PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 15/72	A1	(11) International Publication Number: WO 92/08201 (43) International Publication Date: 14 May 1992 (14.05.92)

(21) International Application Number: PCT/US90/06274

(22) International Filing Date: 29 October 1990 (29.10.90)

(71)(72) Applicants and Inventors: HOROWITZ, Steven, L. [US/US]; 114 Ricardo Avenue, Piedmont, CA 94611 (US). LING, Marvin, T. [US/JP]; Takanawa Homes, Apt. 4A, 2-1-51, Takanawa, Minato-ku, Tokyo 108 (JP).

(74) Agents: HOFFMAN, Charles, R. et al.; Cahill, Sutton & Thomas, 2141 East Highland Avenue, Suite 155, Phoenix, AZ 85016 (US).

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

Published
With international search report.
With amended claims.

(54) Title: AUTOMATIC DRAWING SYSTEM

(57) Abstract

A technique for automatically producing drawings includes hand-drawing a group of similar images of a first type within an active area of a first sheet, scanning it with a scanner to produce corresponding runlength data, operating on the runlength data to form software objects representing the images of the first group, and storing the first software objects in a first software layer. A second group of images of a second type hand-drawn on a second sheet, which is scanned to produce corresponding runlength data which is then operated upon to form software objects representing the images of the second group and storing the second software objects in a second software layer. Data from the first and second software layers are operated upon to plot a first composite drawing including the images of both the first and second groups. Additional groups of similar images are drawn on additional sheets to obtain additional software layers and additional composite drawings. Global commands handwritten on sheets set system and sheet parameters. Editing directives are handwritten and hand-drawn within the active areas of additional sheets to modify and augment the previous software layers until a final, edited, complex composite drawing is obtained.

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.

AT	Austria	ES	Spain	MG	Madagascar
AU	Australia	FI	Finland	ML	Mali
BB	Barbados	FR	France	MN	Mongolia
BE	Belgium	GA	Gabon	MR	Mauritania
BF	Burkina Faso	GB	United Kingdom	MW	Malawi
BG	Bulgaria	GN	Guinea	NL	Netherlands
BJ	Benin	GR	Greece	NO	Norway
BR	Brazil	HU	Hungary	PL	Poland
CA	Canada	IT	Italy	RO	Romania
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland			SN	Senegal
CI	Côte d'Ivoire	KR	Republic of Korea	SU+	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
CS	Czechoslovakia	LK	Sri Lanka	TG	Togo
DE*	Germany	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		

+ Any designation of "SU" has effect in the Russian Federation. It is not yet known whether any such designation has effect in other States of the former Soviet Union.

AUTOMATIC DRAWING SYSTEM

BACKGROUND OF THE INVENTION

The invention relates to computerized systems which can scan hand-drawn drawings using an optical scanner, recognize text and lines, points on arcs, circles, curves and shapes, and recognize certain hand-drawn global commands, local commands, and spatial indicators on a "layer-by-layer" basis, and can generate progressively complete "mathematically accurate" composite drawings by means of a printer or plotter and a corresponding database.

A variety of computerized drafting aids are presently available. For example, the well-known program AUTOCAD can run on IBM-XT and IBM-AT computers and also can run on various IBM and Sun minicomputers and larger computers such as the Digital Corporation VAX systems. The AUTOCAD system is very complex and expensive, costing from roughly $2,000 for the least expensive version to over $10,000 for the most expensive versions.

The AUTOCAD system and other automatic computerized drafting aids generally are difficult to learn to use efficiently. Using a computer monitor as required by AUTOCAD requires panning and zooming operations due to the inherent handicaps of poor resolution and small size of the monitor screen. This requirement impedes operation, since the operator must first pan to locate the position of a drawing on which he is working, and then enlarge the display by "zooming in" to see the detail necessary to continue. Consequently, many draftsmen make little or no use of computerized drafting aids. Roughly three-fourths of all engineering and mechanical drawings and blueprints are completely hand-drawn using traditional drafting tables, rulers, templates, and articulated drafting machines.

A prior art system sold under the trademark TOSGRAPH by Toshiba Corporation of Japan converts hand-sketched drawings of logic diagrams into perfectly shaped logic diagrams suitable for input into a CAD database which can be utilized by analysis programs to perform a logic analysis on the sketched logic diagram. However, the TOSGRAPH system is very limited in its capabilities because it is intended only for logic diagrams. It therefore always forces all lines to be horizontal or vertical, and constrains the system such that all lines must be connected to logic symbols such as AND/OR gates at specific positions, even if the sketched lines do not touch the symbol. The TOSGRAPH system does not recognize any symbols other than standard logic diagram symbols, and cannot draw circles or arcs.

A system called the GTX 5000 marketed by GTX Corporation of Phoenix, Arizona is capable of scanning a drawing, rapidly vectorizing and recognizing drawing entities and text, respectively, and providing a representative database thereof. Since the system must scan and convert existing drawings, there is no means to enhance accuracy by encoding and processing drawing entities on a layer-by-layer basis. Thus the resulting database contains many mistakes, especially where objects are filled or overlap, that require time-consuming corrections. A typical computer workstation is needed to edit the drawing, and a substantial amount of user training and effort are necessary to operate the editor.

U.S. patent 4,058,849 discloses automatic dimensioning of drawings entered by means of a digitizing tablet. The tablet contains a drawing area and a menu area with command buttons and alphanumeric keys. Lines and circles are specified by picking the appropriate commands and locating points in the drawing area. Dimensions and text are entered via the keys after indicating their position and orientation. The system does not permit arc, curve, or shape entry. Dimension rectification is accomplished after all entities have been defined and before output of the finished drawing. The patent concerns itself mainly with the algorithm for adjusting the proportions of the drawing entities by traversing a data structure containing the input dimensioning information.

Thus, there remains an unmet but long existing need for a reasonably priced, easily learned, and easily used computerized drafting aid which substantially reduces the effort required for the draftsman to make "mathematically accurate" engineering drawings and the like in which hand sketched "straight" lines are corrected to be perfectly straight and hand drawn circles and arcs are corrected to be perfectly circular, etc.

SUMMARY OF THE INVENTION

Accordingly, it is an object of the invention to provide a reasonably low-cost computerized drafting aid, the operation of which is easily learned by an average drafsman.

It is another object of the invention to provide a computerized drafting aid which rapidly converts hand-drawn sketches into "mathematically accurate" engineering drawings or the like.

It is another object of the invention to provide a computerized drafting aid which allows a draftsman to continue using practiced, natural drawing skills to create a digital drawing database.

It is another object of the invention to provide a computerized drafting aid which requires a draftsman to depress a minimum number of buttons, switches, and/or keys in order to utilize the system.

It is another object of the invention to provide a computerized drafting aid which requires a minimum number of hand-drawn strokes without the use of specialized mechanical drafting tools to generate a detailed drawing.

It is another object of the invention to provide a computerized drafting aid which allows efficient editing of previously digitized drafted drawings or parts thereof.

Briefly described, and in accordance with one embodiment thereof, the invention provides an automatic drawing system including a scanner or digitizing tablet and a printer or plotter. The most "basic" embodiment of the invention can be operated without a keyboard, using only a few control buttons and a few indicators. Semi-transparent grid paper with grid lines and designated global command areas which are not incorporated in the drawing allow a draftsman to make hand-sketched, fairly accurately positioned lines, circles, arcs, etc. The scanner detects pixels and serially transmits them to a recognition processor that recognizes imperfectly drawn straight lines, eliminates meaningless deviations, and stores mathematically accurate straight line, arc, circle, etc. data. Such data then is used to drive the printer to print a mathematically accurate replica of a hand-drawn sketch and also produce a database thereof on a floppy diskette or the like. Certain areas of the scanned document are reserved for recognizing certain symbols which are not interpreted as part of the drawing but as global commands. The global commands include, but are not limited to, layer numbers, line thicknesses, scale ratios, "clean up", and other commands. The system also recognizes local commands which are hand-drawn or handwritten near or inside of hand-drawn shapes to which such local commands specifically pertain. Local commands also are automatically recognized as command symbols, not as part of the drawing. The local commands include, but are not limited to, mirroring, rotating, copying, "generate or call up symbol", hatching, "repetitive pattern" and other commands. A complex drawing can be input to the automatic drawing system in "layers", in order to enable a draftsman to, for example, draw a large number of similar type entities, i.e., lines, circles, text, etc. on separate sheets, respectively. The recognition module or processor which interprets the scanned pixel data generates a "composite" drawing of all previous layers and prints it out in response to a print command generated by pressing one of the few control buttons of the system. The burden upon the recognition module in recognizing the entities on a particular layer is greatly reduced by having information that all entities on that layer are similar or of a particular type or types. After a new composite drawing is printed out, the draftsman then can obtain a clean sheet of the semi-transparent grid paper, register it with the most up-to-date composite sheet that has been printed out, and add more detail on yet another layer. Certain shapes that are repeated many times can be stored as library symbols and identified and marked on the present layer along with an accurate "position marker". When that layer is scanned by the scanner, the recognition module recognizes the symbol and calls up the symbol library, locates it properly on the drawing, and merges that information into the composite database.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of the automatic drafting system of the present invention.

Fig. 1A is a plan view illustrating the control buttons and indicators of the central processor contained in the housing of the scanner of Fig. 1.

Fig. 1B is a diagram illustrating various combinations implementing the present invention.

Fig. 2 is a plan view of a piece of semi-transparent grid paper which can faciliate use of the automatic drafting system of the present invention.

Figs. 3A-3B constitute a flowchart useful in describing the basic method of using the automatic drafting system of Fig. 1.

Figs. 4A-4B constitute a flow chart of an initialization routine executed by the processor of Fig. 1.

Figs. 5A-5D constitute a flow chart of a routine called by the routine of Figs. 4A-B.

Figs. 6A-6D constitute a routine called by the routine of Figs. 5A-D.

Figs. 7A-7H constitute a flow chart of a program called by the routine of Figs. 5A-5D.

Figs. 8A-8D constitute a flow chart of a program called by the routine of Figs. 4A-B to print a drawing.

Fig. 9 is a diagram useful in explaining active areas and layers.

Figs. 10A-10D are diagrams useful in describing various entity types.

Fig. 11 is a diagram useful in describing edit layers.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT OF THE INVENTION

The invention provides a device to enable draftsmen to accomplish their work more efficiently by minimizing the number of "strokes" required to make and/or edit a drawing and to minimize the difficulty of such strokes, and provide the output in the form of a mathematically accurate drawing and also in the form of a database containing all of the information necessary to input the drawing to a CAD/CAM system or the like.

Fig. 1 shows the basic components of the automatic drawing system 1, including a scanner 2, with a cover 16A, a processor 8 and a printer 7. Fig. 1A shows processor 8 alone with its buttons 3A-G and display lights 5A-C. Fig. 1B shows various other ways of implementing a drafting aid by using different components. A stand alone scanner 2A or a stand alone digitizer tablet 2B can provide graphic input data to either a stand alone processor 8 or a PC workstation computer 17 having a suitable "add-on board" containing a suitable recognition module. The output of the PC workstation 17 containing a suitable recognition module or of stand alone processor 8 containing a suitable recognition module can be provided to a suitable commercially available printer 7 or plotter 7A. Processor 8 can drive either a printer 7 or a plotter 7A to produce mathematically accurate composite layers created by scanning hand-drawn documents by means of scanner 2 or digitizer tablet 2A. The scanner 2 can, for example, be size A or size B flatbed type scanner with a resolution of 200 to 400 dots per inch or more. Smaller drawings can be scanned by scanner 2 and recognized as sections of a larger drawing, so sections of a larger drawing can be merged to produce a data base of the larger drawing.

The processor 8 has been adapted to include a plurality of control buttons 3, a floppy disk drive 4, and a plurality of indicators 5. Scanner 2 outputs serial pixel "runlength" data resulting from line-by-line scanning of a hand-drawn document.

The controls for the central processor are designed to be very simple, operating in response to several buttons and providing only a few indicators.

Referring to Fig. 1A, the power on-off button 3A simply turns system power on or off. "Reset" button 3B is depressed to initialize processor 8 for normal system operation and to load the software program from a diskette inserted into the floppy disk drive 4. "Load" button 3C causes the processor 8 to load the "drawing data" from another diskette inserted into disk drive 4. "Scan" button 3D causes the scanner 2 to scan a hand-drawn document that has been loaded therein. The database containing the data representation of the mathematically accurate drawing is updated onto the drawing data diskette. "Draw" button 3E causes the scanner unit 2 to drive printer 7 to generate a drawing which is a "mathematically accurate" regeneration of the original hand-drawn sketch. "Report" button 3F when depressed causes a report of a variety of system information to be printed. "Stop" button 3G when depressed causes the current operation to be halted.

"Ready" indicator 5A indicates when the user can begin scanning, drawing, reporting or executing disk I/O (Input/Output). "Busy" indicator 5B indicates that the processor is effectuating one of the foregoing operations. "Fault" indicator 5C indicates a system error by a two-digit code indicating the type of error, such as defective floppy disks, printer not ready, out of memory, etc.

The floppy disk drive 4 allows loading of system software, loading and storing tables of line styles, fill styles, and symbols, and inputting and outputting of a CAD database representing the regenerated hand-drawn sketch or a part of it.

Fig. 2 designates a sheet of semi-transparent grid paper in which horizontal grid lines 11, vertical grid lines 12, and a global command area 14 are printed in (for example) light pink or colors which are not detected by scanner 2, so that only the handwritten commands and hand-sketched drawing will be scanned and produce pixel data.

In accordance with the present invention, use of the grid paper 10 allows the draftsman to position endpoints of lines, arcs, etc. and draw fairly accurate straight lines, arcs, etc. much more rapidly than could be accomplished using scale rulers, compasses, french curves, templates, and the like. Global commands, subsequently described, are contained in designated "inactive" area 14 at the top or bottom of the grid paper sheet 10. The "X's" 13 or punched holes 13A are used to align subsequent "sheets" to previous composite printouts.

Various "global commands" can be hand-written into global command area 14 of each drawing data sheet or edit sheet. These global commands apply to the entire sheet in those two cases, unless overriden by a local command written into the active area of an edit sheet. The global commands include, for example, a "sheet type" command, a "layer type" command, a "layer number" command, an "entity type" command, a "position" command, a "scale" command, an "orientation" command, and various "entity attribute" commands.

The subsequently described "layering" technique allows the draftsman to draw the same types of entities of an overall drawing all at the same time on a particular "layer". (See Fig. 9.) The layer numbers included in the composite drawing, and the position, scale, and orientation of the composite drawing are under user control by means of a draw sheet. For example, all straight lines can be drawn on one layer, (as in Fig. 10A) all circles, arcs, and curves on another, (as in Fig. 10B) all dimension lines on another (as in Fig. 10C) and all text on another (as in Fig. 10D). As layers accumulate by being sketched and then scanned, the composite printout represents completion of an increasing percentage of a complicated final drawing. The concept of layering for composition of a complex drawing is expected to substantially increase a draftsman's productivity and increases and enhances the recognition accuracy of the processor 8.

Perfect registration of each layer with the most recent composite printout is, of course, important. If the grid sheet 10 of Fig. 2 is used, the printed grid lines can be used for registration. The grid paper 10 also can utilize four corner points 13 to facilitate accurate registration of successive layers with the printed composites. Other methods of layer registration also could be used, including punching holes 13A along one edge of the paper and placing them over a suitable stationary peg bar.

Before describing the routines executed by processor 8, it will be helpful to provide the following definitions:

SHEET - A sheet of paper scanned into the processor to control system parameters, disk i/o or drawing. Only layer and edit sheets have a central active area. The top and bottom strips outside this active area contain only global commands and registration marks or holes.

ACTIVE AREA - An area in a sheet scanned for drawing data and editing directives. The top left corner (origin), scale and orientation of a layer or edit active area can be set relative to the entire drawing.

LAYER - A set of active areas that contain only drawing data. Many active areas can span a drawing layer, and many layers can be combined to form one large output drawing.

DRAWING DATA - Active area strokes converted directly to mathematically accurate drawing entities.

DRAWING ENTITY - A line, arc, or curve (open) entities, polygon, circle or shape (closed) entities, arrowhead, extent bar or text character appearing in the drawing.

GLOBAL COMMAND - A letter possibly followed by arguments decoded to system parameters, disk i/o operations, and sheet, layer, edit and active area variables.

LOCAL COMMAND - A letter possibly followed by arguments found only within an active area.

SPATIAL INDICATOR - Active area strokes interpreted to describe drawing entities, to group drawing entities together, to guide layer editing, to fill closed entities, and to specify table entries.

EDITING DIRECTIVE - A local command associated with spatial indicators interpreted to modify previously scanned and edited drawing data.

TYPE - A letter command argument that encodes sheet, units, file, layer or edit active area information.

ID - A positive integer assigned uniquely to 1) layers of the same type, 2) entries in the same table, and 3) editing entity groups in the same active area.

Next, the definition of commands used in the flow chart of Figs. 5A-D is as follows:

Sheet Global Commands - Specify sheet type (the first command character found on the sheet when reading from top to bottom and left to right). A sheet type is one of:

S = Scan
P = Process
D = Draw
R = Report
F = File
L = Layer (type, ID, type, [*]) - Global commands and drawing data affect only the layer given by the layer type and ID number. The layer (first) type specifies that layer drawing entities belong to the drawing or one of three tables. If the ID number has been used, the system augments the existing layer. The active area (second) type

restricts the drawing entities in the active area of the sheet to one class. An optional "*" indicates that all previously scanned and edited entities in the layer will be erased before processing. A layer type is one of:

D = Drawing

L = Line style table

F = Fill style table

S = Symbol table

An active area type is one of:

L = Lines - Contains only lines and polygons.

C = Curves - Contains only arcs, circles, curves, and shapes modeled by defining points and chaining lines.

D = Dimensions - Contains only dimensioning lines, arrowheads, extent bars and numbers.

T = Text - Contains only text character strings.

E = Edit (type, [ID, ID,...]) - The sheet active area contains only editing directives that modify or augment the underlying entities in all layers belonging to the layer type (see above) and indexed by the optional ID numbers (none = all combined previously scanned and edited layers).

Scan global commands - Set SCAN button parameters.

R = Resolution (dpu) - The resolution of the scanner.

U = Units (type) - Units for positions, lengths, heights, widths and dimension values. A unit type is one of:

I = Inches

M = Millimeters

C = Centimeters

D = Drawing size (A - E) - The size of the entire drawing assembled from all layers.

S = Sheet size (A - E) - The size of the scanned sheets.

A = Active area (x, y, h, w) - Origin (x, y) of the active area relative to the origin of the sheet, and the height and width of the active area. The height must allow space for a command line at the top or bottom of the sheet (excluding holes).

Process global commands - Set sheet processing parameters:

R = Run/gap filtering (length) - Largest run/gap length to be erased/filled (default = no filtering).

S = Small object removal (length) - Length of the box bounding the largest connected pixel group to be removed (default = no removal).

J = Join/trim distance (length) - Maximum distance from endpoints on entities to other adjacent entities for endpoint extension, overshoot removal and intersection convergence (default = no join/trim).

A = Orthogonal line snap (degrees) - Maximum deviation of lines from horizontal or vertical which will snap orthogonally (default = no line snap).

G = Grid endpoint snap (length) - Length of a square grid cell which all line, arc, and curve endpoints will snap (default = no endpoint snap).

D = Dimension rectification - Enables information from dimension layers to be interpreted and applied to entities in the drawing.

Draw global commands - set DRAW button parameters:

L = Layers (type, [ID, ID, ...] - Draw the combined converted, translated, sized, rotated and clipped drawing entities in the layers belonging to the draw type and indexed by the optional ID numbers (none = all), or the text font table (the "T" type ignores ID numbers). A draw type is one of:

D = Drawing
L = Line style table
F = Fill style table
S = Symbol table
T = Text font table

W = Window (x, y, h, w) - Origin (x, y) of the output drawing relative to the origin of the entire drawing, and the height and width of the output drawing (default = entire drawing).

S = Scale (factor) - Reduction (< 1) or enlargement (> 1) of the output drawing relative to the entire drawing (default = 1).

R = Orientation (degrees) - Orientation of the output drawing relative to the entire drawing (default = 0).

Report global commands - set REPORT button parameters:

P = Parameters - Print system parameters.

L = Layer - Print layer catalog.

D = Directory - Print disk file directory.

F = Fault - Print fault code and explanation, and any relevant sheet, layer and active area data, and any unrecognized entities, commands and command arguments.

File global commands - perform drawing data disk I/O:

F = Format - Format the drawing data disk.

E = Erase - ([type], [ID, ID,...]) -

L = Load - ([type], [ID, ID,...]) -

S = Save - ([type], [ID, ID,...]) - Perform the drawing data disk I/O operation on the layers belonging to the optional file type (none = all) and indexed by the optional ID numbers (none = all), or the system parameters (the "P" type ignores ID numbers). A file type is one of:

D = Drawing

L = Line style table

F = Fill style table

S = Symbol table

P = System parameters

Layer and edit global commands - set active area parameters:

P = Position (x, y) - Position of the origin of the active area relative to the origin of the entire drawing (default = (0, 0)).

S = Scale (factor) - Reduction (< 1) or enlargement (> 1) of the output drawing relative to the entire drawing (default = 1).

