-- 1st column 1111111111111111111111111111111111111111111111

  WINDOW #1
  CLEAR
  SET COLOR "yellow"
  PRINT "RP CONTROL SOFTWARE R67"
  PRINT "by Cheol Lee "
  PRINT

  SET COLOR "red"
  PRINT "JOB LIST"
  PRINT "*******************************"
  SET COLOR "white"
  PRINT " 0) Return to main menu"
  PRINT
  PRINT " 1) Foam thickness"
  PRINT " 2) Max Cut angle  "
  PRINT " 3) Bottom space  "
  PRINT " 4) Top space     "
  PRINT
  PRINT " 11) Min loop gab "
  PRINT " 12) Point resolution"
```

```
  PRINT " 13) Skip points "
  PRINT " 14) Decimal point "
  PRINT
  PRINT " 21) Home"
  PRINT
  PRINT " 31) Save setup"

!-- 2nd column

  WINDOW #2
  CLEAR
  SET COLOR "magenta"
  PRINT " Manufacturing Lab "
  PRINT "   Univ. of Utah "
  PRINT " "
  BOX LINES 0,0.8,0.93,1
  SET COLOR "white"

!-- question window
  WINDOW #3
  CLEAR
  SET COLOR "white"
  INPUT PROMPT "Select a job:  ":job

CLEAR
END SUB

!*************************
! Subroutine: read_setup
!*************************

SUB read_setup (name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
        & xmax,ymax,zmax,xh,yh,decimal,skip)  ! all input

   ! read setup parameters in setup.rp file
   ! name$, num_tri  : object name. number of triangles
   ! dh, angle       : foam thickness. maximum cut angle
   ! space_t, space_b: space bw control point and top or bot surface
   ! d_loop, d_point : min distance bw loops.  discrete point distance
   ! x,y,zmax : object size
   ! xh,yh    : home position in x and y direction
   ! decimal, skip : round decimal. number of skip points.

OPEN #1:name "object\setup.rp",create newold,access outin, org record

RESET #1:Begin
WHEN error IN
   READ #1:name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point
   READ #1:xmax,ymax,zmax
   READ #1:xh,yh,decimal,skip
USE
   exit sub
END WHEN
CLOSE #1
END SUB
```

```
!************************
! Subroutine: save_setup
!************************

SUB save_setup(name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point,&
      & xmax,ymax,zmax,xh,yh,decimal,skip)  ! all input

  ! save current setup parameters in setup.rp file
  ! name$, num_tri  : object name. number of triangles
  ! dh, angle       : foam thickness. maximum cut angle
  ! space_t, space_b: space bw control point and top or bot surface
  ! d_loop, d_point : min distance bw loops.  discrete point distance
  ! x,y,zmax : object size
  ! xh,yh    : home position in x and y direction
  ! decimal, skip : round decimal. number of skip points.

OPEN #1:name "object\setup.rp",create newold,access outin, org record
ERASE #1
SET #1:RECSIZE 10
RESET #1:Begin
WRITE #1:name$,num_tri,dh,angle,space_t,space_b,d_loop,d_point
WRITE #1:xmax,ymax,zmax
WRITE #1:xh,yh,decimal,skip
END SUB