R = Orientation (degrees) - Orientation of the output drawing relative to the entire drawing (default = 0).

Layer global commands - set layer active area parameters:

L = Line style (ID, width) - Line style indexed by the ID at the given width of all non-text entities in the active area (default line style = solid and width = finest line drawn on output device).

T = Text font (ID, width) - Text font indexed by the ID at the given width of all text characters in the active area (default font and height depend upon the output device - text font tables are loaded with the system program during initialization).

Edit global commands - set edit active area parameters:

F = Fill style (ID, width) - Fill style indexed by the ID at the given line width of all filled entities in the active area (default fill style = solid and width = finest line drawn on output device).

Next, the following spatial indicators and editing directives are defined:

Spatial indicators:

Layer drawing data:

Defining points and chaining lines - "+'s" or "X's" connected together by chaining lines precisely define any nonlinear entity. Arcs require three arc points and two lines. Circles need one more closing line or two opposite circle points connected by a diameter line. Curves require at least four curve points connected by at least three lines, and shapes need one more closing line.

Dimension lines, arrowheads, extent bars and numbers - "---" solid lines, "<" and ">" arrowheads, "¦" extent bars and numbers grouped together to form the pattern ¦<--- number --->¦ define dimension entities.

Editing Directives:

Entity group polygon ([0] [ID] [#]) - A surrounding polygon indicates a group of one or more entities to which other spatial indicators and local commands apply. An optional nearby ID number must index a group for copy and repeat commands and table entries. An optional lone or leading "0" indicates that only completely enclosed entities are members. An optional trailing "#" indicates that the entity group defines a table entry (table layer types only) - if the ID number has been used the system replaces the existing entry.

Registration points - One to three "+'s" or "X's" within an entity group indicate the exact position of the group for copying, repeating and symbol insertion in the drawing.

Position markers - One to three "+'s" or "X's" indicate the exact positions of the registration points of an entity group indexed by copy and repeat commands, or symbol indexed by a symbol command. Adjustments to the scale and orientation of the group or symbol achieve an exact match.

Fill marker - A small solid circle within an entity group indicates that all of the minimum area closed entities formed by group members of the same layer and entity type will be filled with the edit active area or override fill style.

Axis line - A line about which the nearest entity group will mirror or along which an indexed entity group will repeat.

Next, the following edit local commands are defined:

Edit local commands:

Drawing functions - Affect only underlying entities or entity points in an associated entity group:

E = Erase - Erase the nearest entity group in the layer (can be used with an index to exclude a copied entity group from the layer or to remove a table entry).

M = Mirror - Draw the nearest entity group mirrored about the nearest mirror axis.

C = Copy (ID, [factor], [degrees]) - Draw the entity group indexed by the ID number at the nearest position markers with optional scaling and orientation relative to the entity group.

R = Repeat (ID, n, [factor], [degrees], [factor], [degrees]) - Draw the entity group indexed by the ID number n times along the nearest repeat axis registered on the nearest position markers with optional scaling and orientation relative to the entity group. The second optional factor and degree arguments increment the scale and orientation of successive copies.

S = Symbol (ID, [factor], [degrees]) - Draw the entities in the symbol indexed by the ID number at the nearest position markers with optional scaling and orientation relative to the symbol.

B = Break - Break a curve at all entity group points (allows lines, arcs, and curves to share defining points).

Parameter and variable overrides - Override the system default or process, layer, and edit settings of the nearest entity group entities or points:

J = Join/trim nearest point of unjoined/trimmed entity

JN = No join/trim points
A = Orthogonal snap line endpoints
AN = No orthogonal snap line endpoints
GN = No grid snap endpoints

L (ID, width) = Reset line style and width
F (ID, width) = Reset fill style and width
T (ID, height) = Reset text font style and height

Figs. 9-11 show examples of the previously described layers and entities. Referring to Fig. 9, the layer 25 is made up of individual areas 26A-D. The drawing is made up of layers 25, 25A, 25B and all layers between 25A and 25B. Fig. 10A-D show the different kinds of areas that are specified by the L, (Line) C, (Curve) D, (Dimension) and T (Text) layer types. In Fig. 10A, a line area 27 shows various lines 28. In Fig. 10B, a curve area 29 is shown with the different kinds of curve entities 31 through 35. For example, the numerals 31 designate defining points chained together by a diameter chaining line 30 which defines the dotted circle 32. Numeral 33 shows arc defining points chained together by chaining lines forming the dotted arc. Numeral 34 shows curve defining points chained together with chaining lines defining the dotted curve. Numeral 35 shows shape defining points chained together with chaining lines defining the dotted shape.

Referring to Fig. 10C, dimension area 36 contains dimension entities referred to by numeral 37 containing extent bars, arrowheads, lines and numeric strings. Fig. 10D shows text area 38 which contain text strings referred to by numeral 39.

Fig. 11 shows an edit layer which contains an entity group 41 and editing local commands 46, 48 and 49. The entity group 41 contains fill marker 42, registration points 44, and groups together entities on other layers indicated by dotted lines 43. The entity group 41 has an entity group ID 47 consisting of the numerals 012, where "0" signifies to only group together completely enclosed entities in the entity group "12", and has an associated command 48 which gives an override fill style and fill line width. The mirror axis 45 and the mirror command 46 will mirror this entity group about the mirror axis. Copy command 49 copies entity group "12" enlarged by a scale factor of 2 at the position markers 50.

Next, the following command list syntax is indicated, followed by several examples:

Command list syntax:

<command list> = <command list> new line <command> |
<command list> "/" <command> |
<command>

<command> = letter <arg list> | letter

<arg list> = <arg list> "," <arg> | <arg>

<arg> = letter | number | null

Scan global command example:

S/R200/UI
DE/SA
A.5,.5,7,10

The scanner resolution is 200 dots per inch. The combined drawing will be size E and scanned sheets will be size A. The top left corner of the active area on a sheet will be at (.5, .5) inches relative to the top left corner of the sheet. The active area height can be 7 inches and width will be 10 inches.

Layer global command example:

LD,2,C/P8.5,11/S2/L2,.1

The arc/circle/curve/shape entities in the active area of this sheet belong to layer 2 of the drawing. The top left corner of the active area corresponds to (8.5, 11) units in the drawing. The active area is enlarged 2 times. Entities will be drawn in line style 2 with widths of .1 units.

Editing local command example:

R2,6,.5,,.1

Repeat entity group 2 along the nearest repeat axis 6 times registered by the nearest position markers. The first copy will be drawn half size at the original orientation, and subsequent copies will be drawn at 60%, 70%, 80%, and 90% reductions and full scale.

Referring to Figs. 3A-3B, the first step in operation by a user is to determine if the power is on, as indicated in block 100. If so, the next step, as indicated in block 104, is to determine if there is a drawing in progress. If the power is not on, then the user has to determine if there is a program disk in the disk drive, as indicated in block 101. If so, the user turns on the Power button 3A, as indicated in block 103. Otherwise, the user inserts a program disk in the disk drive, as indicated in block 102 and then turns on the power and waits for a Ready indicator

5A in block 103.

While the user is waiting for the Ready indicator 5A the processor 8 will be initializing the system, as indicated in the flow chart of Figs. 4A-4B. The program loads instructions for operating the processor in accordance with the invention while the user waits for the Ready indicator 5A. When the Ready indicator 5A occurs the user determines if a drawing is already in progress in the machine, as indicated in block 104. If no drawing is in progress, the user hits the Reset button 3B that clears the computer's memory of any previous drawing, and waits for the Ready indicator 5A to occur, as indicated in block 105.

Next, the user determines whether he is going to start a new drawing or modify an old drawing, and in the latter case, goes to block 107 and inserts a diskette containing data for the old drawing into the disk drive 4, depresses the Load button 3C, waits for the Ready indicator 5A, and then goes to block 110. If in block 106 the user determines that he is going to start a new drawing, he inserts a new drawing disk into the disk drive as indicated in block 109 and then goes to block 110. In any event, the user has taken the necessary steps to begin a drawing or modify an old drawing.

The next group of steps is determined by what the user wishes to accomplish. First the user determines if he wants to change any of the scan parameters, as indicated in block 110. If not, the next step is indicated in block 113. If the user wants to change scan parameters, he then places a "scan sheet" in scanner 2 and depresses the Scan button 3D, and waits for the Ready indicator 5A, as indicated in block

111. In any case, the user then determines if he wants to change the process parameters, and if so he places a "process sheet" in the scanner, depresses the Scan button 3D, and waits for the Ready indicator 5A to occur, as indicated in block 114.

The user then determines if he wants to generate a drawing, as indicated in block 116. If so, he determines whether to change the default drawing parameters, as indicated in block 117. If so, he places a "draw sheet" in scanner 2, depresses the Scan button 3D, and waits for the Ready indicator 5A, while DODRAW is being executed, as indicated in block 119. In any event, the user depresses the Draw button 3E and waits again for Ready indicator 5A to occur as indicated in block 120. After the Ready indicator 5A occurs, or if the user does not wish to generate a drawing, the user determines in accordance with block 121 whether he wishes to generate a report. If so, he determines whether to change the report parameters, as indicated in block 122, and if so, places a "report sheet" in scanner 2, depresses the Scan button 3D, waits for the Ready indicator 5A, as indicated in block 123, then depresses the Report button 3F and waits for the Ready indicator 5A as indicated in block 125.

The user then determines if any disk input/output operations need to be performed. For example, the user may want to make a "backup" copy of his drawing on another drawing disk. If there is to be a disk input/output operation, the user replaces the drawing disk if necessary, as indicated in block 127. The user then places a "file sheet" in scanner 2, depresses the Scan button 3D, and waits for the Ready indicator 5A, while disk I/O is being performed as indicated in block 128. If necessary, the user then replaces the drawing disk, as indicated in block 130. The user then determines whether to add to the drawing, and if so, places a "layer sheet" in scanner 2, depresses the Scan button 3D, and waits for the Ready indicator 5A, while DOLAYER is being executed as indicated in block 132. The user then determines whether to edit the drawing, and if so, places an "edit sheet" in scanner 2, depresses the Scan button 3D, and waits for the Ready indicator 5A, while DOEDIT is being executed as indicated in block 135.

Upon commencing any operation, the system extinguishes the Ready indicator 5A and lights the Busy indicator 5B. Any time the user is waiting for the Ready indicator 5A, he can depress the Stop button 3G, which aborts the ongoing process and lights the Ready indicator 5A. If any scanning operation produces an error, the system displays a "fault code" 5C, stops the operation, and lights the Ready indicator. The user then may initiate the appropriate corrective action.

Referring now to Figs. 4A-4B, processor 8 executes an initialization subroutine and idle loop called MAIN, which calls up the various other subroutines that need to be executed to perform the drafting aid function of the present invention. Once the power is turned on as indicated in block 150, the Busy indicator 5B is turned on and the Ready indicator 5A is turned off. In block 152, the program performs a processor/memory bootstrap operation and tests for failure in block 153. If there is a failure, the Fault indicator 5C is turned on in block 154 and the program waits for the Reset button 3B to be depressed, turns off the Fault indicator 5C, and returns to block 152. If the test is passed, the program goes to block 157 and loads the floppy disk program data.

The program then goes to block 158 and determines if there is any disk I/O error, and if there is, turns on the Fault indicator 5C in block 159, waits until the user depresses Reset button 5B, and turns off the Fault indicator 5C and returns to block 157. If there is no disk error, the program goes to block 162 and performs the functions of resetting system parameters, clearing the drawing data memory, and reading various mode switches to set various input and output device parameters, as indicated in block 162. The program then goes to block 163 turns off the Busy indicator 5B, turns on the Ready indicator 5A, and then goes to block 164 and waits for any of the buttons 3A-G to be depressed.

The program then turns on the Busy indicator 5B and turns off the Ready indicator 5A as indicated in block 165, and in block 166 determines if the Reset button 3B has been depressed. If it has, the program returns to block 162. Otherwise the program tests to determine if the Load button 3C has been depressed, as indicated in block 167. If it has, the program loads floppy disk drawing data as indicated in block 168 and determines if there is a disk error in block 169, and if there is, calls the fault routine, and otherwise returns to block 163. If the Load button 3C has not been depressed, the program determines if the Scan button 3D has been depressed, as indicated in block 171, and if it has, goes to block 172 and calls DOSCAN routine of Figs. 5A-5D, and waits for the results of the DOSCAN routine or aborts the DOSCAN routine if the Stop button 3G has been depressed, as indicated in blocks 173, 174, and 175.

If the Scan button 3D has not been depressed, the program determines if the Draw button 3E has been depressed, as indicated in block 176. If it has, the program calls the DODRAW routine of Figs. 8A-8D as indicated in block 177, and waits for the results of execution of that routine or aborts the execution of the DODRAW routine if the Stop button 3G is depressed, as indicated in blocks 178, 179, and 180. If the Stop button 3G has not been depressed and the results of executing the DOSCAN or the DODRAW routines are available, the program returns to block 163. If neither the Scan nor Draw buttons have been depressed, the program goes to block 181 and determines if the Report button 3F has been depressed, and if it has not, returns to block 163.

If the Report button 3F has been depressed, the program goes to block 182 and determines if RSYSTEM (See "Report Parameters" hereinafter) is greater than zero. If so, the program prints the appropriate system parameters as indicated in block 183 and then goes to block 184. If not, the program goes to block 184. In block 184 the program determines if RLAYER is greater than zero, and if so, prints a layer catalog as indicated in block 185, and then goes to block 186. If not, the program goes to block 186. In block 186 the program determines if RDIRECT is greater than zero, and if so, prints the disk directory, as indicated in block 187. If not, the program goes to block 188. In block 188 the program determines if RFAULT is greater than zero, and if so, prints appropriate fault information, and then goes to block 190. If not, the program goes to block 190.

In block 190 the program waits to detect a depression of the Stop button 3G or termination of the report. If the Stop button 3G has not been depressed the program returns to block 163 when printing of the report is complete.

The system data structures for the invention are shown in Table 1. Note that any name ending in "S" (as in LAYERS) implies the following linked list structure, which is given as an example:

```
             1st LAYER          2nd LAYER           last LAYER
             +---------+        +---------+         +---------+
LAYERS--->|  vars   |  +-->|  vars   | +-->..-->|  vars   |
             +---------+  |     +---------+ |       +---------+
             |  ptrs   |  |     |  ptrs   | |       |  ptrs   |
             +---------+  |     +---------+ |       +---------+
             |  NEXT   |--+     |  NEXT   |-+       |  NEXT   |-->null
             +---------+        +---------+         +---------+
```

where:

LAYERS =	Structure pointer (pointer to the first LAYER record)
LAYER =	Pointer to the vars, ptrs and NEXT in the record
vars =	Variables in the record, if any
ptrs =	Other structure pointers in the record, if any
NEXT =	Pointer to the next record
null =	Pointer value signifying no more records

TABLE 1

Structure	Derivation	Definition
RUNLENS	input	raw run lengths
OBJECTS	RUNLENS	groups of connected run lengths
OUTOBJS	OBJECTS	objects completely outside active area
INOBJS	OBJECTS	objects completely inside active area
COMCHARS	OUTOBJS	objects recognized as global command characters
GCOMS	COMCHARS	parsed global commands:
COM		command
ARGS		command arguments
LAYERS		layers in drawing:
...	GCOMS	(see LAYER structure variables)
ENTITYS	AREA	layer drawing entities
AREAS		layer active areas:
...	GCOMS	(see AREA structure variables)
LAYER	GCOMS	layer containing active area
LINES	INOBJS	lines
POINTS	LINES	defining points
CHARS	INOBJS	characters
STRINGS	CHARS	character strings
BLOBS	INOBJS	arrowheads
EDIT		edit active area:
...	GCOMS	(see EDIT structure variables)
LINES	INOBJS	polygon sides or axis lines
POINTS	LINES	registration points or position markers
CHARS	INOBJS	characters
STRINGS	CHARS	character strings
BLOBS	INOBJS	small filled circles
ECOMS	EDIT	parsed editing commands:
COM	STRINGS	command
ARGS	STRINGS	command arguments
BOX	STRINGS	command line box
LINE	LINES	axis line
POINTS	POINTS	position markers
EGROUP	EDIT	entity group

TABLE 1 - CONTINUED

Structure	Derivation	Definition
EGROUPS	EDIT	entity groups:
...	STRINGS	(see EGROUP structure variables)
POLY	LINES	grouping polygon
POINTS	POINTS	registration points
ENTITYS	LAYER	entities
ENTITYS		drawing entities:
AREA	AREAS	active area of original data
POINTS	AREAS	ordered points
STRINGS	AREAS	character string
POVER	ECOMS	parameter override pointer (null = none)
VOVER	ECOMS	variable override pointer (null = none)
FILL	ECOMS	fill pointer (null = none)
CLUSTERS		entity point clusters:
EPOINTS		entity points:
ENTITY		entity
POINT		point
CHAINS		chained defining points:
LINES		chaining lines
POINTS		defining points
DIMENS		dimension groups:
LINES		extent bars
STRING		number string
STRINGS		character strings:
CHARS		ordered characters
BOX		string box
CHARS		characters:
CHAR		ascii character
BOX		character box
BOX		bounding boxes:
POINTS		corners
POLYS		polygons:
LINES		ordered sides
LINES		lines:
POINTS		endpoints
POINTS		points:
CX, CY		coordinates
BLOBS		small filled blobs:
CX, CY		center points

Next, the definitions of the system parameters and variables are given below in Table 2.

TABLE 2

Command	Name	Default	Definition
Scan parameters:			
R	RES	200	resolution (dots per unit)
U	UNITS	I (inches)	units (I, M, C)
D	DSIZE	A	drawing size (A - E)
S	SSIZE	A	sheet size (A - E)
A	AX, AY	.5, .5	active area origin (units)
A	AH, AW	7, 10	active area extents (units)
Process parameters:			
R	RGLEN	0	max run/gap length (units)
S	SOLEN	0	max small object length (units)
J	JTDIST	0	max join/trim distance (units)
A	OLDEV	0	max line deviation (degrees)
G	GCLEN	0	grid cell length (units)
D	RECT	0 (off)	dimension rectification (off/on)
Draw parameters:			
D	DTYPE	D (drawing)	draw type (D, L, F, S, T)
D	DLIDS	null (all)	output drawing layer ID lists
W	DX, DY	0	output drawing origin (units)
W	DH, DW	(DSIZE)	output drawing extents (units)
S	DSCALE	1	output draw scale (factor)
R	DORIENT	0	output draw orientation (degrees)
Report parameters:			
P	RSYSTEM	1 (on)	report system parameters (off/on)
L	RLAYER	0 (off)	report layer catalog (off/on)
D	RDIRECT	0 (off)	report disk directory (off/on)
F	RFAULT	1 (on)	report fault information (off/on)
LAYER structure variables:			
L	ID	-	layer ID
L	TYPE	-	layer type (D, L, F, S)

TABLE 2 - CONTINUED

Command	Name	Default	Definition
AREA structure variables:			
L	TYPE	-	active area type (L, C, D, T)
P	X, Y	0	origin (units)
S	SCALE	1	scale (factor)
R	ORIENT	0	orientation (degrees)
L	LSTYLE	0 (solid)	line style ID
L	LWIDTH	0 (finest)	line width (units)
T	FONT	0 (default)	text font ID
T	HEIGHT	0 (default)	text height (units)
EDIT structure variables:			
P	X, Y	0	origin (units)
S	SCALE	1	scale (factor)
R	ORIENT	0	orientation (degrees)
F	FSTYLE	0 (solid)	fill style ID
F	FWIDTH	0 (finest)	fill line width (units)
EGROUP structure variables:			
-	ID	0 (none)	entity group ID
-	ENC	0 (no)	enclosure ("0" = yes)
-	FIL	0 (no)	fill (1 = yes)
-	TAB	0 (no)	table entry ("#" = yes)
E	DEL	0 (no)	erase (1 = yes)
POVER structure variables:			
J	JOIN	0 (none)	join/trim (-1 = no, 1 = yes)
A	LSNAP	0 (none)	line snap (-1 = no, 1 = yes)
G	PSNAP	0 (none)	point snap (-1 = no)
VOVER structure variables:			
L	LSTYLE	AREA	line style ID
L	LWIDTH	AREA	line width (units)
T	FONT	AREA	text font ID
T	HEIGHT	AREA	text height (units)
FILL structure variables:			
F	FSTYLE	EDIT	fill style ID
F	FWIDTH	EDIT	fill line width (units)
Drawing and table pointers:			
-	DLAYERS	-	drawing layers
-	LLAYERS	-	line style table layers
-	FLAYERS	-	fill style table layers
-	SLAYERS	-	symbol table layers
-	LEGROUPS	-	line style entry entity groups
-	FEGROUPS	-	fill style entry entity groups
-	SEGROUPS	-	symbol entry entity groups
E, D	TEGROUPS	null (none)	table entry entity groups

TABLE 2 - CONTINUED

Command	Name	Default	Definition
Current sheet, layer, area, edit and draw variables:			
(first)	STYPE	-	sheet type (S, P, D, R, F, L, E)
L, E, D	LAYERS	-	drawing or table pointer
L, E, D	LAYER	-	layer pointer
L, E	LTYPE	-	layer type (D, L, F, S)
L, E, D	LID	-	layer ID
E, D	LIDS	null (all)	layer ID list
D	DENTITYS	-	output drawing entities
Current area variables:			
-	AREA	-	active area pointer
L	ATYPE	-	active area type (L, C, D, T)
L	AERASE	null (no)	erase ("*" = yes)
Current edit variables:			
CRS, LFT	EID	0 (none)	entity group, table entry ID
L, F, T	ENUM	0 (none)	line width, font height
R	RCOUNT	-	repeat count
C, R, S	ESCALE	1	scale (factor)
C, R, S	EORIENT	0	orientation (degrees)
R	RSCALE	0	scale increment (factor)
R	RORIENT	0	orientation increment (degrees)
R	RPOINTS	-	repeat line point list

Referring now to Figs. 5A-5D, the DOSCAN routine performs the function of scanning a sheet and interpreting the action to be taken. First, it scans the page of the document in the scanner 2 into runlengths in the above mentioned RUNLENS data structure as indicated in block 200. Suitable scanning hardware and circuitry are available in the commercially available GTX 5000 system, and are described in detail in the allowed pending patent application "HIGH SPEED SERIAL PIXEL NEIGHBORHOOD PROCESSOR AND METHOD" application, by Krich, serial No. 016,230, filed 2/19/87, incorporated herein by reference. If there is a scan error, the fault routine is called. If not, the program determines if RGLEN is greater than zero. RGLEN is a process parameter that determines run length filtering by filling small gaps and erasing small runlengths, both of length RGLEN units. If so, fill and erase operations are performed as indicated in block 204.