!*************
! End of File
!*************
```

CLAIMS

What is claimed is:

1. In a rapid prototype system wherein an electronic model of an object is electronically decomposed into a first series of electronic layers, a second series of physical layers is generated from a construction material, said second series corresponding layer by layer to said first series, and a physical prototype of that object is then constructed by assembling said physical layers, the improvement which comprises:

a. electronically decomposing said object into said first series of electronic layers in accordance with a paradigm characterized by a higher than zero order fit with respect to the surface of said object; and

b. generating said second series of physical layers from said construction material by mechanical means constructed and arranged to operate in accordance with said paradigm.

2. An improvement according to Claim 1 wherein said paradigm has a higher order of fit than that of the vertical cylinder paradigm (VCP).

3. An improvement according to Claim 2, wherein said paradigm is selected from either the trapezoidal cylinder paradigm (TCP) or the arc cylinder paradigm (ACP).

4. An improvement according to Claim 1, wherein said steps a and b are conducted by means controlled by software equivalent to that set forth on pages 14 through 117.

5. A rapid prototype system, comprising:

electronically decomposing an electronic model of an object into a first series of electronic layers in accordance with a paradigm characterized by a higher than zero order fit with respect to the surface of said object;

generating a second series of physical layers from a construction material by mechanical means constructed and arranged to operate in accordance with

said paradigm, said second series corresponding layer by layer to said first series; and

constructing a physical prototype of said object by assembling said physical layers.

6. A rapid prototype system according to Claim 5, wherein said mechanical means is structured and arranged to provide four degrees of freedom in positioning a cutting device with respect to sheets of said construction material.

7. A rapid prototype system according to Claim 6, wherein said mechanical means is constructed and arranged to accept and manipulate sheets of construction material 1.22 meters (four feet) wide.

8. A rapid prototype system according to Claim 6, wherein said mechanical means is operated by control means responsive to software equivalent to that set forth at pages 14 through 117.