In any event, the program goes to block 205 and determines if RUNLENS is empty, and if it is, calls the fault routine, but otherwise goes to block 207 and groups the connected runlengths into "objects" in the OBJECTS data structure. This can be done using known hardware and software available in the above-mentioned GTX 5000 system and is described in detail in allowed pending patent application "METHOD AND APPARATUS FOR SIMPLIFYING RUNLENGTH DATA FROM SCANNING OF IMAGES", by Roye, filed on 2/19/87, Serial No. 016,662 which is incorporated herein by reference. Next the program goes to block 208 and determines if SOLEN is greater than zero. SOLEN is a process parameter that determines if objects smaller than length SOLEN units are to be removed. If so, object removal is performed as indicated in block 209. In any case, the program goes to block 210 and determines if there is more than one object, and if not, calls the fault routine.

Otherwise the program converts the first object into a sheet type in the variable STYPE as indicated in block 212, and then in block 213 determines if STYPE is equal to one of S (scan), P (process), D (draw), R (report), or F (file) and if it is not, the program goes to block 241, but otherwise goes to block 214 and converts and removes the OBJECTS characters into COMCHARS, which is a list of global command characters. This can be done using known hardware and software available in the above-mentioned GTX 5000 system and is described in detail in allowed pending patent application "METHOD AND APPARATUS FOR GENERATING SIZE AND ORIENTATION INVARIANT SHAPE FEATURES", Serial No. 026,672, filed 3/13/87, by Horowitz, and also allowed pending patent application "HIERARCHICAL PARAMETRIC APPARATUS AND METHOD FOR RECOGNIZING DRAWN CHARACTERS", Serial No. 199,361, filed 5/26/88, by Filipski, both incorporated herein by reference. (It should be noted that "[operation] and remove" as used herein followed by a structure name, means that each member of the data structure is removed therefrom after a successful operation; in the flowcharts, the term "and remove" is omitted for simplicity.) The program then goes to block 215 and determines if there are any more objects, which indicates unrecognized characters, and if so, calls the fault routine. If there are no more unrecognized characters, the program goes to block 217 and parses and removes the successfully converted COMCHARS characters into the GCOMS global command structure, previously defined. That is, the program parses the COMCHARS and removes all those that fit into the command syntax previously established for GCOMS.

The program then goes to block 218 and determines if there are any more COMCHARS which indicates unparsed characters, and if so, calls the fault routine. If there are no unparsed characters, the program goes to a string of decision blocks including 220, 222, 224, 226, and 228, and determines which of the types S, P, D, or R, that STYPE is, and appropriately decodes GCOMS to get system parameters, as indicated in blocks 221, 223, 225, and 227, respectively. If STYPE is not any of the foregoing types, or if it is an S, P, D, or R, the decoded result is obtained and the program goes to block 234. If STYPE is equal to F, the program decodes GCOMS to perform disk input/output, as indicated in block 229, and the program goes to block 230 and determines if there are any more GCOMS, which indicates invalid commands, and if so, calls the fault routine, but otherwise goes to block 232 and determines if there is a disk error, and if there is, calls the fault routine, and if not, returns to the calling program.

In block 234 the program determines if there are any more GCOMS, and if so, calls the fault routine as indicated in block 235, but otherwise goes to block 236 and updates the system parameter file, and then determines if there is a disk error, if there is, calls the fault routine, and otherwise returns to the calling program, as indicated in blocks 236 and 237.

Next, the program goes to block 241 and moves OBJECTS completely outside of the active area 14A into OUTOBJS, and also moves OBJECTS completely inside the active area 14A of Fig. 2 into INOBJS. Next, the program goes to block 242 and determines if OBJECTS is empty, which indicates there are no objects crossing the active area border, and if it is not, calls the fault routine, but if OBJECTS is empty, the program determines if INOBJS is empty and OUTOBJS is empty in blocks 244 and 246, and if so in either case, the program calls the fault routine. If they are both not empty, the program goes to block 248 and converts and removes OUTOBJS characters to COMCHARS.

The program then determines in block 249 if OUTOBJS is empty, if it is not, calls the fault routine, and if it is, goes to block 251 and parses and removes COMCHARS characters into GCOMS commands. The program then goes to block 252 and determines if COMCHARS is empty, if it is not, calls the fault routine, and if it is, goes to block 253 and sets STYPE equal to the first GCOM->COM command letter. Next, the program goes to block 254 and determines whether STYPE is one of L (layer) or E (edit), and if it is not, the fault routine is called. If it is, the program goes to block 256 and sets LTYPE equal to the first GCOM->ARG, i.e., set LTYPE to the first argument of the first global command.

Next the program goes to a string of decision blocks 257, 259, 261, and 263 where it determines if the LTYPE is equal to D (drawing), L (line style), F (fill style), or S (symbol table), and if it is, sets LAYERS equal to DLAYERS, LLAYERS, FLAYERS, or SLAYERS, respectively, and goes to block 266. If LTYPE is not equal to D, L, F, or S, the program calls the fault routine.

In block 266 the program determines if STYPE is equal to L, and if it is not, block 268 it determines if STYPE is equal to E. If STYPE is equal to L, the program calls the DOLAYER routine of Figs. 6A-6D, as indicated in block 267, and returns to the calling program. If STYPE is equal to E, the program calls the DOEDIT routine of Figs. 7A-7I, as indicated in block 269 and returns to the calling program.

Referring now to Figs. 6A-6D, the DOLAYER routine is called after the sheet has been scanned and has been determined to be a layer sheet. The DOLAYER routine then scans the active area AREA of the sheet and puts all of the raw entities into different entity records. In block 300 the program decodes the second, third, and optional fourth arguments of the first command, the L command, into LID, ATYPE and AERASE, and the remaining GCOMS into AREA variables (vars). The program then goes to block 301 and determines if GCOMS is empty, and if it is not, calls the fault routine.

Then, in block 302 the program translates, scales, and rotates INOBJS according to the AREA variables. The program then, in block 302A, determines if all transformed objects in INOBJS are entirely inside the drawing, and if not, calls the fault routine. The program then goes to block 303 and finds which layer has a layer identification number equal to LID. In block 304, if no such layer is found, the program creates a new LAYER and appends it to LAYERS, and assigns to it the identification number equal to LID and a type equal to LTYPE, as indicated in blocks 305 and 306, from which the program proceeds to block 310.

If the determination of block 304 is affirmative, the program goes to block 307 and determines if AERASE is equal to an asterisk, and if it is, removes all entities from the LAYER in block 308 and in block 309 removes all AREAS pointing to the LAYER. The program otherwise proceeds to block 310.

In block 310 the program creates a new AREA and appends it to the AREAS, and in block 311 sets the LAYER pointer of AREA equal to LAYER and sets the TYPE variable of AREA equal to ATYPE. The program then goes to block 312 and determines if ATYPE is equal to L, i.e., whether the type of the present area is a line area. If it is, the program goes to block 313 and processes the present area in accordance with blocks 313-319 before returning to the calling routine.

At block 313 the program converts and removes all objects inside the active area into AREA->LINES. This can be done using known hardware and software available in the above-mentioned GTX 5000 system and is described in detail in allowed pending patent application "APPARATUS AND METHOD FOR VECTORIZATION OF INCOMING SCANNED DATA", by Lien, filed on 2/19/87, Serial No. 016,660, incorporated herein by reference. The program then goes to block 314 and determines if there are any more INOBJS, which indicates the presence of other entities, and if so, calls the fault routine.

Otherwise the program goes to block 316 and enters a loop to process all lines in the present active area. For each line in the active area, the program creates a new entity in block 317 and appends it to the present layer. In block 318 the program sets the new entity to point to the present active area and sets the entity's points equal to the two line endpoints and then in block 319 removes the line from the area, to conserve memory space. When this has been completed for each entity in the active area, the program returns to the calling routine.

If the determination of block 312 was negative, the program determines in block 320 if ATYPE is equal to C, i.e., is a curve area. If so, the program goes to block 321 and converts and removes INOBJS lines into AREA->LINES in block 321, determines if INOBJS is empty in block 322, and if it is, calls the fault routine. Otherwise the program goes to block 324 and extracts and removes all sets of lines in AREA forming crosses to AREA->POINTS. The program then goes to block 325 and extracts and removes points properly connected by lines from AREA->POINTS and AREA->LINES into CHAINS.

Next the program goes to block 326 and determines if there are any more AREA->LINES or AREA->POINTS, which indicates isolated points or lines without two endpoints, and if not, calls the fault routine. Otherwise the program goes to block 328 and begins to loop through all of the chains. For each chain, the program creates a new entity and appends it to the present layer, as indicated in block 329. The program then goes to block 330 and sets the AREA pointer of the new entity to the present AREA and ENTITY->POINTS to CHAIN->POINTS. The program then goes to block 331 and determines if there are two chained points, which indicates a circle, and if so, goes to block 332 and generates an additional third point, and appends the first point to the last, as indicated in block 333. If there are more than two chained points, the program goes to block 334 and determines if the number of chained points is equal to the number of chaining lines, which indicates that the entity is closed, and if it is, the program goes to block 333 and appends the first point to the last. In either case, the program then goes to block 335 and removes the present chain and returns to block 328. When this process has been completed for each chain in the active area, the program returns to the calling routine.

If the determination of block 320 is negative, the program goes to block 336 (Fig. 6C) and determines if ATYPE is equal to D, i.e., is a dimensions area, and if it is not, goes to block 350 (Fig. 6D), but if it is, goes to block 337. The program then converts and removes INOBJS characters into AREA->CHARS, INOBJS lines into AREA->LINES, and INOBJS filled triangles into AREA->BLOBS, in that order, as indicated in blocks 337, 338, and 339. In block 340 the program determines if INOBJS is empty. If it is not empty, it calls the fault routine. If INOBJS is empty, the program goes to block 342 and extracts and removes colinear adjacent AREA->CHARS into AREA->STRINGS. These strings also may contain converted extent bars and arrowheads. The program then goes to block 343 and extracts and removes AREA->STRINGS, AREA->LINES, AREA->BLOBS dimension lines, arrowheads and extent bars and AREA->STRINGS numbers forming colinear and adjacent ¦<---number--->¦ patterns into DIMENS in accordance with well known "blackboard" programs such as the one used in the commercially available GTX 5000 system yet, and described in "The HEARSAY-II speech understanding system: integrating knowledge to resolve uncertainty", "Computing Surveys" 12, pp. 213-253, 1980, by L.D. Erman, F. Hayes-Roth, V. Lesser, and D. Reddy, and also in pending patent application "METHOD AND APPARATUS FOR RECOGNITION OF GRAPHIC SYMBOLS", Serial No. 016,253, filed 2/19/87, by Bhaskaran, and incorporated herein by reference.

Next, the program goes to block 344 and determines if the AREA->LINES, AREA->STRINGS and AREA->BLOBS are empty, and if any is not empty, the fault routine is called.

If they are empty, the program goes to block 346 and enters a loop for each of the dimension records in DIMENS. First, the program creates a new entity to append to LAYER in block 347, sets the AREA pointer of the new entity to the present AREA, sets ENTITY->POINTS to the endpoints of the extent bars, sets ENTITY->STRING to DIMEN->STRING, and then removes the present dimension from DIMENS, as indicated in blocks 347, 348, and 349. When this has been completed for each of the dimension records in the dimension list, the program returns to the calling routine.

If the determination of block 336 is negative, the program goes to block 350 (Fig. 6D) and determines if ATYPE is equal to a T, i.e, a text area. If so, the program goes to block 351, converts and removes INOBJS characters into AREA->CHARS. In block 352 the program determines if INOBJS is empty, and if it is not, calls the fault routine. If INOBJS is empty, the program goes to block 354 and extracts and removes colinear adjacent AREA->CHARS into AREA->STRINGS.

Then, as indicated in block 355, for every string in the present area, the program creates a new entity and appends it to the present layer, sets the ENTITY->AREA pointer equal to AREA and the ENTITY->STRING equal to AREA->STRING, and then removes the string from AREA->STRINGS, as indicated in blocks 356, 357, and 358. When this has been completed for each string of characters in AREA, the routine returns to the calling program.

If the determination of block 350 is negative, which indicates that the area type was not one of L, C, D, or T, the program calls the fault routine.

Referring to Figs. 7A-7H the DOEDIT program allows editing of previously scanned drawing layers. In block 400 the DOEDIT routine decodes and removes the first GCOM arguments into LIDS and decodes and removes the remaining GCOMS into EDIT variables. In block 401 the routine determines if GCOMS is empty, and if it is not, calls the fault routine. If GCOMS is empty, the program goes to block 402 and the program translates, sizes and rotates INOBJS according to edit variables. The program in block 402a determines if all transformed objects in INOBJS are entirely inside the drawing, and if not, calls the fault routine. The program then converts and removes INOBJS characters into EDIT->CHARS, and then, block 404 extracts and removes colinear adjacent EDIT->CHARS into EDIT->STRINGS. In block 405 the program parses and removes EDIT->STRINGS into ECOMS. In block 406, the program converts and removes INOBJS lines into EDIT->LINES, and in block 407, the program converts and removes EDIT->LINES forming crosses into EDIT->POINTS.

In block 408 the program converts and removes INOBJS filled circles into EDIT->BLOBS. In block 409 the program determines if INOBJS is empty, and if it is not empty, calls the fault routine. If INOBJS is empty, the program goes to block 411 and extracts minimum area polygons from EDIT->LINES into POLYS, and in block 412 determines if POLYS is empty, and if it is, goes to block 413 and determines if EDIT->STRINGS, EDIT->LINES, or EDIT->BLOBS is empty, and if any is not empty, calls the fault routine. If POLYS is not empty, the program goes to block 415 and enters a loop for each polygon in POLY. For each polygon the program creates a new entity group EGROUP and appends it to EGROUPS. In block 417 the program sets the entity group polygon to POLY, and in block 419 removes POLY from POLYS. When the loop has been completed for all of the polygons, the program goes to block 419. If the determination of block 413 is affirmative, the program goes to block 471 (Fig. 7D).

In block 419 the program begins a loop for each string in the edit active area finding the nearest entity group polygon in block 420, and then first decoding the string characters in block 421 into EGROUP->ENC, EID, and EGROUP->TAB. In block 422 the program determines if the string characters list is empty, and if it is not empty, calls the fault routine. Otherwise, the program determines in block 424 if the entity group identification number is zero, and if it is, goes to block 430 to determine if the entity group is a table entry, and if it is not, the program goes to block 433 and removes the present string of characters from EDIT->STRINGS and returns to the beginning of the loop at block 419.

If the entity identification number is not zero in block 424, the program goes to block 425 and determines if the entity identification number has been assigned already to a different entity group, and if it has, calls the fault routine. If the entity identity number has not been assigned to another group, the program goes to block 427 to see if the entity group identification number is zero, and if it is not, which indicates that the entity group already has been assigned an identification number, calls the fault routine, but otherwise assigns the entity group identification number to EID, and then goes to block 430. If the entity group is a table entry, the program goes to block 431 and determines if LTYPE is D or if EID is zero.

If this is true, it means that the layer is not a table or that the table entry index is zero, and the fault routine is called. Otherwise, the program goes to block 433.

When the loop has been completed for all strings in the edit active area string list, the program goes to block 434 and loops through all points in the edit active area. In doing this, the program first goes to block 435 and enters a loop for each EGROUP to determine if each point is in it and if so, then if the number of entity group points is already the maximum of three, and if so, calls the fault routine. Otherwise, the program appends the present point to the entity group points, which are registration points, and removes the point from EDIT->POINTS as indicated in block 439, so that at the end of the loop beginning at block 434 only position markers will remain.

The program then goes to block 440 and loops through blobs in the edit active area. In block 441 the program enters another loop through the entity groups, and in block 442 determines if the present blob is inside the present entity group, and if it is, sets EGROUP->FIL to one in block 443, indicating that closed entities are to be filled, and in block 444 removes the blob from EDIT->BLOBS and returns to the beginning of the loop. When this has been completed, the program goes to block 445 to determine if the blob list is empty, and if it is not, calls the fault routine, indicating that not all the blobs were filled circles, but otherwise goes to block 450.

In block 450, the DOEDIT routine enters into a loop for each layer in LAYERS in which it first determines if LIDS is empty, and if it is not, goes to block 452 and finds and removes the LID in LIDS equal to the layer identification number. The program then goes to block 453 and determines if LID has been found, and if so, goes to block 454 but otherwise returns to the beginning of the loop.

If LIDS is not empty, the program goes to block 454, and enters another loop for each entity in the present layer entity list. This loop contains a subloop 454 wherein for each EGROUP in the entity group list, it determines in block 456 if the generated entity is completely inside the present EGROUP polygon or if it intersects the present EGROUP polygon and EGROUP->ENC is zero, which indicates that membership does not require complete enclosure. If this determination is affirmative, the program goes to block 457, but otherwise returns to the beginning of the subloop. In block 457 the program appends the present entity to the entity group list, and in block 458 tags all entity points inside the EGROUP polygon. In block 459 the program determines if EGROUP->ENC is zero, and if it is, returns to block 455, but if it is not, returns to block 454 to determine if the present entity possibly intersects another entity group.

When the loop beginning at block 455 is done, it returns to block 454, and when the subloop beginning at 454 is done, it returns to block 450. When the loop beginning at 450 is done, it goes to block 460 and it determines if LIDS is empty, and if it is not, calls the fault routine, which indicates identification numbers without a corresponding layer. If LIDS is empty, the program goes to block 462 and begins a loop for each entity group in the entity group list in which the program determines if the present entity group is empty in block 463, and if it is, calls the fault routine, but otherwise goes to block 465 to determine if the closed entities in the entity group are to be filled, if it is not, the program goes back to the beginning of the loop beginning at block 462, but otherwise goes to block 466 and starts a new subloop for each entity in the entity group list wherein in block 467 it determines if the entity is a closed curve, i.e., circle or shape, and if it is, the program goes to block 468 and determines if a FILL record exists, and if it is not, creates the fill record in block 469, but otherwise goes to block 470 and sets the entity FSTYLE and FWIDTH equal to the edit active area FSTYLE or to FWIDTH to indicate the style for filling and the width of any fill pattern lines. The program then goes back to block 466 for the next entity in the present entity group. If the determination of block 467 is negative, the program goes back to block 466.

When the loop beginning at block 462 is done, the program goes to block 471. In block 471 the program begins a new loop for each entity command, in which the first step in block 472 is to determine if the present command is one of the commands E (erase), M (mirror), or B (break), or J (join), A (angle line snap), G (grid point snap), L (line style), F (fill style), or T (text font) overrides. If the determination is affirmative, the program goes to block 473 and determines if the entity group is empty, and if it is, calls the fault routine, but otherwise finds the nearest polygon in the entity group as indicated in block 475. The program then goes to bock 476 and sets the entity group associated with that command to the entity group found. The program then goes to block 477. If the determination of block 472 is negative, the program goes to block 477. In block 477 the program determines if the editing command is a mirror or repeat command. If it is, the program goes to block 478 and determines if the edit axis lines are empty, and if they are, calls the fault routine, but otherwise goes to block 480 and finds the nearest axis line, and then goes to block 481 and sets the edit command axis line equal to the line found in block 480. The program then goes to block 482.

If the edit command is not a mirror or repeat command, the program goes to block 482, and determines if the edit command is a copy, repeat, or symbol command, and if it is not, returns to block 471, but if it is, goes to block 483, decodes the argument of the first edit command into EID, then goes to block 484 and determines if EID is equal to zero, and if it is, calls the fault routine, but otherwise goes to block 486 and determines if the edit command is a copy or repeat command. If it is, the program goes to block 487 and finds the entity group with the same identification number as EID, and then goes to block 490 and determines if such an entity group has been found, and if not, calls the fault routine. If the edit command is not a copy or repeat command, the program goes to block 489 because the command then is a symbol command, and finds the entity group in the symbol table pointed to by SEGROUPS with a table entry identification number equal to EID, and then determines in block 490 if such a group was found, and if not, calls the fault routine. If the determination of block 490 is affirmative, the program goes to block 492 and sets the entity group associated with the edit command equal to EGROUP. The program goes to block 493 and determines if the list of edit points is empty, and if it is, calls the fault routine, but otherwise goes to block 495 and finds the nearest position markers with a count equal to the corresponding entity group registration points. In block 496 the program determines if the position markers are found, and if not, calls the fault routine, but otherwise goes to block 498 and sets the edit command position markers equal to the points found. The program then goes back to block 471.

If the loop beginning at block 471 is done, the program goes to block 499 and removes all assigned EDIT->LINES, goes to block 500, determines if the list of lines is empty, and if it is not, calls the fault routine. Otherwise the program goes to block 502 and removes all assigned EDIT->POINTS, goes to block 503, determines if the list of points is empty, and if it is not, calls the fault routine, and then goes to block 510 (Fig. 7E). In block 510 the program begins a loop for each edit command, in which it determines if the present edit command is equal to E (erase), and if it is, goes to block 512 and sets EGROUP->DEL equal to 1, which indicates that the entity group will be deleted after editing is completed, and then goes to block 558.

If the present edit command is not equal to E, the program goes to block 513 and determines if ECOM is equal to B (break), and if it is, enters a subloop at block 514 wherein for each entity, it determines if the area type is equal to C, and if it is, goes to block 516 and duplicates each single entity point tagged (inside the entity group) and returns to the beginning of the loop. When the subloop is done, the program goes to block 558. If the entity command is not equal to B, the program goes to block 517 determines if the edit command is equal to J (join).

If the determination of block 517 is positive, the program goes to block 518 and begins a new subloop for each entity in which the program goes to block 519 and determines if the ENTITY->POVER record exists, and if it does not in block 520, creates a point override record POVER and then goes to block 521. If POVER exists, the program goes to block 521 and determines if the first argument of the edit command is equal to N, and if it is, goes to block 522 and sets the variable JOIN of the point override list to -1, which informs the DODRAW routine not to join any points in the entity group, as indicated in block 522 and then returns to block 518. If the argument of the edit command is not N, the program goes to block 523 and sets JOIN equal to 1 which informs the DODRAW routine to join the present entity points to the nearest unjoined entity points.

When the loop of block 518 has been completed for each entity, the program goes to block 558.

If the edit command is not J, the program goes from block 517 to block 524 and determines if ECOM is A (angle line snap). If this determination is affirmative, the program goes to block 525 and begins a subloop for each entity in which it is determined in block 526 if the ENTITY->POVER record exists, and if it does not, the program creates a point override record POVER in block 527 and goes to block 526. If POVER exists, the program goes to block 528 and determines if the argument of the edit command is N, and if it is, sets LSNAP equal to -1, and otherwise sets LSNAP equal to 1, and then returns to the beginning of the loop. When the subloop 525 has been completed for each entity, the program goes to block 558.

If the edit command is not A, the program goes to block 531 (Fig. 7F) and determines if the edit command is G (grid point snap). If this is the case, the program begins a subloop at block 532 which is essentially similar to the subloops of blocks 518 and 525, except that if the argument of the edit command is N in block 535, then PSNAP is set equal to -1 in block 536, but otherwise the fault routine is called, and when this has been completed for each entity, the program goes to block 558.

If the edit command is not G, the program goes to block 538 and determines if the edit command is equal to L, F, or T, which indicate line style, fill style, or text. If the edit command is one of these commands, the program goes to block 539 and decodes the arguments of the edit command into EID and ENUM, tests for an error, and if there is, calls the fault routine, but otherwise goes to block 542. If the edit command is not equal to L, F, or T, the program returns to block 510.

In block 542 the program determines if the edit command is equal to L. If it is, the program goes to block 543 and begins a subloop for each entity in which it determines if the entity type is indicated by L, C, or D, that is, a line, curve, or dimension entity. If it is not, the program returns to block 543, but if it is one of these three, then the program goes to block 545 and determines if the ENTITY->VOVER variable override record exists, and if it does not, the program creates a variable override record VOVER and goes to block 547. If VOVER exists, the program goes to block 547 and sets the line style and line width variables of VOVER to EID and ENUM. That is repeated for each entity, and then the program goes to block 558.

If the ECOM is not equal to L, the program goes to block 548 and determines if ECOM is equal to F. If it is, the program goes to block 549 and for each entity determines if the fill style record FILL exists in block 550, and if it does not, the program goes back to block 549, but otherwise goes to block 551 and sets the fill style and fill line width variables of FILL to EID and ENUM, and returns to block 549. When this is done for each entity, the program returns to block 558.

If ECOM is not equal to F, the program then goes to block 553 because the command then is a text command, and for each entity, determines in block 554 whether the entity type is D or T, that is, a dimension or a text entity, and if not, the program goes to block 555 and determines if the ENTITY->VOVER record exists, if it does not, creates VOVER in block 556 and then goes to block 557. If VOVER exists, the program goes to block 557 and sets the font and height variables of VOVER to EID and ENUM and returns to block 553. When this has been completed for each entity, the program goes to block 558.

In block 558 the program removes the edit command ECOM from the list ECOMS. The program then goes from block 558 back to the beginning of the loop starting at block 510, and when that loop has been completed for each edit command, the program goes to block 560 (Fig. 7G).

It should be noted that the foregoing portion of DOEDIT sets up all of the edit commands associated with entity groups that do not require creating entities. These edit commands look at the various entity groups and set certain entity attributes which are in place when the M, C, R, and S commands create new entities.

Blocks 560-565 (Fig. 7G) set pointers to the correct table from the layer type set in the DOSCAN routine. If LTYPE is L, F, or S, TEGROUPS is set to LEGROUPS, FEGROUPS, or SEGROUPS, respectively. The program goes to block 566, which begins a loop for each entity group in the edit command list, in which in block 567 the program determines if EGROUP->TAB is equal to #, and if it is, the program goes to block 568 and finds and removes any table entry with a prior index EGROUP->ID. The program then goes to block 569 and determines if EGROUP->DEL is set to 1, and if it is not, goes to block 570 and appends EGROUP to TEGROUPS and returns to block 566. If the determination of block 567 is negative or the determination of block 569 is affirmative, the program returns to block 566.

When the loop beginning with block 566 is done, the program goes to block 571 and begins a new loop for each edit command, in which it determines in block 572 if the edit command is M (mirror). If it is, in blocks 573 the program begins another subloop for each entity in the associated entity group in which it creates a new entity record in block 574 and appends it to the layer pointed to by the original active area of the entity, and in block 575 mirrors the entity about the mirror axis into the new entity and then returns to block 573. When this loop is done for each entity, the program goes to block 595 and removes the present edit command and returns to block 571.

If the present edit command is not M, the program goes to block 576 (Fig. 7H) and determines if the edit command is C (copy) or S (symbol). If so, the program goes to block

577 and decodes the arguments of the edit command into ESCALE and EORIENT to obtain scaling and orientation information. A decoding error results in calling of the fault routine. Otherwise the program goes to block 580 and begins executing a loop for each entity, including creating and appending a new entity record in block 581, and tranforming the entity to generate the new entity so that the entity registration points match the edit command position markers. When this process is done, the program goes to block 595 and removes the edit command and returns to block 571. If the edit command is not C or S, the program goes to block 583 and determines if the edit command is R (repeat), and if it is not, calls the fault routine, but if it is, the program goes to block 584 and decodes the arguments of the edit command into RCOUNT, ESCALE, EORIENT, RSCALE, and RORIENT. The fault routine is called if there is a decode error. Otherwise, the program goes to block 587 and generates RCOUNT equispaced RPOINTS along a line parallel to the edit command axis line through the top left points of the edit command position markers.

In block 588, for each entity, and for each generated repeat line point, as indicated in block 589, the program creates and appends a new entity, translates the position markers to the repeat line point and transforms the entity so that the entity registration points match the translated position markers, as indicated in block 592. The program then goes to block 593 and increments ESCALE and EORIENT by setting ESCALE equal to ESCALE plus RSCALE and setting EORIENT equal to EORIENT plus RORIENT, and then returns to block 589.

When the subloop of block 589 is complete, the program returns to block 588. When the program process has been completed for each repeat line point and each entity, the program goes to block 595. When the loop beginning with block 571 has been completed for each edit command, the program goes to block 596.

In block 596 the program determines if the edit command list is empty, and if it is not empty, calls the fault routine. If it is empty, the program goes to block 598, and for each entity group, determines if DEL is equal to 1 indicating that the entity group should be deleted, and if not, goes to block 607, but if so, goes to block 600, and for each entity in the entity group, finds and removes that entity from its layer, as indicated in block 601.

In block 602, the program determines if the entity belongs to any table entry or LTYPE is equal to D, and if not, goes back to block 600, but if so, goes to block 603, and for each entity group, finds the entity as indicated in block 604 and removes it from any table entries, goes to block 605, determines if that table entry is empty, and if it is, removes EGROUP from TEGROUPS, and returns to block 603. If EGROUP is not empty, the program returns to block 603. When the loop beginning with block 603 has been completed for each EGROUP, the program returns to block 600. When the loop beginning with block 600 is completed for each entity, the program goes to block 607 and removes EGROUP from EGROUPS, and then returns to block 598. When the loop beginning with block 598 has been completed, DOEDIT is complete, and returns to the calling program.

The DODRAW flowchart of Figs. 8A-8D takes the drawing entities created in the DOLAYER routine and all of the modifications and augmentations that were done to those drawing entities by the DOEDIT routine and "loops" through the various layers and the drawing entities therein to effectuate printing of designated layers or composite drawings.

Referring to blocks 610-617, if DTYPE is D, L, F, or S, then the program sets LAYERS to DLAYERS, LLAYERS, FLAYERS, or SLAYERS, respectively. Also, in blocks 612-617, if DTYPE is one of L, F, or S, then TEGROUPS is set equal to LEGROUPS, FEGROUPS, or SEGROUPS, respectively. This in effect sets the pointer LAYERS to the correct drawing or table layers and sets the pointer TEGROUPS to the correct set of table entries, if any. If DTYPE is not equal to any of D, L, F, or S, the fault routine is called.

Next, the program goes to block 618, and begins a loop through each layer identification number list LIDS in DLIDS that comprise each output drawing. Then, the program goes to block 619, and begins a loop through each layer in which the program, in block 620, determines if LIDS, which is the list of layer identification numbers to draw, is empty. If not, the program in block 621 attempts to find the layer identification number in the list of layer identification numbers that matches the identification number of the present layer. Then in block 622, the program determines if that layer identification number was found, and if not, returns to block 619. If the layer identification list is empty, or if the layer identification number was found, the program goes to block 623 and begins a new loop through each entity in the layer whereby the program appends that entity to a new entity list DENTITYS that contains all entities that will be in the drawing. When the loop is done, the program returns to block 619.

When the loop of block 619 is done, the program goes to block 625 and determines if JTDIST, the maximum join/trim distance, is greater than zero. If it is, the program goes to block 626 and begins a loop through each entity in which it goes to block 627 and which clusters all ENTITYS->POINTS, except points tagged in block 458 of the above described DOEDIT routine with an existing join/trim override JOIN not equal to zero, within the distance JTDIST of each other into CLUSTERS. Clustering is described in detail in the above incorporated by reference Bhaskaran pending patent application. When the loop of block 626 is completed, or if JTDIST is not greater than zero, the program goes to block 628.

In block 628, the program starts a loop for each entity in which, in block 629, it starts a subloop for each tagged point in the present entity with an existing join/trim override JOIN equal to one, wherein it finds the nearest point in any entity and the nearest point in any cluster in block 630. Then in block 631, the program determines if the nearest entity point that it found is equal to the nearest cluster point that it found. If so, the program appends that point to the cluster, as indicated in block 632, and then returns to block 629. If the points were not equal in block 631, the program goes to block 633 and creates a new cluster that contains the tagged point and the nearest entity point, and then returns to block 629.

When the loop of block 629 is done, it returns to the loop of block 628. When the loop of block 628 is done, the program goes to block 634 and begins a new loop through each cluster, wherein in block 635 the program computes the centroid of the present cluster, and in block 636 it enters a subloop for each point in the cluster, wherein in block 637 the program sets each point equal to the cluster centroid. When this has been done for each point in each cluster, the loop of block 634 is done and the program goes to block 638. From block 625 through block 637, the DODRAW routine has extended entity points to meet nearby entities, removed small endpoint overshoots, and converged points close to intersections.

In block 638, the program enters a loop through each entity wherein in block 639, it determines if OLDEV, the maximum orthogonal line deviation, is greater than zero, and if it is, goes to block 640 and "snaps" all linear ENTITY->POINTS endpoints, except tagged points with an existing line snap override LSNAP not equal to zero, within an angle OLDEV degrees of horizontal or vertical to exactly horizontal or vertical, respectively. Then, or if OLDEV is not greater than zero, the program goes to block 631 and determines if an existing line snap override LSNAP of that entity is equal to one, and if so, goes to block 642 and "snaps" all tagged linear ENTITY->POINTS endpoints to the closest horizontal or vertical line. Then, or if LS is not equal to one, the program goes to block 643 and determines if GCLEN, the grid cell length, is greater than zero, and if it is, the program goes to block 644 and "snaps" all ENTITY->POINTS endpoints, except tagged points with the point snap override PSNAP equal to -1, to the nearest grid point. Then, or if GCLEN is not greater than zero, the program goes to block 645 and determines if RECT, which is a dimension rectification flag, is greater than zero. If it is, and the entity area type is equal to L or C, the program goes to block 646 and extracts the dimensions from the nearest dimension entities in DENTITYS and rectifies ENTITYS->POINTS, as described in detail in the above referenced U.S. patent 4,058,849. Then the program goes to block 647 and determines if there is a rectification error, i.e., whether any dimensioning was underdetermined, overdetermined, or otherwise inconsistent, and if so, calls the fault routine, but otherwise returns to block 638. If RECT is equal to zero, the program goes back to block 638.

When the loop of block 638 has been performed for each entity, the program goes to block 650 (Fig. 8C) and starts a loop for each entity wherein in block 651 the program "clips" and "transforms" all ENTITY->POINTS and ENTITY->STRING->BOX->POINTS by DX, DY, DH, DW, DSCALE, and DORIENT, which represent the output drawing window, scale, and orientation. The program then goes to block 652 and determines if the entity type is equal to L, and if it is, it determines if the entity variable override record exists, and if it does not, it outputs the line represented by ENTITY->POINTS in the line style and line width given by the entity area record. If so, it outputs the line represented by ENTITY->POINTS in the line style and line width of the entity variable override record, indicated in blocks 654 and 655. The program then goes to block 673.

If the entity type is not L in block 652, the program goes to block 656 and determines if the entity type is C. If it is, the program goes to block 657 and enters a loop through each set of ENTITY->POINTS separated by endpoints and duplicate point pairs. In the loop, the program goes to block 658 and determines if the entity variable override record exists. If it does not, it outputs the spline represented by the ENTITY->POINTS set in the line style and line width given by the entity area record. If it does exist, it outputs the spline represented by the ENTITY->POINTS set in the line style and line width given by the entity variable override record, as indicated in blocks 659 and 660. Then the program goes to block 651 and determines if the entity fill record exists, and if it does, the program goes to block 662 and outputs a filled shape bordered by the ENTITY->POINTS spline in the fill style and fill line width given by the entity fill record. In either case, the program returns to block 657, and when that loop is done, the program goes to block 673.

If the entity type was determined not to be C in block 656, the program goes to block 663 (Fig. 8D) and determines if the entity type is equal to D, and if so, the program goes to block 664 and determines if the entity variable override record exists. If it does not, then in block 665 and 666, the program outputs the line, arrowhead, and extent bar pairs within the box defined by ENTITY->POINTS in the line style and line width given by the entity area and also outputs the number characters in the ENTITY->STRING in the font and height given by the entity area record. If it does exist, then in blocks 667 and 668, the program outputs the line, arrowhead, and extent bar pairs within the box defined by ENTITY->POINTS in the line style and line width given by the entity variable override record and also outputs the number characters in ENTITY->STRING in the font and height given by the entity variable override record. The program then goes to block 673.

If it is determined in block 663 that the entity type is not D, the program goes to block 669 and determines that the entity type is T, and goes to block 670 and determines whether the entity variable override record exists. If it does not, then the program outputs the text in ENTITY->STRING in the font and height given by the entity area record. If it does exist, then the program outputs the text in ENTITY->STRING in the font and height given by the entity variable override record as indicated in blocks 671 and 672. The program then goes to block 673.

In block 673, the program removes the ENTITY, which is the drawing entity just drawn, from DENTITYS. The program then returns to block 650. When the loop of block 650 has been completed for each entity, the program goes to block 674 and determines if DTYPE is equal to L, F, or S which means that the program has drawn one of the tables. If not, the program returns to block 618 but otherwise goes to block 675 and begins a loop through each table entry entity group EGROUP in TEGROUPS, and then goes to block 676 and outputs EGROUP->ID, which is the table entry identification number, at the top left corner of the entity group polygon. The program then goes to block 677 and "clips" and "transforms" EGROUP->POINTS by DX, DY, DH, DW, DSCALE, and DORIENT. The program then goes to block 678 and outputs an "X" at each registration point in EGROUP->POINTS, and then returns to block 675. When the loop of block 675 is done, the program returns to block 618. When the loop of block 618 is done, i.e., there are no more drawings to output, the program returns to the calling routine.

The above described automatic drawing system is suitable for general drafting of all types of drawings. The drawings may include lines of any inclination. The lines may be of a wide variety of styles and thicknesses.

Polygons, arcs, circles, curves, filled entities, arrowheads, character strings at any angle, hatchings of filled shapes in any of a wide variety of patterns, and any pre-created, stored symbols or new symbols created in the course of making the drawings may be included. Drafting parameters such as line types, line thicknesses, fill types, mirroring, copying and repetition, scaling, rotation, joining, trimming, line snap, point snap, and dimension rectification, all can be hand-sketched and abbreviated on the present layer of the drawing, rather than by use of keyboards, mouses, light pens, etc.

The character recognition techniques used in the above-mentioned GTX 5000 can be utilized to recognize character streams and convert them to ASCII code. The system also utilizes well-known vectorization, clustering, spatial relationships, pattern extraction, and dimensioning techniques, as implemented in the above mentioned GTX 5000 and U.S. patent 4,058,849. The described technique for hand-writing global command in pre-determined positions of the current layer allows the draftsman to easily, naturally, intuitively enter commands, without having to enter them via a keyboard, or mouse, or other menu selection technique. The above indicated command syntax is arbitrary and various other suitable command syntaxes can be devised.

Every draftsman uses a straight ruler, compass, french curve, a variety of templates and the drafting machine attached to his drafting table to position the coordinates and establish proper directions and lengths in horizontal, vertical and any other angular direction. The motion to move the ruler and identify the desired coordinate through the ruler scale takes a significant amount of time. The concept of using pre-printed grid paper such as sheet 10 of Fig. 2 effectively facilitates positioning of coordinates. For complex drawings, the draftsman can draw outlines and different entities or portions of the composite total drawing in a number of steps. With the use of the scaling commands available in the system, it is possible to draw a dense area of an image in an enlarged scale, which is much easier than would be the case if it were drawn to the same scale as the images in the composite drawing. The capability of the system to allow drawings to be folded and both sides scanned, and then to combine or merge the two sides or sections of the drawing in order to use a smaller scanner and a smaller printer reduces the size and cost of the system for many users.

The above described system also can be enhanced by providing a keypad or keyboard for fast entry of system parameter values and fast correction of text errors. The above described system can be enhanced for use by a more sophisticated user by providing both a graphics display and a mouse to expedite (1) the entry of system parameters without having to use menus to get scan, process, draw, and report parameters, or execute file commands, and (2) interactive display and correction of user and system errors.

It is expected that in the future, multiple entity types will be allowed within an individual active area, and the concepts of layers and editing can be applied to an already existing drawing for modification by converting it to the entities in layer structures of the type described. Also, the suite of global and local commands can be expanded to meet the needs of diverse users.

Attached Appendix "A" is a printout of pseudocode corresponding to the flowcharts of Figs. 3A-3B, 4A-4B, 5A-5D, 6A-6D, 7A-7H, and 8A-8D.

While the invention has been described with reference to a particular embodiment thereof, those skilled in the art will be able to make various modifications to the described embodiment without departing from the true spirit and scope thereof of the invention. For example, instead of defining an active area outside of which the global commands are written, global commands can be written anywhere on a first sheet as active area parameters and layer or editing data on the next sheet scanned. These second data sheets comprise larger layers in the drawing in the same fashion as active areas in Fig. 9. Another example is grouping curve points within an entity group rather than with chaining lines as in Fig. 10B.