9. A rapid prototype system according to Claim 6, wherein said mechanical means includes:

a support structure, operable to move a sheet of construction material forwards and backwards along a travel path in a plane;

an electronically heated wire with a first end held by a first linear positioning device and a second end held by a second linear positioning device, said first and second positioning devices being supported to position said wire transverse said plane to intersect and cut a sheet of construction material traveling on said support structure;

first structure, mounting said first positioning device such that it may travel back and forth transverse said travel path;

second structure, mounting said second positioning device such that it may travel independently of said first positioning device back and forth transverse said travel path; and

electronic control means, responsive to operating software, for controlling the relative movements of a sheet of construction material on said support structure, said first positioning means and said second positioning means.

10. A rapid prototype system according to Claim 9, including heat control means, responsive to said operating software, constructed and arranged to adjust the temperature of said wire.

11. A rapid prototype system according to Claim 9, wherein said operating software is equivalent to that set forth at pages 14 through 117.

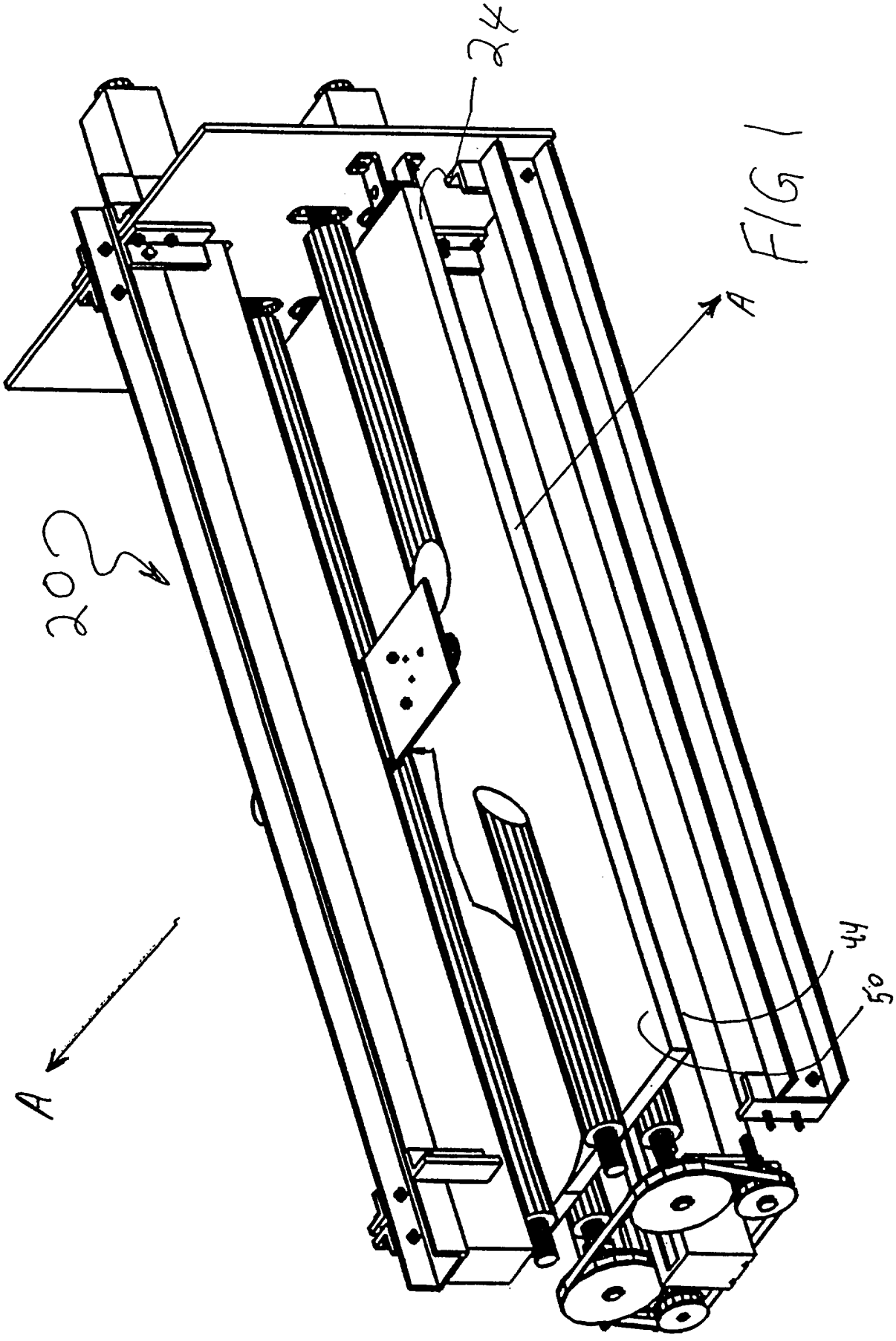


FIG 1

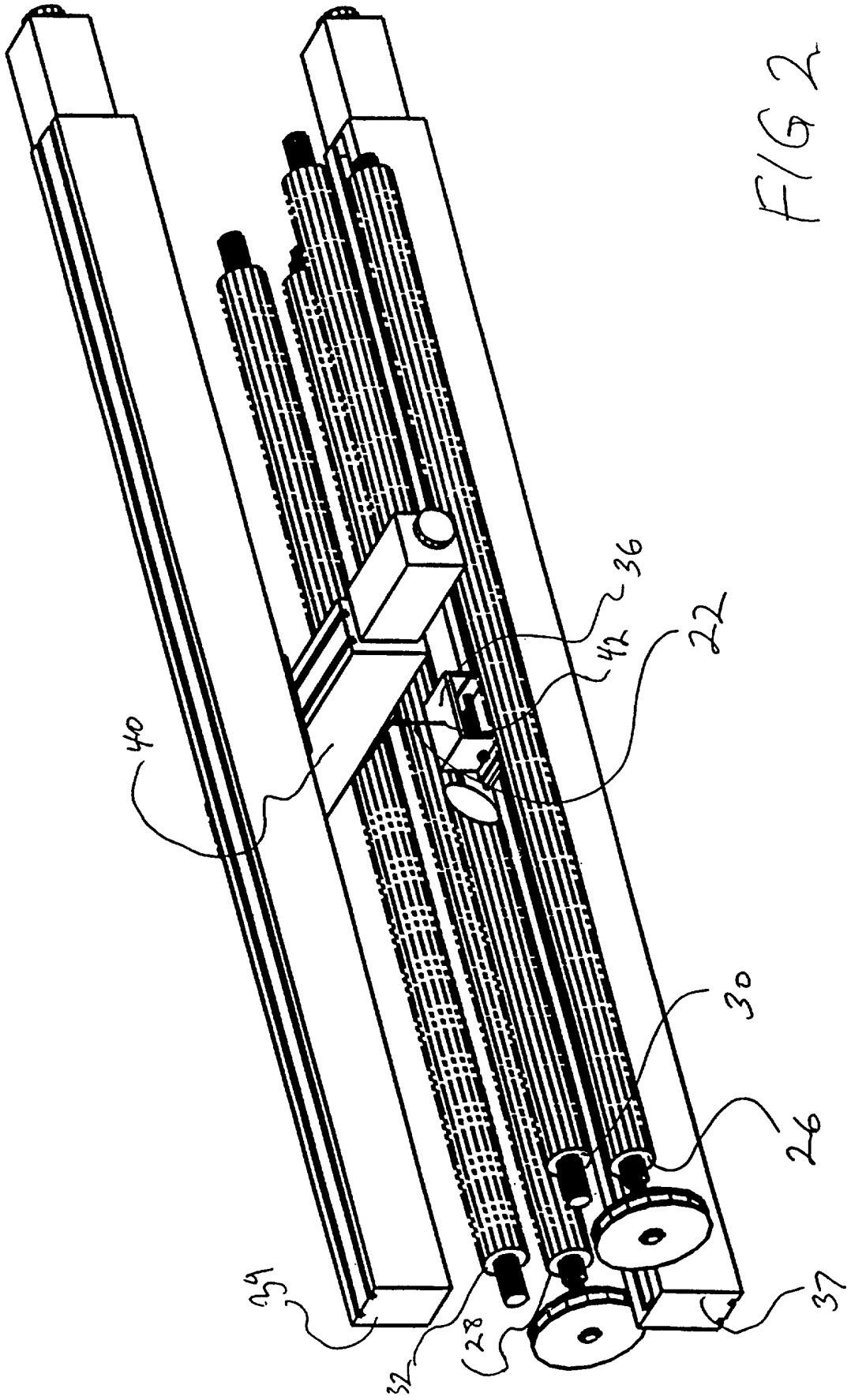


FIG 2

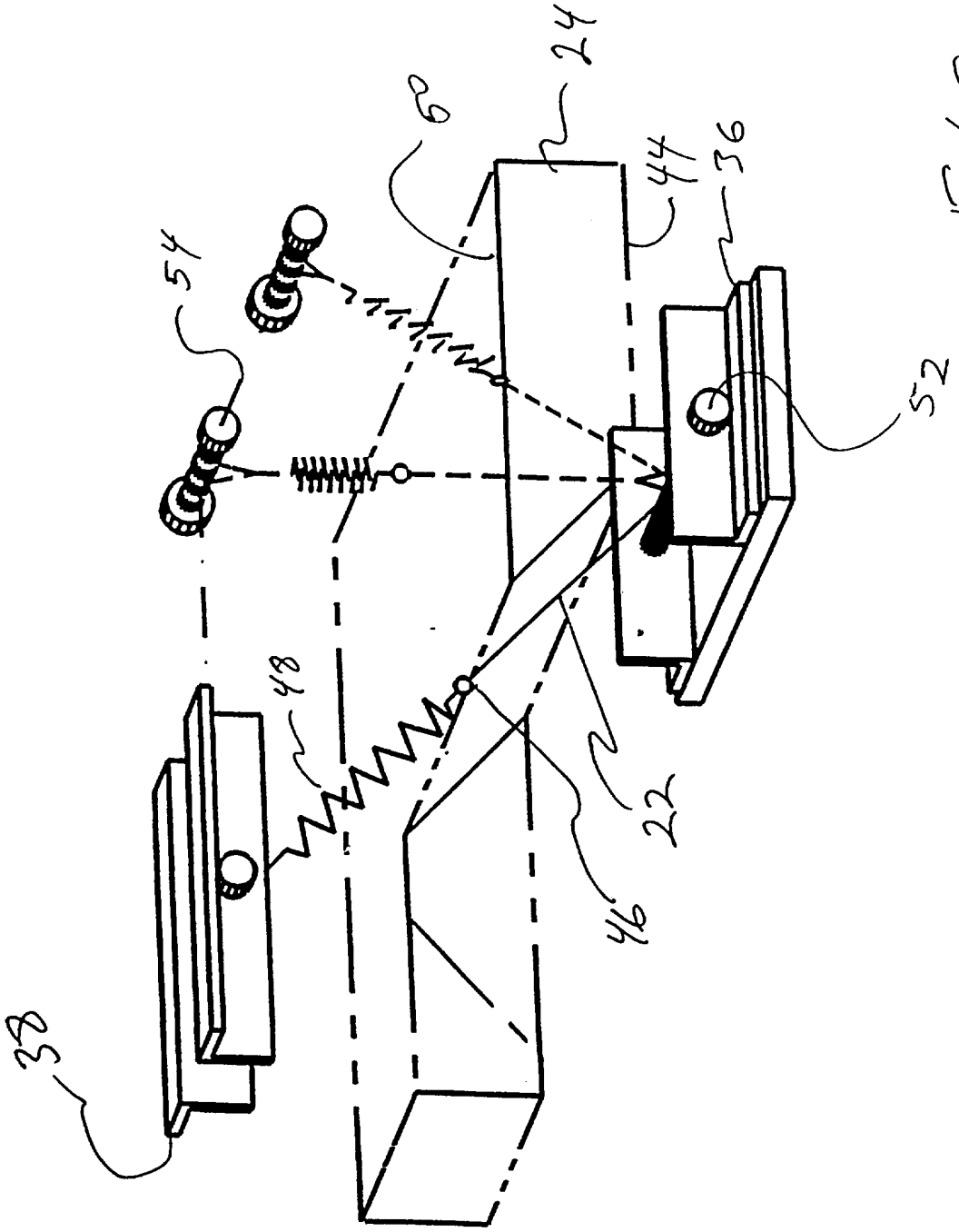