APPENDIX "A"

```
MAIN Routine

   POWER on
   BUSY on / READY off
A: processor/memory bootstrap and test
   test failure?  yes:
      FAULT 00 on
      wait for RESET hit
      FAULT off
      goto A
B: load floppy disk program data
   disk i/o error?  yes:
      FAULT 01 on
      wait for RESET hit
      FAULT off
      goto B
C: reset system parameters and clear drawing data memory
   read switches to set input and output device parameters
D: BUSY off / READY on
   wait for button hit
   BUSY on / READY off
   RESET hit?  yes:
      goto C
   LOAD hit?  yes:
      load floppy disk drawing data
      disk i/o error?  yes: call FAULT
      goto D
   SCAN hit?  yes:
      call DOSCAN
      wait for STOP hit or DOSCAN return
      STOP hit?  yes: terminate scanning and call ABORT
      goto D
   DRAW hit?  yes:
      call DODRAW
      wait for STOP hit or DODRAW return
      STOP hit?  yes: terminate drawing and call ABORT
      goto D
   REPORT hit?  yes:
      RSYSTEM > 0?  yes: print system parameters
      RLAYER > 0?  yes: print layer catalog
      RDIRECT > 0?  yes: print disk directory
      RFAULT > 0? yes: print fault information
      wait for STOP hit or report termination
      STOP hit?  yes: terminate report
      goto D
   end
```

DOSCAN Routine

```
   scan page run lengths into RUNLENS
      input device error?  yes: call FAULT
   RGLEN > 0?  yes:
      fill RUNLENS gaps <= RGLEN
      erase RUNLENS runs <= RGLEN
   RUNLENS empty?  yes: call FAULT
   group connected RUNLENS into OBJECTS
   SOLEN > 0?  yes: remove OBJECTS with length <= SOLEN
   more than one OBJECT?  no: call FAULT
   convert first OBJECT character into STYPE
   STYPE one of "SPDRF"?  yes:
      convert and remove OBJECTS characters into COMCHARS
      OBJECTS empty?  no: call FAULT
      parse and remove COMCHARS into GCOMS
      COMCHARS empty?  no: call FAULT
      STYPE = "S"?  yes:
         decode and remove GCOMS to set scan parameters
         goto A
      STYPE = "P"?  yes:
         decode and remove GCOMS to set process parameters
         goto A
      STYPE = "D"?  yes:
         decode and remove GCOMS to set draw parameters
         goto A
      STYPE = "R"?  yes:
         decode and remove GCOMS to set report parameters
         goto A
      STYPE = "F"?  yes:
         decode and remove GCOMS to perform disk i/o
         GCOMS empty?  no: call FAULT
         disk i/o error?  yes: call FAULT
         return
A: GCOMS empty?  no: call FAULT
   update system parameter file
   disk i/o error?  yes: call FAULT
   return
```

DOSCAN Routine page 2

```
   move OBJECTS completely outside active area into OUTOBJS
   move OBJECTS completely inside active area into INOBJS
   OBJECTS empty?  no: call FAULT
   convert and remove OUTOBJS characters into COMCHARS
   OUTOBJS empty?  no: call FAULT
   parse and remove COMCHARS into GCOMS
   COMCHARS empty?  no: call FAULT
   STYPE = first GCOM->COM
   STYPE = "L" or "E"?  no: call FAULT
   LTYPE = first GCOM->ARG
   LTYPE = "D"?  yes:
      LAYERS = DLAYERS
      goto B
   LTYPE = "L"?  yes:
      LAYERS = LLAYERS
      goto B
   LTYPE = "F"?  yes:
      LAYERS = FLAYERS
      goto B
   LTYPE = "S"?  yes:
      LAYERS = SLAYERS
      goto B
   call FAULT
B: STYPE = "L"?  yes:
      call DOLAYER
      return
   STYPE = "E"?  yes:
      call DOEDIT
      return
   end
```

DOLAYER Routine

```
decode and remove remaining first GCOM->ARGS into LID,
   ATYPE and AERASE and remaining GCOMS into AREA variables
GCOMS empty?  yes: call FAULT
translate, size and rotate INOBJS by AREA->X, AREA->Y,
   AREA->SCALE and AREA->ORIENT
INOBJS completely inside drawing?  no: call FAULT
find LAYER in LAYERS with LAYER->ID = LID
LAYER found?  no:
   create new LAYER and append to LAYERS
   LAYER->ID = LID
   LAYER->TYPE = LTYPE
yes: AERASE = "*"?  yes:
   remove all ENTITYS from LAYER
   remove all AREAS with AREA->LAYER = LAYER
create new AREA and append to AREAS
AREA->LAYER = LAYER
AREA->TYPE = ATYPE
ATYPE = "L"?  yes:
   convert and remove INOBJS lines into AREA->LINES
   INOBJS empty?  no: call FAULT
   for each LINE in AREA->LINES:
      create new ENTITY and append to LAYER
      ENTITY->AREA = AREA
      ENTITY->POINTS = LINE->POINTS
      remove LINE from AREA->LINES
   return
ATYPE = "C"?  yes:
   convert and remove INOBJS lines into AREA->LINES
   INOBJS empty?  no: call FAULT
   extract and remove AREA->LINES crosses into AREA->POINTS
   extract and remove AREA->POINTS and AREA->LINES chains
      (at least two points connected by one line) into CHAINS
   AREA->LINES empty?  no: call FAULT
   AREA->POINTS empty?  no: call FAULT
   for each CHAIN in CHAINS:
      create new ENTITY and append to LAYER
      ENTITY->AREA = AREA
      ENTITY->POINTS = CHAIN->POINTS
      #(CHAIN->POINTS) = 2?  yes:
         generate third circle POINT and add after
            first ENTITY->POINT
         goto A
      no: #(CHAIN->POINTS) = #(CHAIN->LINES)?  yes:
      A: append first ENTITY->POINT to ENTITY->POINTS
      remove CHAIN from CHAINS
   return
```

DOLAYER Routine page 2

```
ATYPE = "D"? yes:
   convert and remove INOBJS characters into AREA->CHARS
   convert and remove INOBJS lines into AREA->LINES
   convert and remove INOBJS filled triangles into AREA->BLOBS
   INOBJS empty? no: call FAULT
   extract and remove colinear adjacent AREA->CHARS
      into AREA->STRINGS
   extract and remove AREA->STRINGS, AREA->LINES and
      AREA->BLOBS dimension lines, arrowheads and extent bars,
      and AREA->STRINGS numbers forming colinear adjacent
      "|<--- number --->|" patterns into DIMENS
   AREA->LINES empty? no: call FAULT
   AREA->STRINGS empty? no: call FAULT
   AREA->BLOBS empty? no: call FAULT
   for each DIMEN in DIMENS:
      create new ENTITY and append to LAYER
      ENTITY->AREA = AREA
      ENTITY->POINTS = DIMEN->LINES->POINTS
      ENTITY->STRINGS = DIMEN->STRING
      remove DIMEN from DIMENS
   return
ATYPE = "T"? yes:
   convert and remove INOBJS characters into AREA->CHARS
   INOBJS empty? no: call FAULT
   extract and remove colinear adjacent AREA->CHARS
      into AREA->STRINGS
   for each STRING in AREA->STRINGS:
      create new ENTITY and append to LAYER
      ENTITY->AREA = AREA
      ENTITY->STRING = STRING
      remove STRING from AREA->STRINGS
   return
call FAULT
end
```

DOEDIT Routine page 1

```
decode and remove first GCOM->ARGS into LIDS and
   remaining GCOMS into EDIT variables
GCOMS empty?  no: call FAULT
translate, size and rotate INOBJS by EDIT->X, EDIT->Y,
   EDIT->SCALE, EDIT->ORIENT
INOBJS completely inside drawing?  no: call FAULT
convert and remove INOBJS characters into EDIT->CHARS
extract and remove colinear adjacent EDIT->CHARS
   into EDIT->STRINGS
parse and remove EDIT->STRINGS into ECOMS
convert and remove INOBJS lines into EDIT->LINES
extract and remove EDIT->LINES crosses into EDIT->POINTS
convert and remove INOBJS filled circles into EDIT->BLOBS
INOBJS empty?  no: call FAULT
extract EDIT->LINES minimum area polygons into POLYS
POLYS empty?  yes:
   EDIT->STRINGS empty?  no: call FAULT
   EDIT->LINES empty?  no: call FAULT
   EDIT->BLOBS empty?  no: call FAULT
   goto A
for each POLY in POLYS:
   create new EGROUP and append to EGROUPS
   EGROUP->POLY = POLY
   remove POLY from POLYS
for each STRING in EDIT->STRINGS:
   find nearest EGROUP->POLY in EGROUPS
   decode and remove STRING->CHARS into EGROUP->ENC,
      EID and EGROUP->TAB
   STRING->CHARS empty?  no: call FAULT
   EID = 0?  no:
      EID assigned?  yes: call FAULT
      EGROUP->ID = 0?  no: call FAULT
      EGROUP->ID = EID
   EGROUP->TAB = "#"?  yes:
      LTYPE = "D" or EID = 0?  yes: call FAULT
   remove STRING from EDIT->STRINGS
for each POINT in EDIT->POINTS:
   for each EGROUP in EGROUPS:
      POINT inside EGROUP->POLY?  yes:
         #(EGROUP->POINTS) = 3?  yes: call FAULT
         append POINT to EGROUP->POINTS
         remove POINT from EDIT->POINTS
         break
for each BLOB in EDIT->BLOBS
   for each EGROUP in EGROUPS:
      BLOB inside EGROUP->POLY?  yes:
         EGROUP->FIL = 1
         remove BLOB from EDIT->BLOBS
         break
EDIT->BLOBS empty?  no: call FAULT
```

DOEDIT Routine page 2

```
   for each LAYER in LAYERS:
      LIDS empty?  no:
         find and remove LID = LAYER->ID in LIDS
         LID found?  yes:
      yes: for each ENTITY in LAYER->ENTITYS:
            for each EGROUP in EGROUPS:
               generated ENTITY completely inside EGROUP->POLY or
               intersects EGROUP->POLY and EGROUP->ENC = 0?  yes:
                  append ENTITY to EGROUP->ENTITYS
                  tag all ENTITY->POINTS inside EGROUP->POLY
                  EGROUP->ENC = "0"?  yes: break
   LIDS empty?  no: call FAULT
   for each EGROUP in EGROUPS:
      EGROUP empty?  yes: call FAULT
      EGROUP->FIL = 1?  yes:
         for each ENTITY in EGROUP->ENTITYS:
            ENTITY->AREA->TYPE = "C" and
               first ENTITY->POINT = last ENTITY->POINT?  yes:
               ENTITY->FILL = null?  yes: create ENTITY->FILL
               ENTITY->FILL->FSTYLE = EDIT->FSTYLE
               ENTITY->FILL->FWIDTH = EDIT->FWIDTH
A: for each ECOM in ECOMS:
      ECOM->COM one of "EMBJAGLFT"?  yes:
         EGROUPS empty?  yes: call FAULT
         find nearest EGROUP->POLY in EGROUPS
         ECOM->EGROUP = EGROUP
      ECOM->COM = "M" or "R"?  yes:
         EDIT->LINES empty?  yes: call FAULT
         find nearest LINE in EDIT->LINES
         ECOM->LINE = LINE
      ECOM->COM = "C", "R" or "S" yes:
         decode first ECOM->ARG into EID
         EID = 0?  yes: call FAULT
         ECOM->COM = "C" or "R"?  yes:
            find EGROUP in EGROUPS with EGROUP->ID = EID
         no:
            find EGROUP in SEGROUPS with EGROUP->ID = EID
         EGROUP found?  no: call FAULT
         ECOM->EGROUP = EGROUP
         EDIT->POINTS empty?  yes: call FAULT
         find nearest #(ECOM->EGROUP->POINTS) POINTS
            in EDIT->POINTS
         POINTS found?  no: call FAULT
         ECOM->POINTS = POINTS
   remove all assigned EDIT->LINES
   EDIT->LINES empty?  no: call FAULT
   remove all assigned EDIT->POINTS
   EDIT->POINTS empty?  no: call FAULT
```

```
DOEDIT Routine                                                    page 3

   for each ECOM in ECOMS:
      ECOM->COM = "E"?  yes:
         EGROUP->DEL = 1
         goto B
      ECOM->COM = "B"?  yes:
         for each ENTITY in ECOM->EGROUP->ENTITYS
            ENTITY->AREA->TYPE = "C"?  yes:
               duplicate each single tagged ENTITY->POINT
         goto B
      ECOM->COM = "J"?  yes:
         for each ENTITY in ECOM->EGROUP->ENTITYS
            ENTITY->POVER = null?  yes: create ENTITY->POVER
            first ECOM->COM->ARG = "N"?  yes:
               ENTITY->POVER->JOIN = -1
            no: ENTITY->POVER->JOIN = 1
         goto B
      ECOM->COM = "A"?  yes:
         for each ENTITY in ECOM->EGROUP->ENTITYS
            ENTITY->POVER = null?  yes: create ENTITY->POVER
            first ECOM->COM->ARG = "N"?  yes:
               ENTITY->POVER->LSNAP = -1
            no: ENTITY->POVER->LSNAP = 1
         goto B
      ECOM->COM = "G"?  yes:
         for each ENTITY in ECOM->EGROUP->ENTITYS
            ENTITY->POVER = null?  yes: create ENTITY->POVER
            first ECOM->COM->ARG = "N"?  yes:
               ENTITY->POVER->PSNAP = -1
            no: call FAULT
         goto B
      ECOM = "L", "F" or "T"?  yes:
         decode ECOM->ARGS into EID and ENUM
         ECOM decode error?  yes: call FAULT
         ECOM->COM = "L"?  yes:
            for each ENTITY in ECOM->EGROUP->ENTITYS
               ENTITY->AREA->TYPE = "L", "C" or "D"?  yes:
                  ENTITY->VOVER = null? yes: create ENTITY->VOVER
                  ENTITY->VOVER->LSTYLE = EID
                  ENTITY->VOVER->LWIDTH = ENUM
            goto B
         ECOM->COM = "F"?  yes:
            for each ENTITY in ECOM->EGROUP->ENTITYS
               ENTITY->FILL = null?  no:
                  ENTITY->FILL->FSTYLE = EID
                  ENTITY->FILL->FWIDTH = ENUM
            goto B
         ECOM->COM = "T"?  yes:
            for each ENTITY in ECOM->EGROUP->ENTITYS
               ENTITY->AREA->TYPE = "D" or "T"?  yes:
                  ENTITY->VOVER = null? yes: create ENTITY->VOVER
                  ENTITY->VOVER->FONT = EID
                  ENTITY->VOVER->HEIGHT = ENUM
         B: remove ECOM from ECOMS
```

DOEDIT Routine page 4

```
   LTYPE = "L"?  yes: TEGROUPS = LEGROUPS
   LTYPE = "F"?  yes: TEGROUPS = FEGROUPS
   LTYPE = "S"?  yes: TEGROUPS = SEGROUPS
   for each EGROUP in EGROUPS:
      EGROUP->TAB = "#"?  yes:
         find and remove EGROUP in TEGROUPS with same EGROUP->ID
         EGROUP->DEL = 1?  no: append EGROUP to TEGROUPS
   for each ECOM in ECOMS:
      ECOM->COM = "M"?  yes:
         for each ENTITY in ECOM->EGROUP->ENTITYS:
            create new ENTITY and append to ENTITY->AREA->LAYER
            mirror ENTITY about ECOM->LINE into new ENTITY
         goto C
      ECOM->COM = "C" or "S"?  yes:
         decode ECOM->ARGS into ESCALE and EORIENT
         ECOM decode error?  yes: call FAULT
         for each ENTITY in ECOM->EGROUP->ENTITYS:
            create new ENTITY and append to ENTITY->AREA->LAYER
            transform ENTITY by ESCALE and EORIENT into
               new ENTITY so ENTITY->POINTS match ECOM->POINTS
         goto C
      ECOM->COM = "R"?  yes:
         decode ECOM->ARGS into RCOUNT, ESCALE, EORIENT
            RSCALE and RORIENT
         ECOM decode error?  yes: call FAULT
         generate RCOUNT equispaced RPOINTS along line parallel
            to ECOM->LINE through top left ECOM->POINTS
         for each ENTITY in ECOM->EGROUP->ENTITYS:
            for each POINT in RPOINTS:
               create new ENTITY and append
                  to ENTITY->AREA->LAYER
               translate ECOM->POINTS to POINT
               transform ENTITY by ESCALE and EORIENT into
                  new ENTITY so ENTITY->POINTS match ECOM->POINTS
               ESCALE = ESCALE + RSCALE
               EORIENT = EORIENT + RORIENT
         goto C
      call FAULT
   C: remove ECOM from ECOMS
   ECOMS empty?  no: call FAULT
   for each EGROUP in EGROUPS:
      EGROUP->DEL = 1? yes:
         for each ENTITY in EGROUP->ENTITYS:
            find and remove ENTITY in ENTITY->AREA->LAYER
            EGROUP->TAB = "#" or LTYPE = "D"?  no:
               for each EGROUP in TEGROUPS:
                  find and remove ENTITY in EGROUP
                  EGROUP empty?  yes: remove EGROUP from TEGROUPS
      remove EGROUP from EGROUPS
   return
   end
```

DODRAW Routine page 1

```
   DTYPE = "D"?  yes:
      LAYERS = DLAYERS
      goto A
   DTYPE = "L"?  yes:
      LAYERS = LLAYERS
      TEGROUPS = LEGROUPS
      goto A
   DTYPE = "F"?  yes:
      LAYERS = FLAYERS
      TEGROUPS = FEGROUPS
      goto A
   DTYPE = "S"?  yes:
      LAYERS = SLAYERS
      TEGROUPS = SEGROUPS
      goto A
   call FAULT
A: for each LIDS in DLIDS:
   for each LAYER in LAYERS:
      LIDS empty?  no:
         find LID in LIDS with LID = LAYER->ID
         LID found?  yes:
      yes: for each ENTITY in LAYER->ENTITYS:
            append ENTITY to DENTITYS
   for each ENTITY in DENTITYS:
      JTDIST > 0?  yes:
         cluster all ENTITY->POINTS except tagged POINTS
            with ENTITY->POVER->JOIN not = 0 within JTDIST
            of each other into CLUSTERS
   for each ENTITY in DENTITYS:
      for each tagged POINT in ENTITY with ENTITY->POVER->JOIN = 1:
         find nearest ENTITY->POINT and CLUSTER->EPOINT->POINT
            ENTITY->POINT = CLUSTER->EPOINT->POINT?  yes:
               append POINT to CLUSTER
            no: create new CLUSTER of POINT and ENTITY->POINT
   for each CLUSTER in CLUSTERS:
      compute CLUSTER centroid
      for each POINT in CLUSTER->EPOINTS:
         POINT = CLUSTER centroid
   for each ENTITY in DENTITYS:
      OLDEV > 0?  yes:
         snap all linear ENTITY->POINTS endpoints
            except tagged POINTS with ENTITY->POVER->LSNAP not = 0
            within OLDEV of orthogonal to horizontal or vertical
      ENTITY->POVER->LSNAP = 1?  yes:
         snap all tagged linear ENTITY->POINTS endpoints
            to closest horizontal or vertical
      GCLEN > 0?  yes:
         snap all ENTITY->POINTS endpoints except tagged POINTS
            with ENTITY->POVER->PSNAP = -1 to nearest grid point
      RECT > 0 and ENTITY->AREA->TYPE = "L" or "C"?  yes:
         extract dimensions from nearest ENTITYS in DENTITYS with
            ENTITY->AREA->TYPE = "D" and rectify ENTITY->POINTS
         rectification error?  yes: call FAULT
```

DODRAW Routine page 2

```
for each ENTITY in DENTITYS:
   clip and transform ENTITY->POINTS and
      ENTITY->STRING->BOX->POINTS by DX, DY, DH, DW,
      DSCALE and DORIENT
   ENTITY->AREA->TYPE = "L"?  yes:
      ENTITY->VOVER null?  yes:
         output ENTITY->POINTS line
            in ENTITY->AREA->LSTYLE/LWIDTH
      no: output ENTITY->POINTS line
            in ENTITY->VOVER->LSTYLE/LWIDTH
      goto B
   ENTITY->AREA->TYPE = "C"?  yes:
      for each set of ENTITY->POINTS separated by
         endpoints and duplicated points:
         ENTITY->VOVER null?  yes:
         output ENTITY->POINTS set spline
            in ENTITY->AREA->LSTYLE/LWIDTH
      no: output ENTITY->POINTS set spline
            in ENTITY->VOVER->LSTYLE/LWIDTH
      ENTITY->FILL = null? no:
         output ENTITY->POINTS fill
            in ENTITY->FILL->FSTYLE/FWIDTH
      goto B
   ENTITY->AREA->TYPE = "D"?  yes:
      ENTITY->VOVER null?  yes:
         output line, arrowhead and extent bar pairs within
            ENTITY->POINTS box in ENTITY->AREA->LSTYLE/LWIDTH
         output ENTITY->STRING in ENTITY->AREA->FONT/HEIGHT
      no: output line, arrowhead and extent bar pairs within
            ENTITY->POINTS box in ENTITY->VOVER->LSTYLE/LWIDTH
         output ENTITY->STRING in ENTITY->VOVER->FONT/HEIGHT
      goto B
   ENTITY->AREA->TYPE = "T"?  yes:
      ENTITY->VOVER null?  yes:
         output ENTITY->STRING in ENTITY->AREA->FONT/HEIGHT
      no: output ENTITY->STRING in ENTITY->VOVER->FONT/HEIGHT
   B: remove ENTITY from DENTITYS
DTYPE = "L", "F" or "S"?  yes:
   for each EGROUP in TEGROUPS:
      output EGROUP->ID at top left corner of EGROUP->POLY
      clip and transform EGROUP->POINTS by DX, DY, DH, DW,
         DSCALE and DORIENT
      output "X" at each POINT in EGROUP->POINTS
return
end
```

WHAT IS CLAIMED IS:

1. A method of producing a drawing, comprising the steps of:

(a) drawing representations of a plurality of similar images of a first type on a first sheet;

(b) scanning the first sheet by means of an input device to produce first digital data representative of the images of the first type;

(c) operating on the first digital data to form first software objects representing the images of the first type, respectively, and storing the first software objects in a first software layer;

(d) drawing representations of a plurality of similar images of a second type on a second sheet;

(e) scanning the second sheet by means of the input device to produce second digital data representative of the images of the second type;

(f) operating on the second digital data to form second software objects representing the images of the second type, respectively, and storing the second software objects in a second software layer;

(g) operating on data from the first and second software layers to output or store a first composite drawing or table including the images of the first type and the images of the second type.

2. The method of Claim 1 repeating steps (e) through (g) for additional similar images of an additional type, to obtain additional software objects in an additional software layer, and an additional composite drawing including the images of the first, second, and additional types.

3. The method of Claim 1 wherein the images of the first type are straight lines and the representations thereof include rough approximations thereof, and the images of the second type are curves and the representations thereof include a set of points on the curves.

4. The method of Claim 3 wherein the curves include a circle and the representations of the images of the second type include a diameter of the circle.

5. The method of Claim 3 wherein the curves include an arc and the representations of the images of the second type include two endpoints of the arc and a midpoint of the arc grouped together.

6. The method of Claim 3 wherein the representations of the images of the second type include sets of points grouped together to define curves which include spline approximations.

7. The method of Claim 6 wherein the sets of points have the same first and last point to define closed shapes, respectively.

8. The method of Claim 6 wherein the sets of points include duplicate pairs of points to indicate separate curves or lines connected at the duplicate points.

9. The method of Claim 2 wherein the images of a third type include dimension entities, including dimension lines, extent bars, arrowheads and numbers.

10. The method of Claim 2 wherein the images of a fourth type includes text character strings.

11. The method of Claim 1 wherein the first sheet contains images in any selected position, scale, and/or orientation with respect to a portion of the first composite drawing represented by the first software layer.

12. The method of Claim 1 including augmenting the first software layer by

drawing representations of additional images of the first type on an additional sheet;

scanning the additional sheet by means of the scanner to produce additional digital data representative of the additional images of the first type;

operating on the additional digital data to augment or replace the first software objects, and storing the augmented or replaced first software objects in the first software layer.

13. The method of Claim 1 wherein the first sheet includes an active area and an inactive area, the active area containing the images of the first type, the inactive area containing global commands.

14. The method of Claim 13 wherein an additional sheet includes an active area and an inactive area, the method including drawing editing directives including local commands and spatial indicators in the active area of the additional sheet that modify the first and/or software layers.

15. The method of Claim 1 including modifying scanning, processing, or drawing parameters by writing them as global commands on an additional sheet, scanning the additional sheet by means of the input device to modify the parameters.

16. The method of Claim 9 wherein the images of the third type are dimension entities located inside the active area of the third sheet, the method including determining that the images of the third type are dimension entities located in the active area of the third sheet, extracting colinear adjacent characters into strings, extracting lines, and extracting arrowheads, creating an entity, appending that entity to a third software layer, calling and executing a clustering program to cluster the strings, lines, and arrowheads into dimension entities, and for each cluster, calling and executing a character recognition program to recognize the characters, extracting a pattern consisting of number strings within a pair of opposing dimension lines, arrowheads, and extent bars, storing digital codes representing the number string and the endpoints of the extent bars in the entity, respectively, encoding the digital codes representing the number string into numbers of units for executing a dimension rectification program, and decoding the digital codes to output dimension information during step (g).

17. The method of Claim 10 wherein the images of the fourth type are characters located inside the active area of the fourth sheet, the method including determining that the images of the fourth type are characters located in the active area of the fourth sheet, extracting colinear adjacent characters into strings, and for each string, creating an entity, appending that entity to a fourth software layer, and calling and executing a character recognition program to recognize the characters, storing digital codes representing the characters in the entity, respectively, and decoding the digital codes to output the characters during step (g).

18. The method of Claim 13 wherein the images include characters, the method including calling and executing a character recognition program to recognize the characters, parsing the characters into commands and their arguments, and interpreting the commands and their arguments to set parameters and variables and execute operations.

19. The method of Claim 1 wherein the first and second sheets have grid patterns thereon which cannot be detected by the input device, the method including aligning the first and second sheets to the input device in precisely the same way, steps (a) and (d) including drawing the representations of the images of the first and second types on selected grid line and intersections thereof of the first and second grid sheets, respectively, and steps (b) and (e) include detecting the images of the first and second types without

detecting grid lines of the first and second sheets, respectively.

20. The method of Claim 14 wherein the editing directive images include characters, lines, curves, and blobs in an active area of an edit sheet, the method including determining that the editing directive images are inside the active area of the edit sheet, extracting colinear adjacent characters into strings, lines, points, blobs, and grouping boundaries in the editing directive images, calling and executing a character recognition program to recognize those characters in those strings, storing digital codes representing those character strings, parsing those strings into local commands and their arguments or grouping boundary information.

21. The method of Claim 20 including clustering the strings, lines, points, blobs, and grouping boundaries to associate interior registration points and fill blobs with surrounding grouping boundaries, grouping boundary information strings with nearby grouping boundaries, and position markers, axis lines, and grouping boundaries with editing command strings.

22. The method of Claim 21 including searching the first and second software layers for entities that fall completely within and/or intersect the grouping boundaries.

23. The method of Claim 22 including interpreting the local commands and their arguments, or grouping boundary information, or blob indicators to edit the entities in the first and/or second software layers.

24. The method of Claim 23 wherein editing includes erasing, mirroring, copying, or repeating entities in a grouping boundary, or insertion of a symbol table entry, overriding default or previously set entity attributes including joining points, snapping lines orthogonally, snapping points to a grid, setting line width, text height, line and fill styles to line and fill style table entries, or text font to a text font table entry, establishing criteria for including entities in a grouping boundary, a grouping boundary identification, setting flags including table entry, shape filling and curve breaking.

25. The method of Claim 1 wherein the table includes entries representing line styles, fill styles, or symbols.

AMENDED CLAIMS

[received by the International Bureau on 23 August 1991 (23.08.91); original claims 1,15,18 and 22 amended; other claims unchanged (10 pages)]

1. A method of producing a drawing, comprising the steps of:

(a) manually drawing a plurality of images of a first type on a first sheet, the images of the first type being readily vectorized by a first vectorizing routine;

(b) scanning the first sheet by means of an input device to produce first digital data representative of the manually drawn images of the first type;

(c) vectorizing the first digital data by operating on the first digital data with the first vectorizing routine to form first software objects representing the images of the first type of greater accuracy than the manually drawn images of the first type, respectively, and storing the first software objects in a first software layer;

(d) manually drawing representations of a plurality of images of a second type on a second sheet, the images of the second type being readily vectorized by a second vectorizing routine;

(e) scanning the second sheet by means of the input device to produce second digital data representative of the images of the second type;

(f) vectorizing the second digital data by operating on the second digital data with the second vectorizing routine to form second software objects representing the images of the second type of greater accuracy than the manually drawn images of the second type, respectively, and storing the second software objects in a second software layer;

(g) operating on data from the first and second software layers by means of a processor to output or store data representing a first composite drawing including the images of the first type and the images of the second type.

2. The method of Claim 1 repeating steps (e) through (g) for additional similar images of an additional type, to obtain additional software objects in an additional software layer, and an additional composite drawing including the images of the first, second, and additional types.

3. The method of Claim 1 wherein the images of the first type are straight lines and the representations thereof include rough approximations thereof, and the images of the second type are curves and the representations thereof include a set of points on the curves.

4. The method of Claim 3 wherein the curves include a circle and the representations of the images of the second type include a diameter of the circle.

5. The method of Claim 3 wherein the curves include an arc and the representations of the images of the second type include two endpoints of the arc and a midpoint of the arc grouped together.

6. The method of Claim 3 wherein the representations of the images of the second type include sets of points grouped together to define curves which include spline approximations.

7. The method of Claim 6 wherein the sets of points have the same first and last point to define closed shapes, respectively.

8. The method of Claim 6 wherein the sets of points include duplicate pairs of points to indicate separate curves or lines connected at the duplicate points.

9. The method of Claim 2 wherein the images of a third type include dimension entities, including dimension lines, extent bars, arrowheads and numbers.

10. The method of Claim 2 wherein the images of a fourth type includes text character strings.

11. The method of Claim 1 wherein the first sheet contains images in any selected position, scale, and/or orientation with respect to a portion of the first composite drawing represented by the first software layer.

12. The method of Claim 1 including augmenting the first software layer by

drawing representations of additional images of the first type on an additional sheet;

scanning the additional sheet by means of the scanner to produce additional digital data representative of the additional images of the first type;

operating on the additional digital data to augment or replace the first software objects, and storing the augmented or replaced first software objects in the first software layer.

13. The method of Claim 1 wherein the first sheet includes an active area and an inactive area, the active area containing the images of the first type, the inactive area containing global commands.

14. The method of Claim 13 wherein an additional sheet includes an active area and an inactive area, the method including drawing editing directives including local commands and spatial indicators in the active area of the additional sheet that modify the first and/or software layers.

15. The method of Claim 1 including modifying scanning, processing, or drawing parameters by writing global commands on an additional sheet, and scanning the additional sheet by means of the input device to modify the parameters.

16. The method of Claim 9 wherein the images of the third type are dimension entities located inside the active area of the third sheet, the method including determining that the images of the third type are dimension entities located in the active area of the third sheet, extracting colinear adjacent characters into strings, extracting lines, and extracting arrowheads, creating an entity, appending that entity to a third software layer, calling and executing a clustering program to cluster the strings, lines, and arrowheads into dimension entities, and for each cluster, calling and executing a character recognition program to recognize the characters, extracting a pattern consisting of number strings within a pair of opposing dimension lines, arrowheads, and extent bars, storing digital codes representing the number string and the endpoints of the extent bars in the entity, respectively, encoding the

digital codes representing the number string into numbers of units for executing a dimension rectification program, and decoding the digital codes to output dimension information during step (g).

17. The method of Claim 10 wherein the images of the fourth type are characters located inside the active area of the fourth sheet, the method including determining that the images of the fourth type are characters located in the active area of the fourth sheet, extracting colinear adjacent characters into strings, and for each string, creating an entity, appending that entity to a fourth software layer, and calling and executing a character recognition program to recognize the characters, storing digital codes representing the characters in the entity, respectively, and decoding the digital codes to output the characters during step (g).

18. The method of Claim 13 wherein the images include characters, the method including calling and executing a character recognition program to recognize the characters, parsing the characters into commands and arguments of the commands, and interpreting the commands and the arguments to set parameters and variables and execute operations.

19. The method of Claim 1 wherein the first and second sheets have grid patterns thereon which cannot be detected by the input device, the method including aligning the first and second sheets to the input device in precisely the same way, steps (a) and (d) including drawing the representations of the images of the first and second types on selected grid line and intersections thereof of the first and second grid sheets, respectively, and steps (b) and (e) include detecting the images of the first and second types without detecting grid lines of the first and second sheets, respectively.

20. The method of Claim 14 wherein the editing directive images include characters, lines, curves, and blobs in an active area of an edit sheet, the method including determining that the editing directive images are inside the active area of the edit sheet, extracting colinear adjacent characters into strings, lines, points, blobs, and grouping boundaries in the editing directive images, calling and executing a character recognition program to recognize those characters in those strings, storing digital codes representing those character strings, parsing those strings into local commands and their arguments or grouping boundary information.

21. The method of Claim 20 including clustering the strings, lines, points, blobs, and grouping boundaries to associate interior registration points and fill blobs with surrounding grouping boundaries, grouping boundary

information strings with nearby grouping boundaries, and position markers, axis lines, and grouping boundaries with editing command strings.

22. The method of Claim 21 including searching the first and second software layers for entities that fall completely within the grouping boundaries for entities that intersect the grouping boundaries.

23. The method of Claim 22 including interpreting the local commands and their arguments, or grouping boundary information, or blob indicators to edit the entities in the first or second software layers.

24. The method of Claim 23 wherein editing includes erasing, mirroring, copying, or repeating entities in a grouping boundary, or insertion of a symbol table entry, overriding default or previously set entity attributes including joining points, snapping lines orthogonally, snapping points to a grid, setting line width, text height, line and fill styles to line and fill style table entries, or text font to a text font table entry, establishing criteria for including entities in a grouping boundary, a grouping boundary identification, setting flags including table entry, shape filling and curve breaking.

25. The method of Claim 1 wherein the table includes entries representing line styles, fill styles, or symbols.

26. A system for producing a drawing, comprising in combination:

(a) a first sheet having thereon drawing representations of a plurality of images of a first type;

(b) means for scanning the first sheet to produce first digital data representative of the images of the first type;

(c) means for operating on the first digital data to form first software objects representing the images of the first type, respectively;

(d) means for storing the first software objects in a first software layer;

(e) a second sheet having thereon drawing representations of a plurality of images of a second type;

(f) means for scanning the second sheet to produce second digital data representative of the images of the second type;

(g) means for operating on the second digital data to form second software objects representing the images of the second type, respectively;

(h) means for storing the second software objects in a second software layer;

(i) means for operating on data from the first and second software layers to output or store a first composite drawing or table including the images of the first type and the images of the second type.

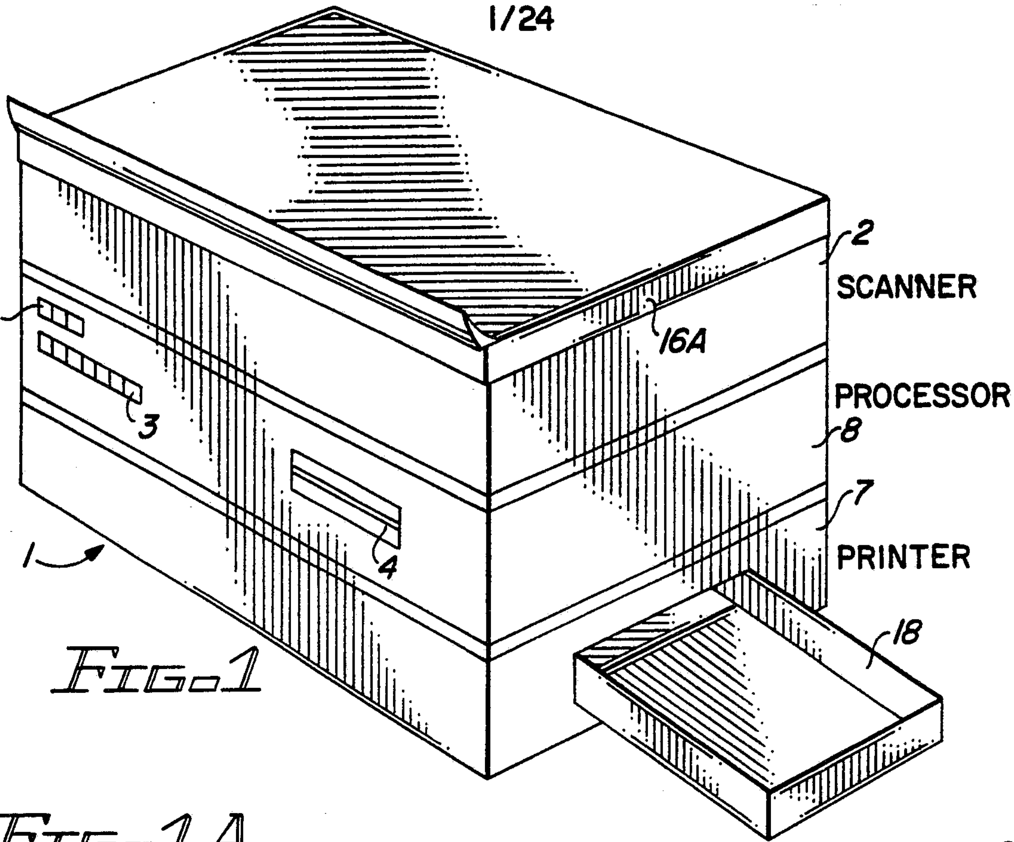
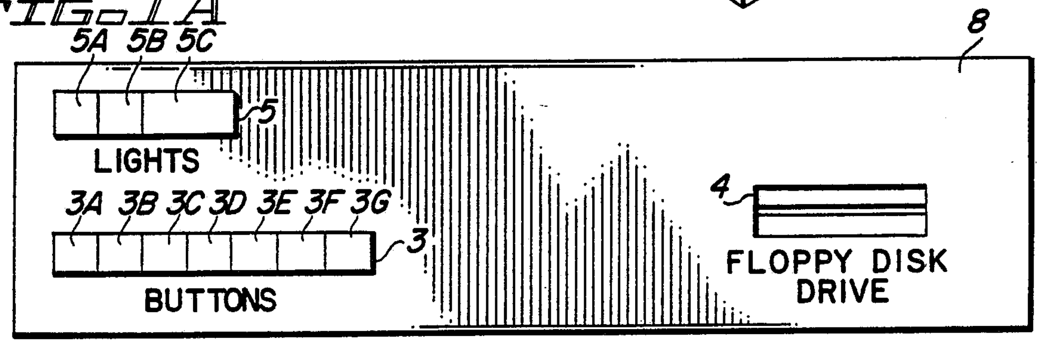