FIG 3

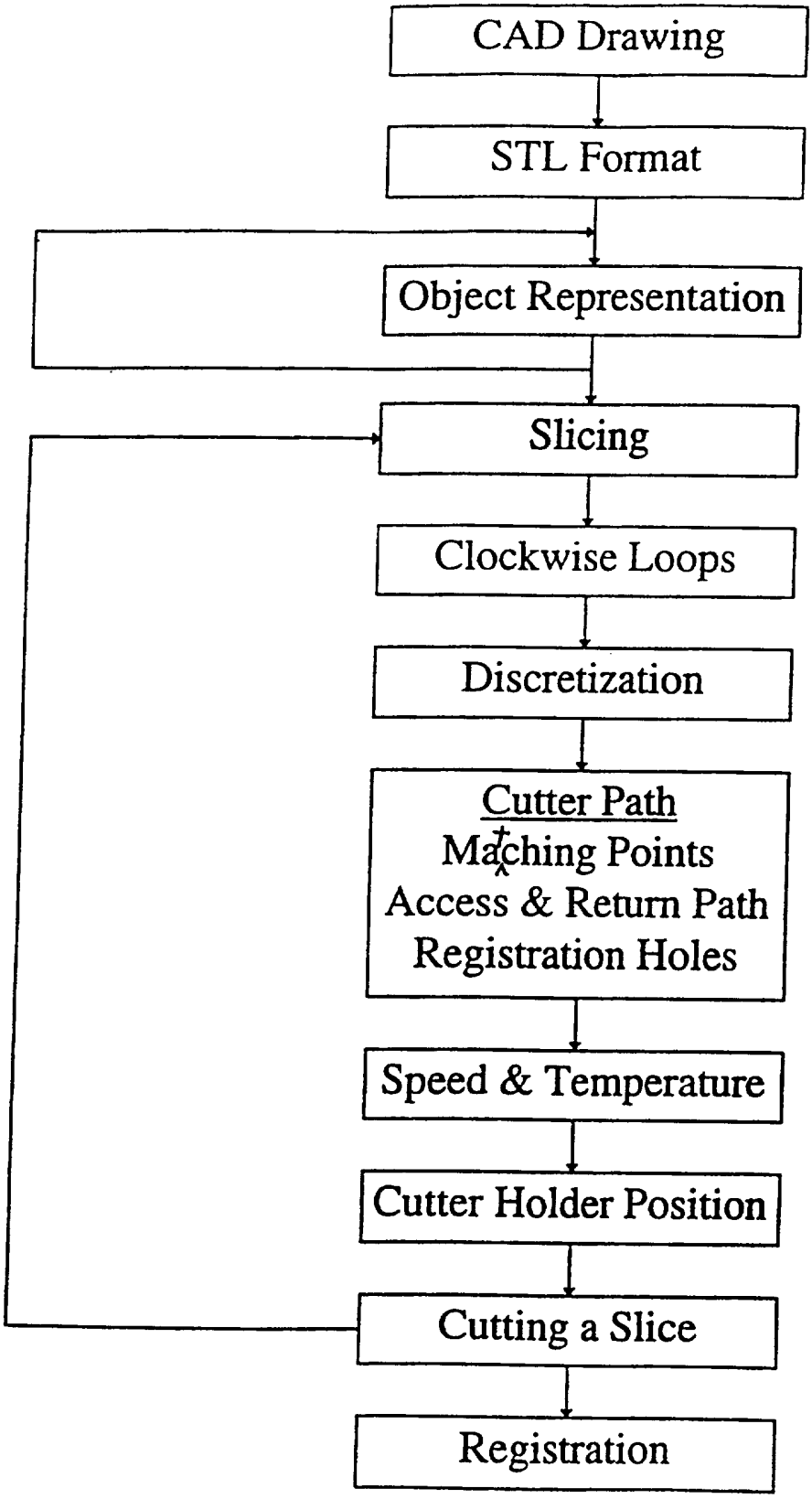

CAD Drawing
STL Format
Object Representation
Slicing
Clockwise Loops
Discretization
Cutter Path
Matching Points
Access & Return Path
Registration Holes
Speed & Temperature
Cutter Holder Position
Cutting a Slice
Registration

FIG 4

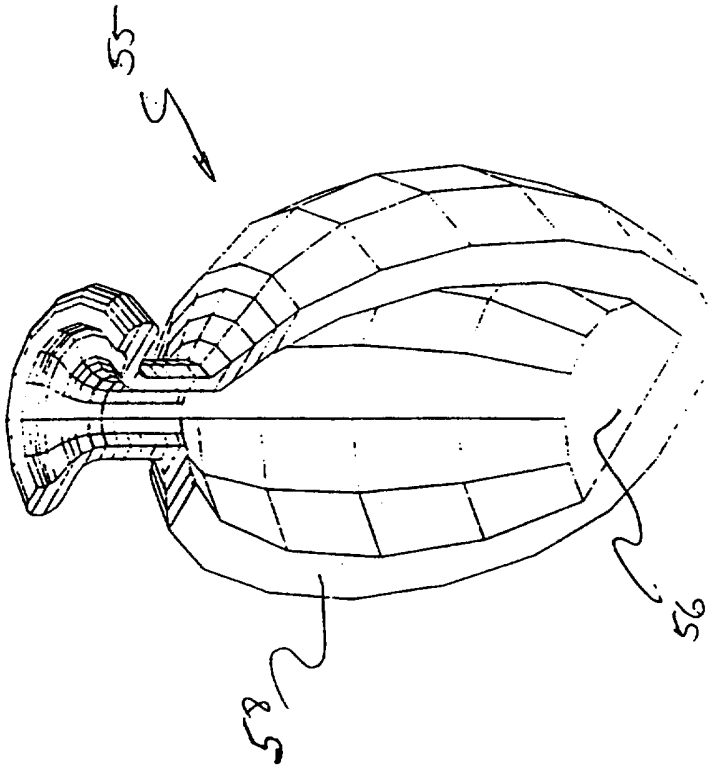


FIG. 6

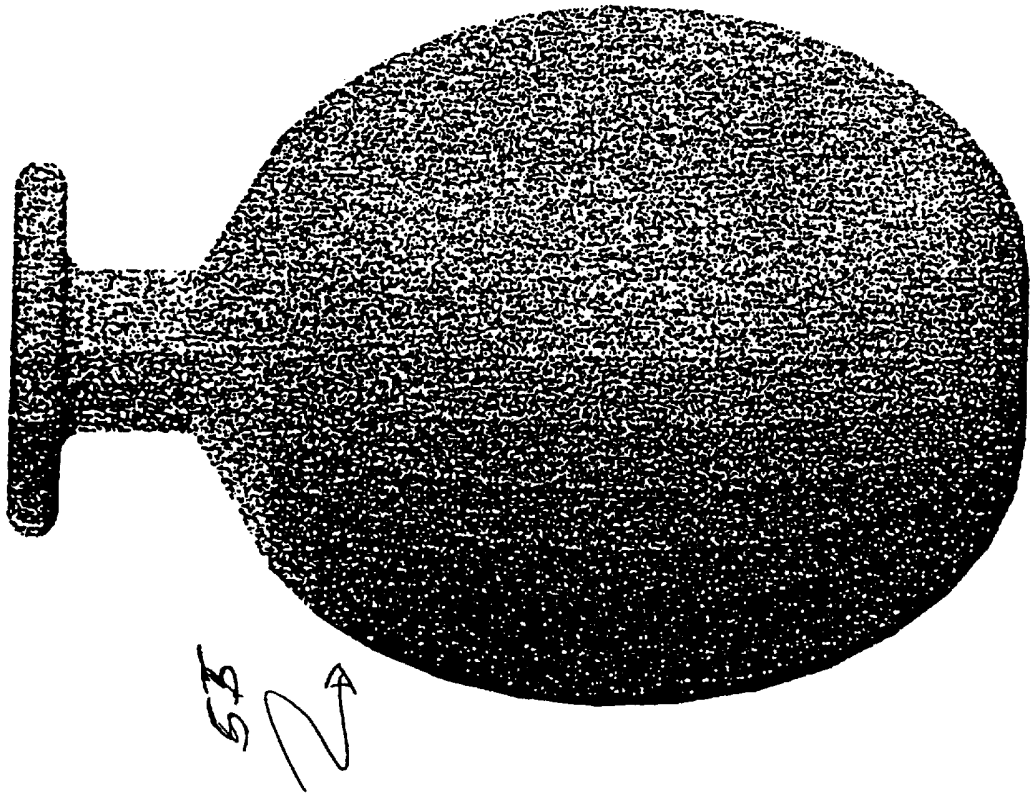


FIG. 5

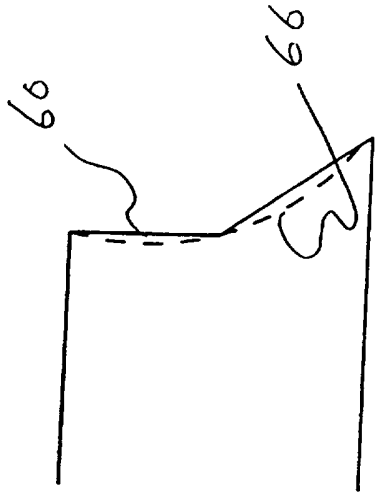


FIG. 9

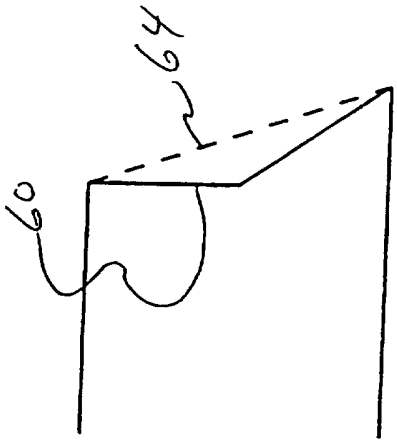


FIG. 8

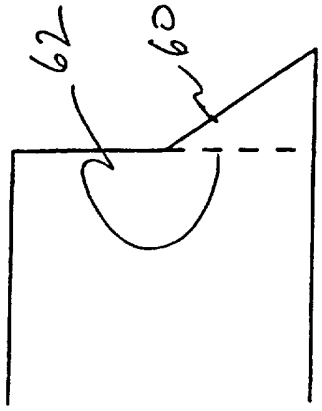


FIG. 7

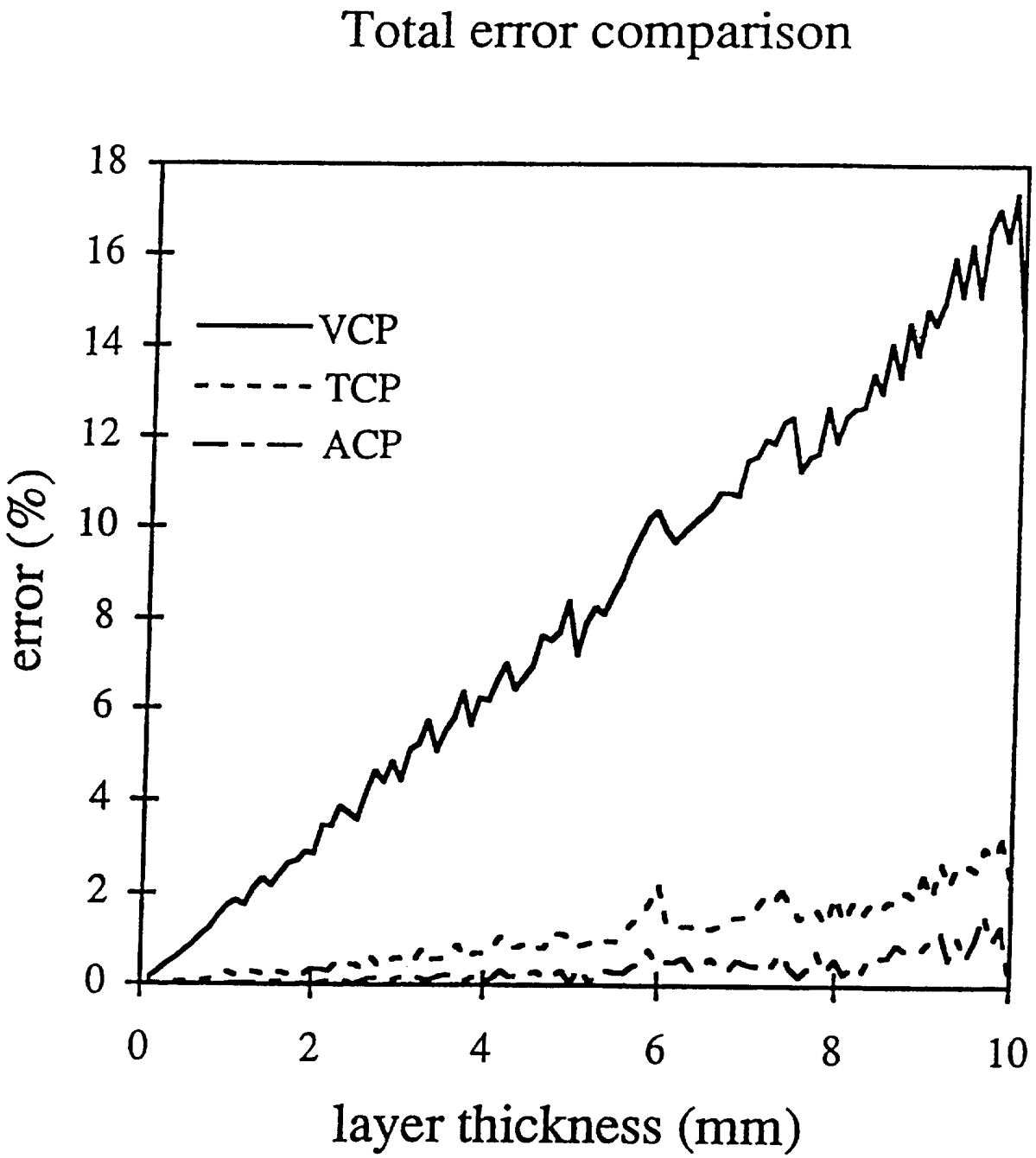

Total error comparison
18
16
14
12
10
8
6
4
2
0
error (%)
VCP
TCP
ACP
0
2
4
6
8
10
layer thickness (mm)

FIG 10

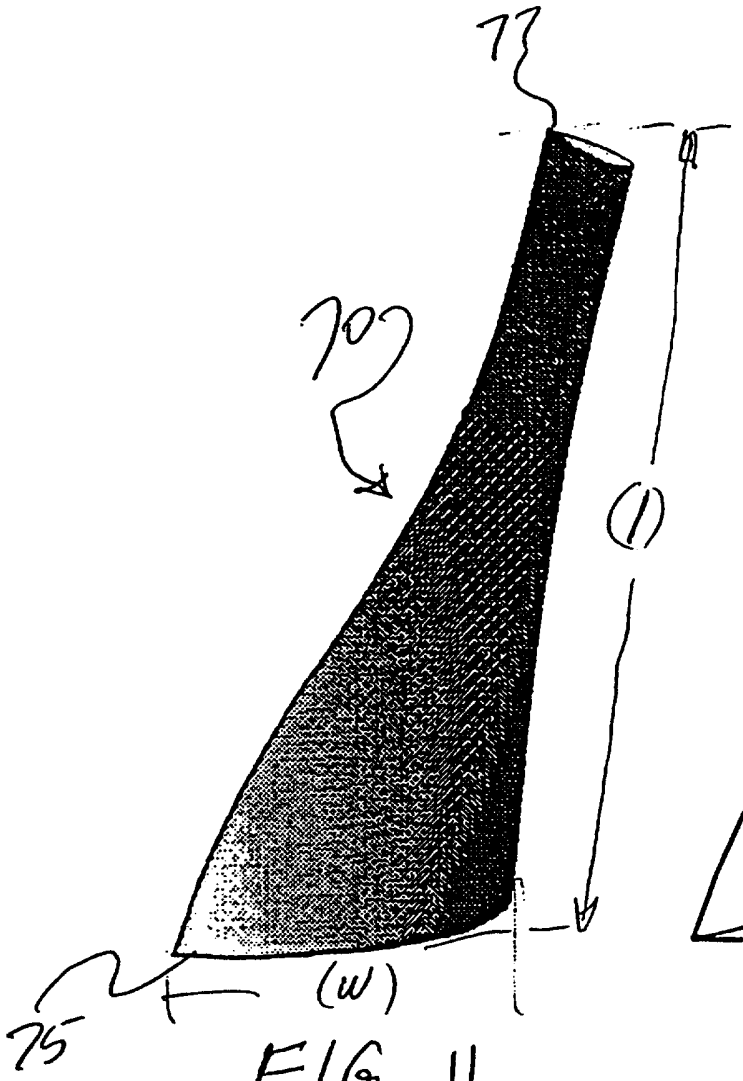


FIG. 11

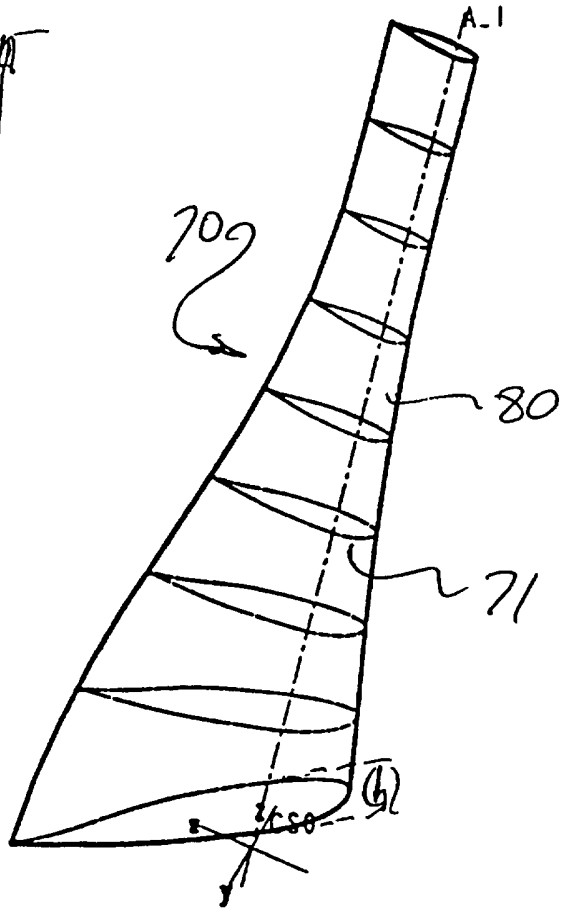


FIG 12