FIG. 1

FIG. 1A



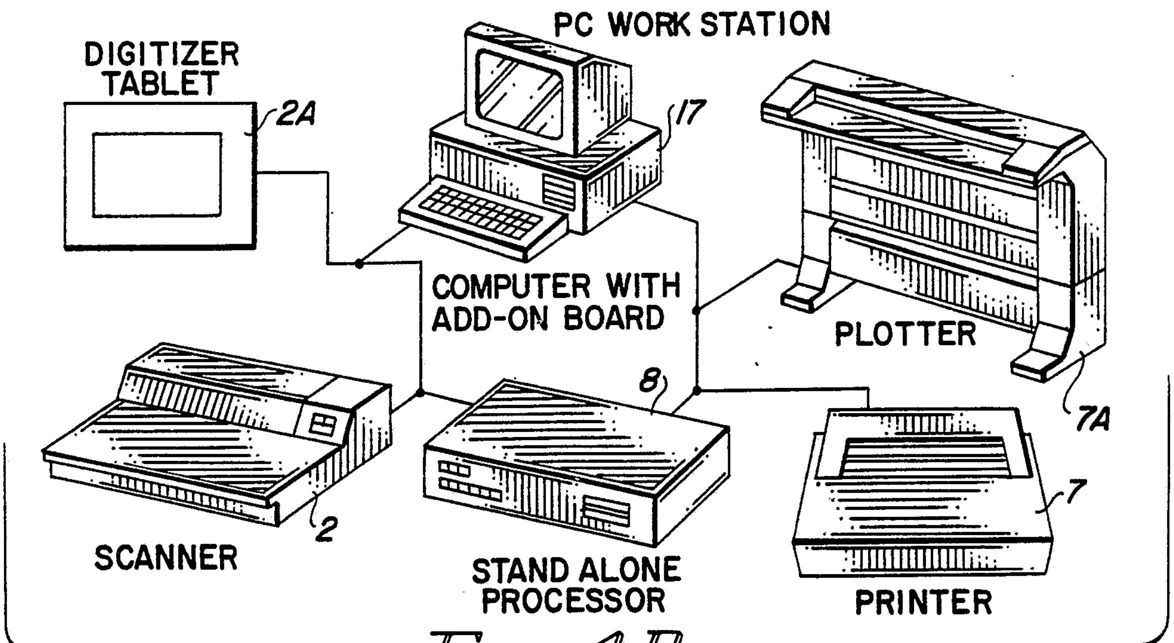


FIG. 1B

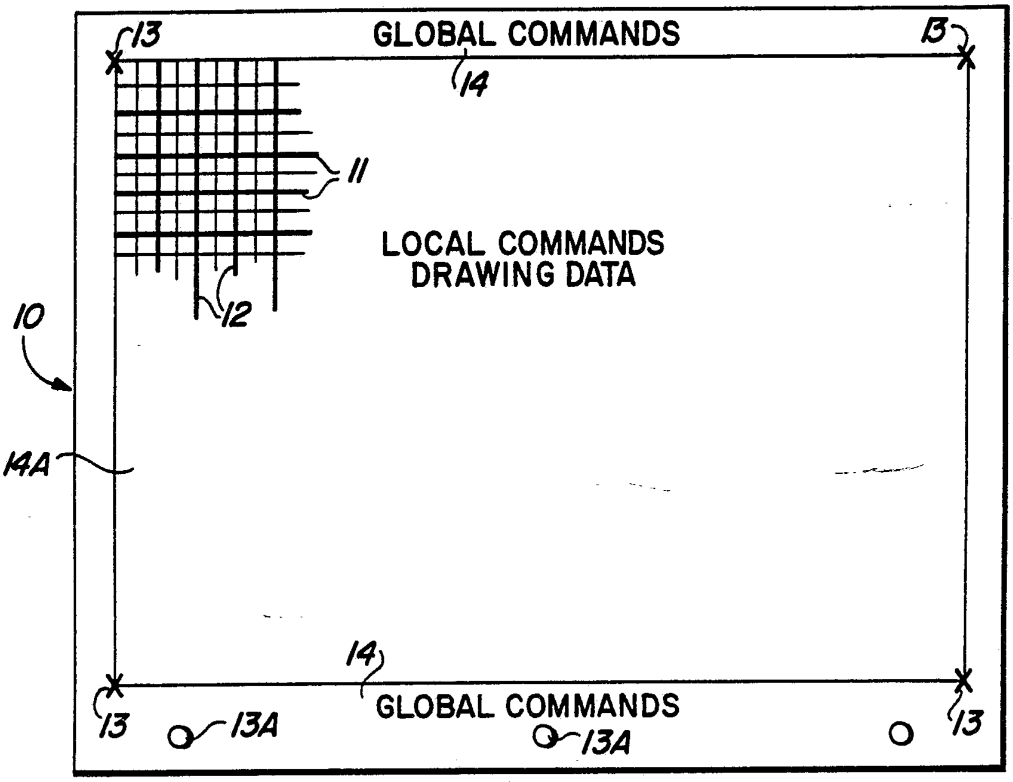


FIG. 2

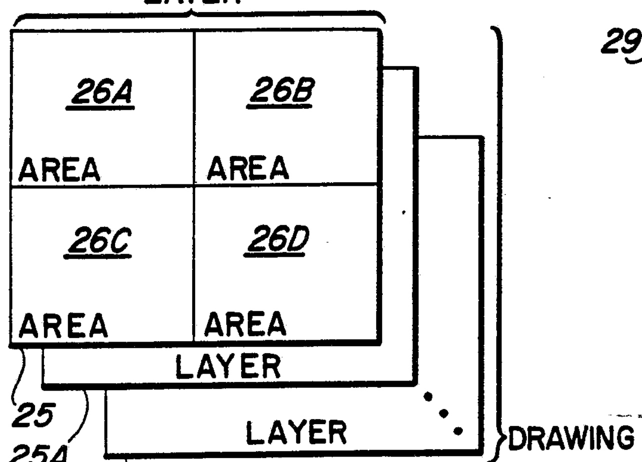


FIG. 9

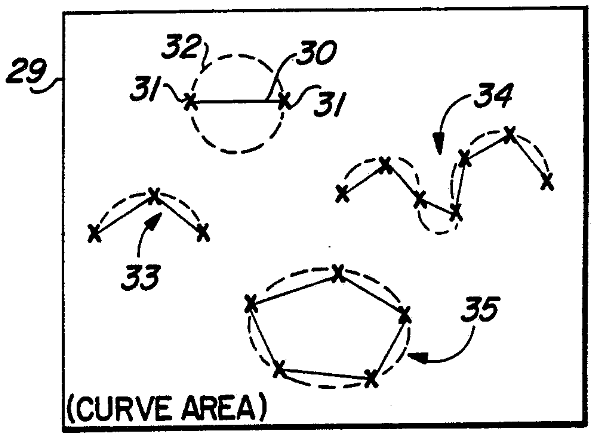


FIG. 10B

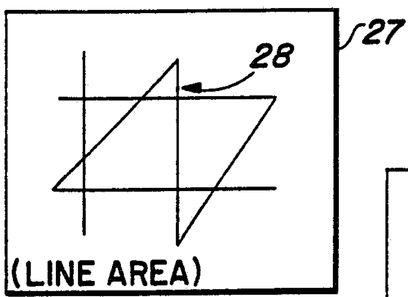


FIG. 10A

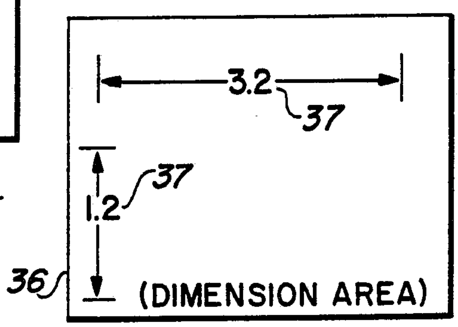


FIG. 10C

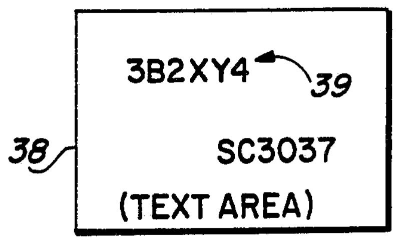


FIG. 10D

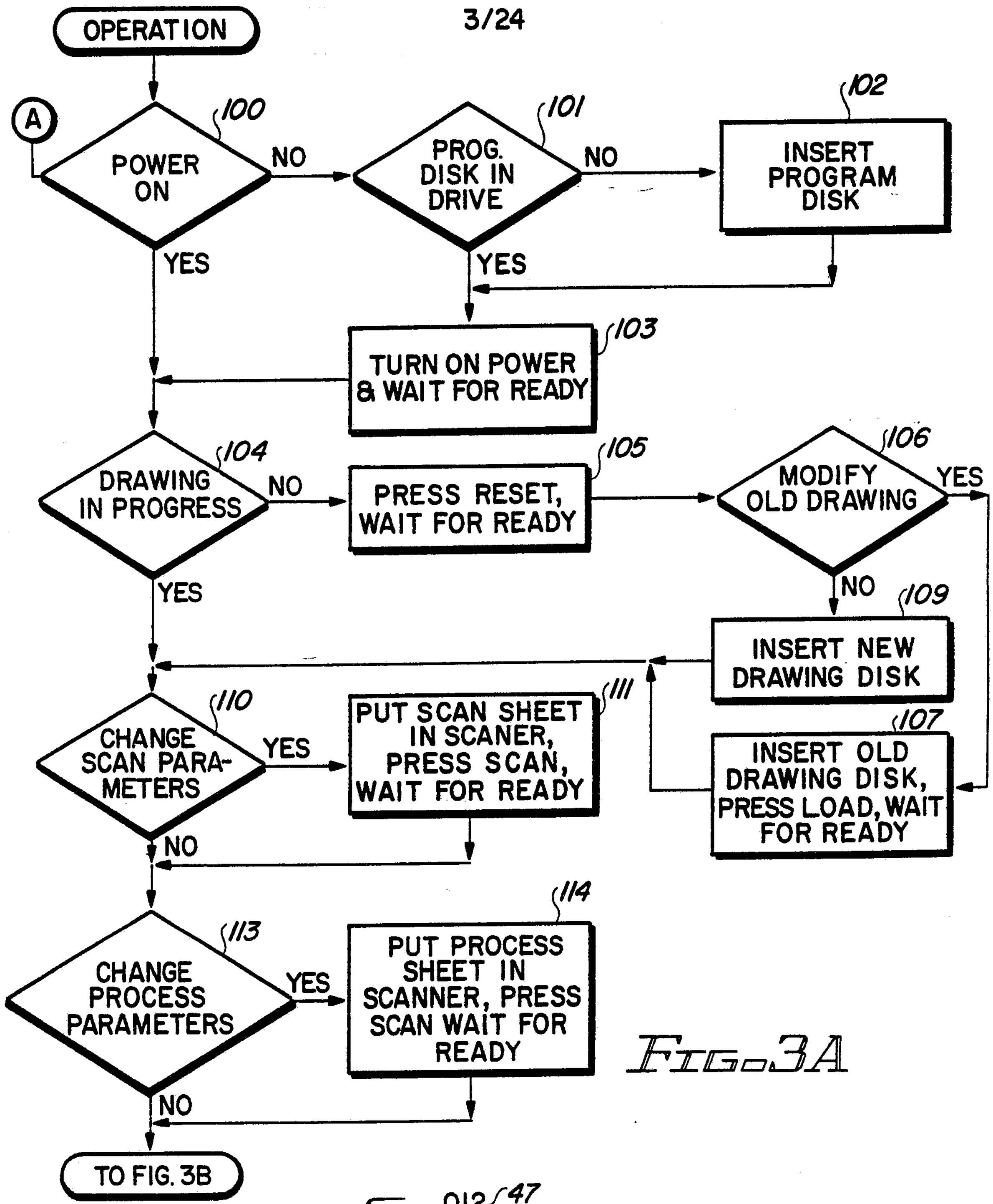

OPERATION
A
100
POWER ON
NO
YES
101
PROG. DISK IN DRIVE
NO
YES
102
INSERT PROGRAM DISK
103
TURN ON POWER & WAIT FOR READY
104
DRAWING IN PROGRESS
NO
YES
105
PRESS RESET, WAIT FOR READY
106
MODIFY OLD DRAWING
YES
NO
109
INSERT NEW DRAWING DISK
107
INSERT OLD DRAWING DISK, PRESS LOAD, WAIT FOR READY
110
CHANGE SCAN PARA-METERS
YES
NO
111
PUT SCAN SHEET IN SCANER, PRESS SCAN, WAIT FOR READY
113
CHANGE PROCESS PARAMETERS
YES
NO
114
PUT PROCESS SHEET IN SCANNER, PRESS SCAN WAIT FOR READY
TO FIG. 3B

FIG. 3A

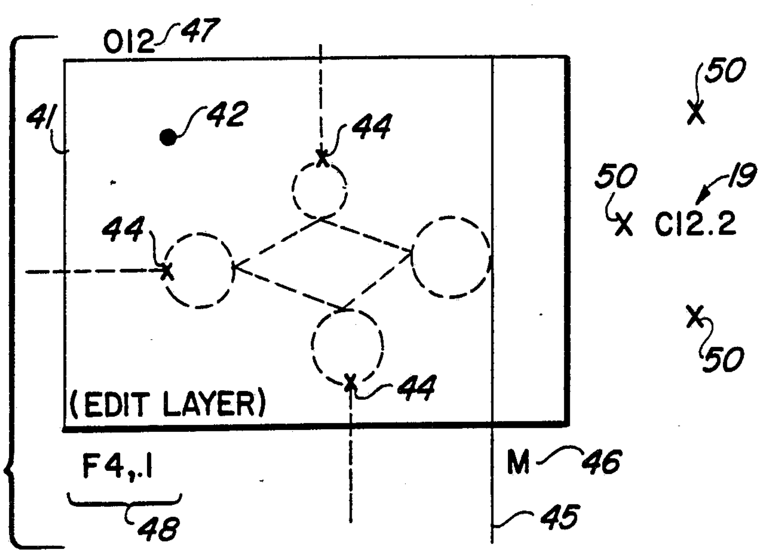

012
47
41
42
44
44
44
(EDIT LAYER)
F4,.1
48
M
46
45
50
50
19
C12.2
50

FIG. 11

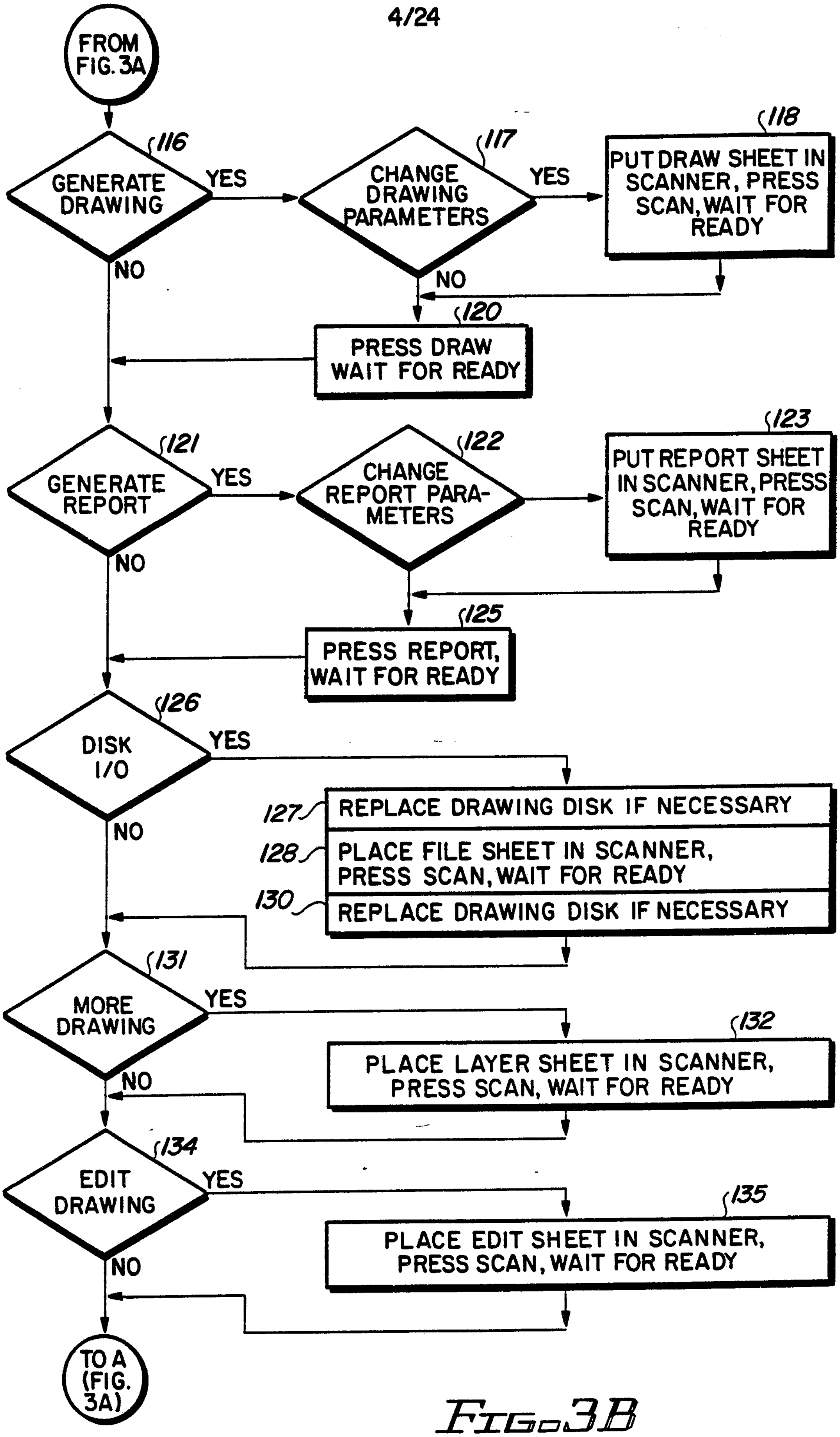

FROM FIG. 3A
GENERATE DRAWING
116
YES
NO
CHANGE DRAWING PARAMETERS
117
YES
NO
PUT DRAW SHEET IN SCANNER, PRESS SCAN, WAIT FOR READY
118
PRESS DRAW WAIT FOR READY
120
GENERATE REPORT
121
YES
NO
CHANGE REPORT PARA-METERS
122
PUT REPORT SHEET IN SCANNER, PRESS SCAN, WAIT FOR READY
123
PRESS REPORT, WAIT FOR READY
125
DISK I/O
126
YES
NO
127
REPLACE DRAWING DISK IF NECESSARY
128
PLACE FILE SHEET IN SCANNER, PRESS SCAN, WAIT FOR READY
130
REPLACE DRAWING DISK IF NECESSARY
MORE DRAWING
131
YES
NO
PLACE LAYER SHEET IN SCANNER, PRESS SCAN, WAIT FOR READY
132
EDIT DRAWING
134
YES
NO
PLACE EDIT SHEET IN SCANNER, PRESS SCAN, WAIT FOR READY
135
TO A (FIG. 3A)

FIG. 3B

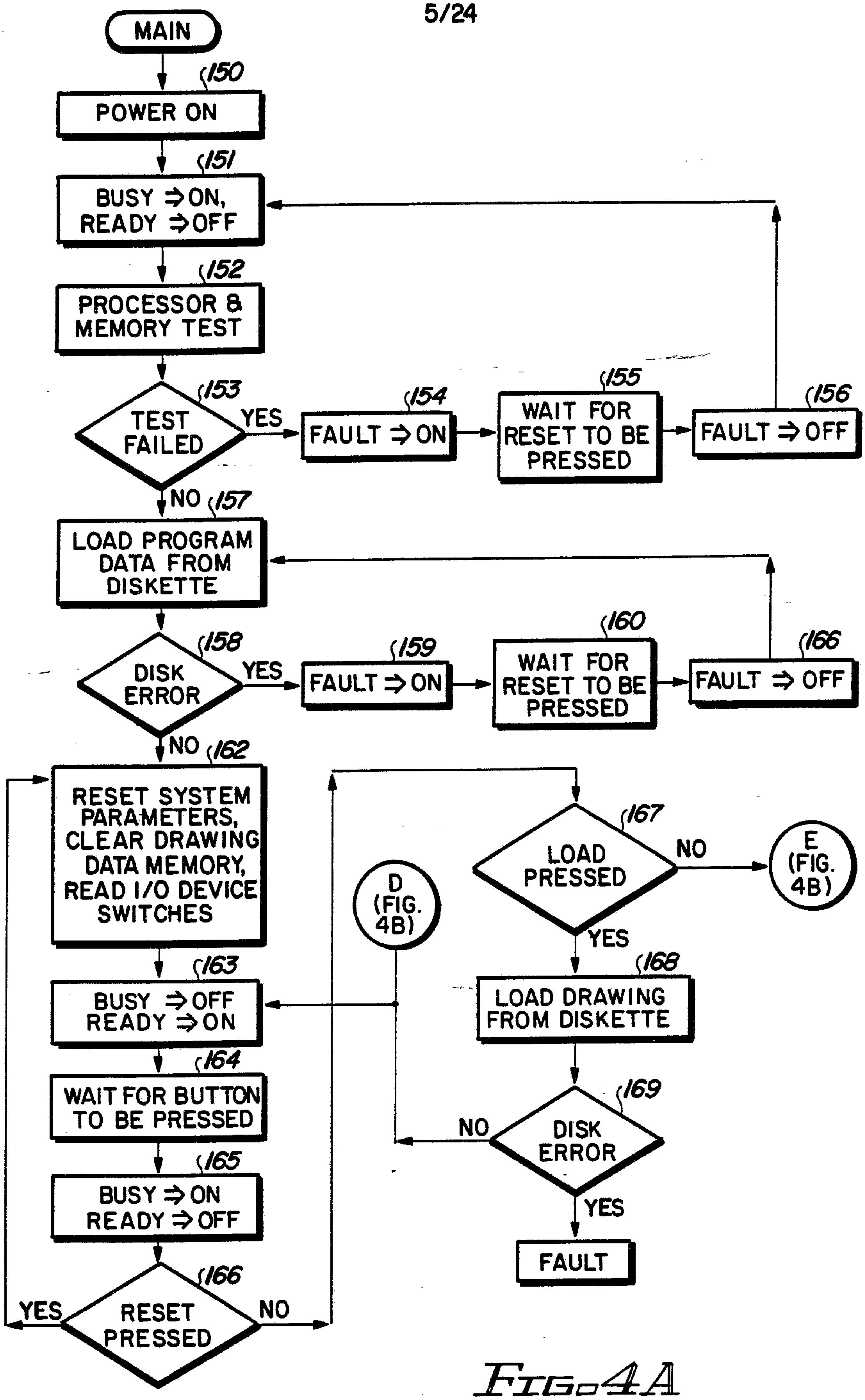

MAIN
150
POWER ON
151
BUSY ⇒ON,
READY ⇒OFF
152
PROCESSOR &
MEMORY TEST
153
TEST
FAILED
YES
154
FAULT ⇒ON
155
WAIT FOR
RESET TO BE
PRESSED
156
FAULT ⇒OFF
NO
157
LOAD PROGRAM
DATA FROM
DISKETTE
158
DISK
ERROR
YES
159
FAULT ⇒ON
160
WAIT FOR
RESET TO BE
PRESSED
166
FAULT ⇒ OFF
NO
162
RESET SYSTEM
PARAMETERS,
CLEAR DRAWING
DATA MEMORY,
READ I/O DEVICE
SWITCHES
167
LOAD
PRESSED
NO
E
(FIG.
4B)
D
(FIG.
4B)
YES
168
LOAD DRAWING
FROM DISKETTE
163
BUSY ⇒OFF
READY ⇒ON
164
WAIT FOR BUTTON
TO BE PRESSED
169
NO
DISK
ERROR
YES
165
BUSY ⇒ON
READY ⇒OFF
FAULT
166
YES
RESET
PRESSED
NO