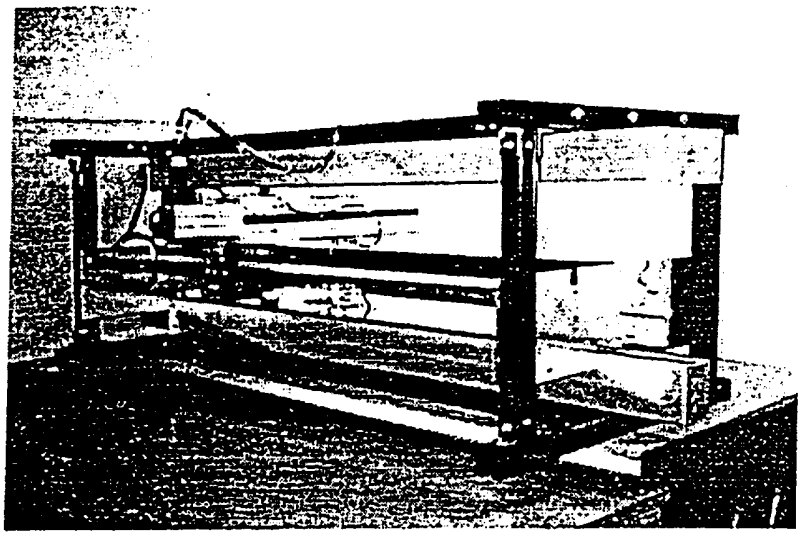


FIG. 13

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/05419

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :G06F 19/00
US CL :364/468.26
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 364/468.26, 468.25

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,287,435 A (COHEN et al) 15 Febuary 1994, figure 7	1-11
A	US 4,665,492 A (MASTERS) 12 May 1987, figure 1	1-11
A	US, 5,398,193 A (DEANGELIS) 14 March 1995, figures 1a, and 2	1-11
A	US 5,432,704 A (VOUZELAUD et al) 11 July 1995, figure 6	1-11

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 21 MAY 1997	Date of mailing of the international search report 09 JUN 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer JAMES PAUL TRAMMELL Telephone No. (703)-305-3800

Form PCT/ISA/210 (second sheet)(July 1992)★