FIG. 4A

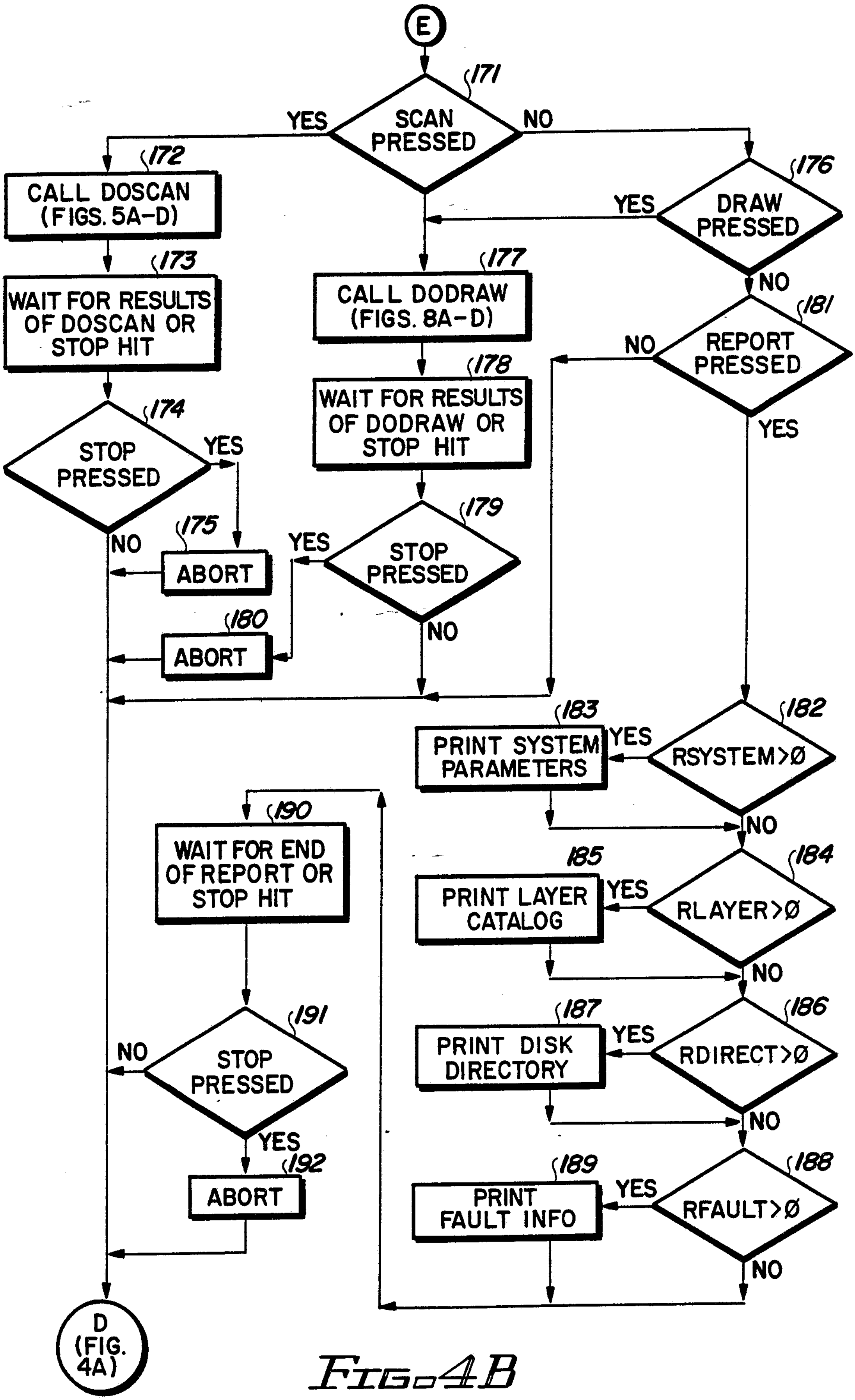
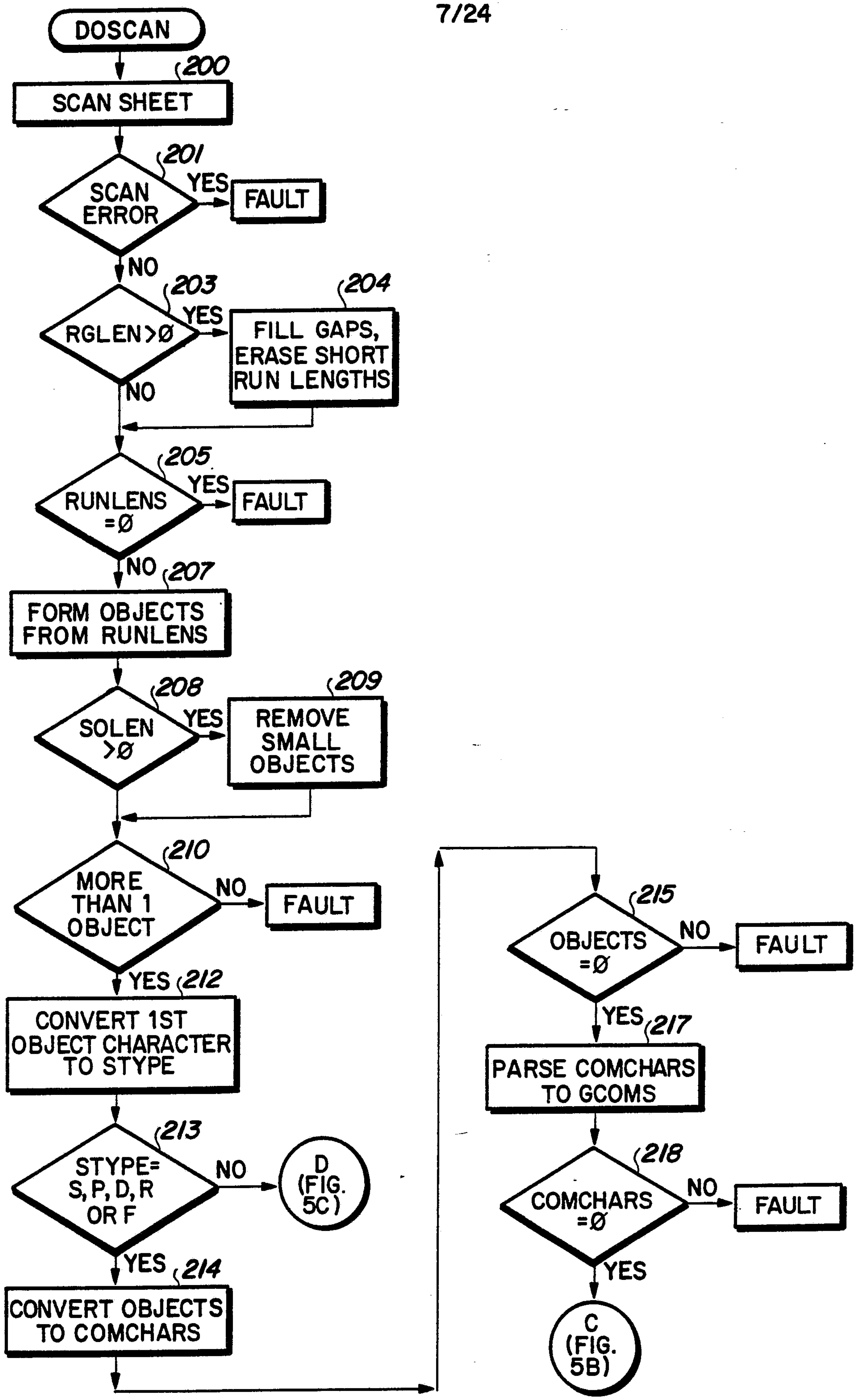

E
171
SCAN PRESSED
YES
NO
172
CALL DOSCAN (FIGS. 5A-D)
176
DRAW PRESSED
YES
NO
173
WAIT FOR RESULTS OF DOSCAN OR STOP HIT
177
CALL DODRAW (FIGS. 8A-D)
181
REPORT PRESSED
NO
YES
178
WAIT FOR RESULTS OF DODRAW OR STOP HIT
174
STOP PRESSED
YES
NO
175
ABORT
179
STOP PRESSED
YES
NO
180
ABORT
183
PRINT SYSTEM PARAMETERS
YES
182
RSYSTEM>Ø
NO
190
WAIT FOR END OF REPORT OR STOP HIT
185
PRINT LAYER CATALOG
YES
184
RLAYER>Ø
NO
191
STOP PRESSED
NO
YES
187
PRINT DISK DIRECTORY
YES
186
RDIRECT>Ø
NO
192
ABORT
189
PRINT FAULT INFO
YES
188
RFAULT>Ø
NO
D (FIG. 4A)

FIG. 4B



DOSCAN
200
SCAN SHEET
201
SCAN ERROR
YES
FAULT
NO
203
RGLEN>Ø
YES
204
FILL GAPS, ERASE SHORT RUN LENGTHS
NO
205
RUNLENS =Ø
YES
FAULT
NO
207
FORM OBJECTS FROM RUNLENS
208
SOLEN >Ø
YES
209
REMOVE SMALL OBJECTS
210
MORE THAN 1 OBJECT
NO
FAULT
YES
212
CONVERT 1ST OBJECT CHARACTER TO STYPE
213
STYPE= S, P, D, R OR F
NO
D (FIG. 5C)
YES
214
CONVERT OBJECTS TO COMCHARS
215
OBJECTS =Ø
NO
FAULT
YES
217
PARSE COMCHARS TO GCOMS
218
COMCHARS =Ø
NO
FAULT
YES
C (FIG. 5B)

FIG. 5A

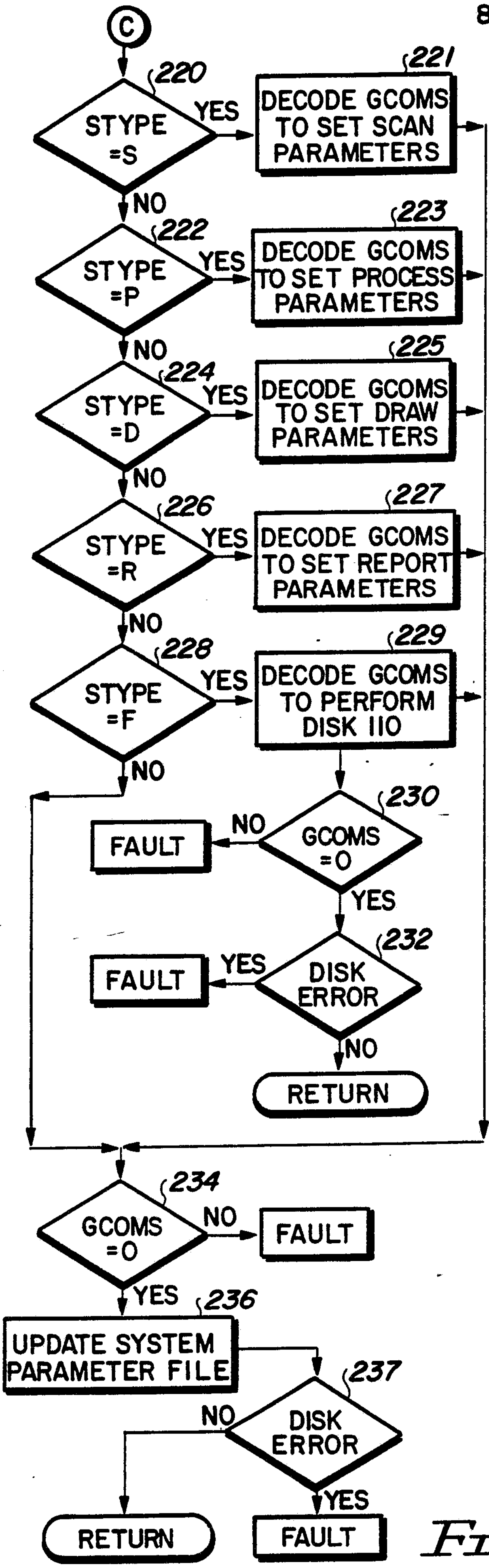

C
220
STYPE =S
YES
221
DECODE GCOMS TO SET SCAN PARAMETERS
NO
222
STYPE =P
YES
223
DECODE GCOMS TO SET PROCESS PARAMETERS
NO
224
STYPE =D
YES
225
DECODE GCOMS TO SET DRAW PARAMETERS
NO
226
STYPE =R
YES
227
DECODE GCOMS TO SET REPORT PARAMETERS
NO
228
STYPE =F
YES
229
DECODE GCOMS TO PERFORM DISK I/O
NO
230
GCOMS =0
NO
FAULT
YES
232
DISK ERROR
YES
FAULT
NO
RETURN
234
GCOMS =0
NO
FAULT
YES
236
UPDATE SYSTEM PARAMETER FILE
237
DISK ERROR
NO
RETURN
YES
FAULT

FIG. 5B

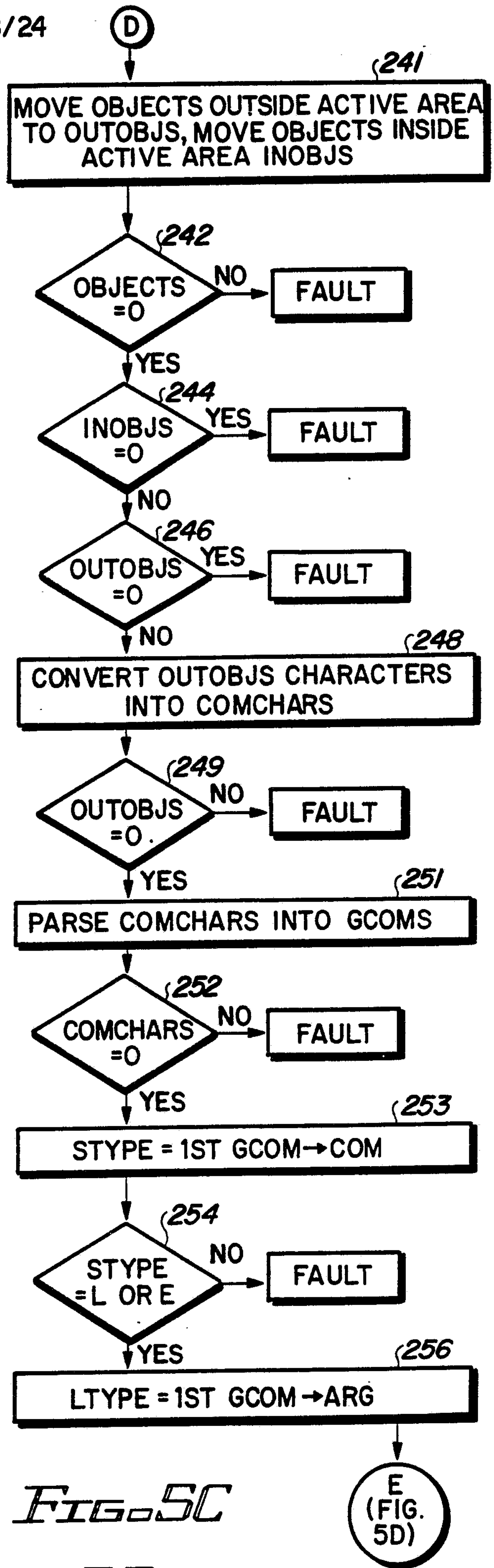

D
241
MOVE OBJECTS OUTSIDE ACTIVE AREA TO OUTOBJS, MOVE OBJECTS INSIDE ACTIVE AREA INOBJS
242
OBJECTS =0
NO
FAULT
YES
244
INOBJS =0
YES
FAULT
NO
246
OUTOBJS =0
YES
FAULT
NO
248
CONVERT OUTOBJS CHARACTERS INTO COMCHARS
249
OUTOBJS =0
NO
FAULT
YES
251
PARSE COMCHARS INTO GCOMS
252
COMCHARS =0
NO
FAULT
YES
253
STYPE = 1ST GCOM→COM
254
STYPE =L OR E
NO
FAULT
YES
256
LTYPE = 1ST GCOM→ARG
E (FIG. 5D)

FIG. 5C

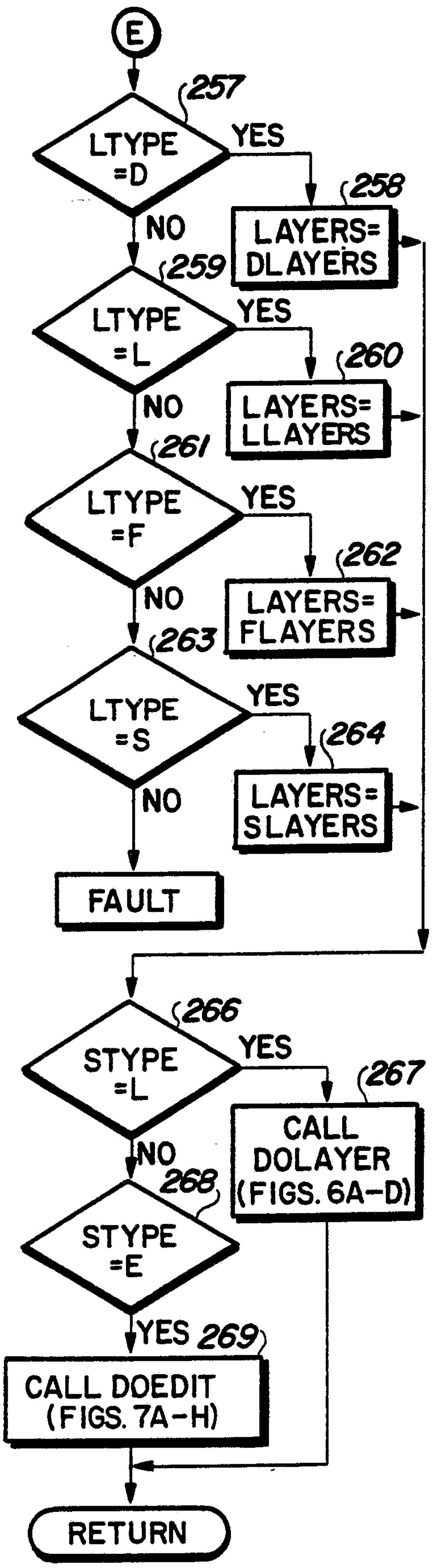

E
257
LTYPE =D
YES
NO
258
LAYERS= DLAYERS
259
LTYPE =L
YES
NO
260
LAYERS= LLAYERS
261
LTYPE =F
YES
NO
262
LAYERS= FLAYERS
263
LTYPE =S
YES
NO
264
LAYERS= SLAYERS
FAULT
266
STYPE =L
YES
NO
267
CALL DOLAYER (FIGS. 6A-D)
268
STYPE =E
YES 269
CALL DOEDIT (FIGS. 7A-H)
RETURN

FIG. 5D

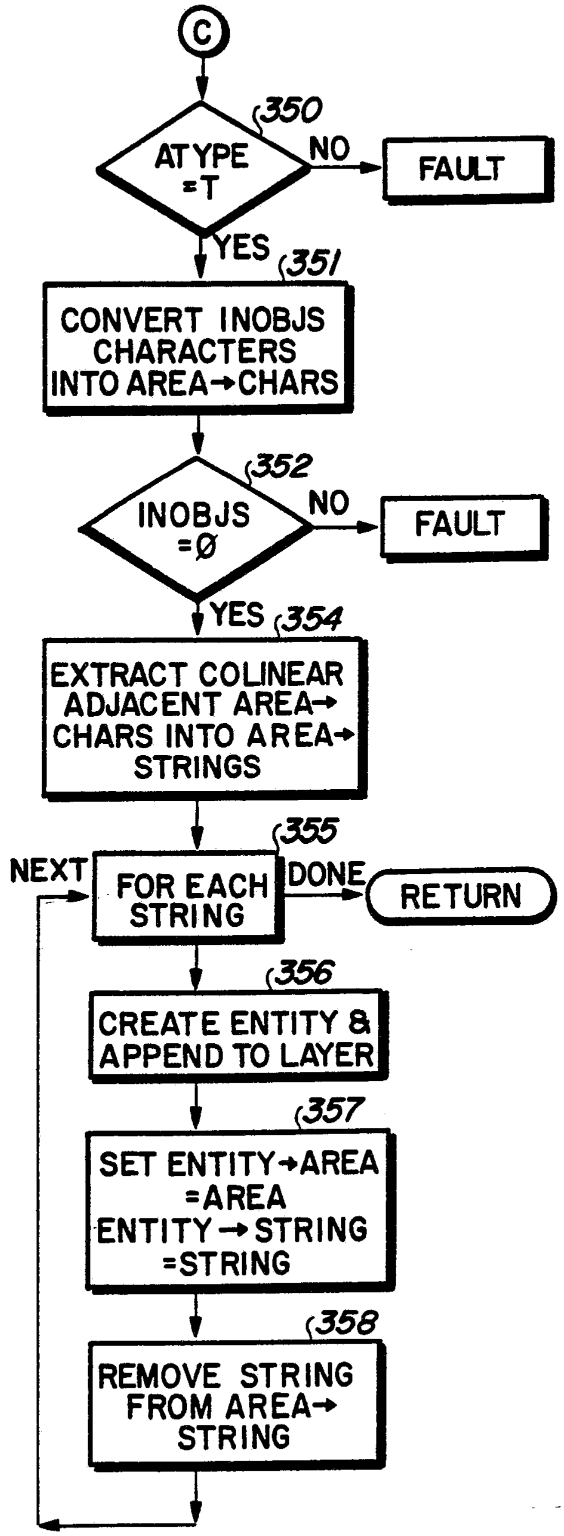

C
350
ATYPE =T
NO
FAULT
YES
351
CONVERT INOBJS CHARACTERS INTO AREA→CHARS
352
INOBJS =Ø
NO
FAULT
YES
354
EXTRACT COLINEAR ADJACENT AREA→ CHARS INTO AREA→ STRINGS
355
NEXT
FOR EACH STRING
DONE
RETURN
356
CREATE ENTITY & APPEND TO LAYER
357
SET ENTITY→AREA =AREA ENTITY→STRING =STRING
358
REMOVE STRING FROM AREA→ STRING

FIG. 6D

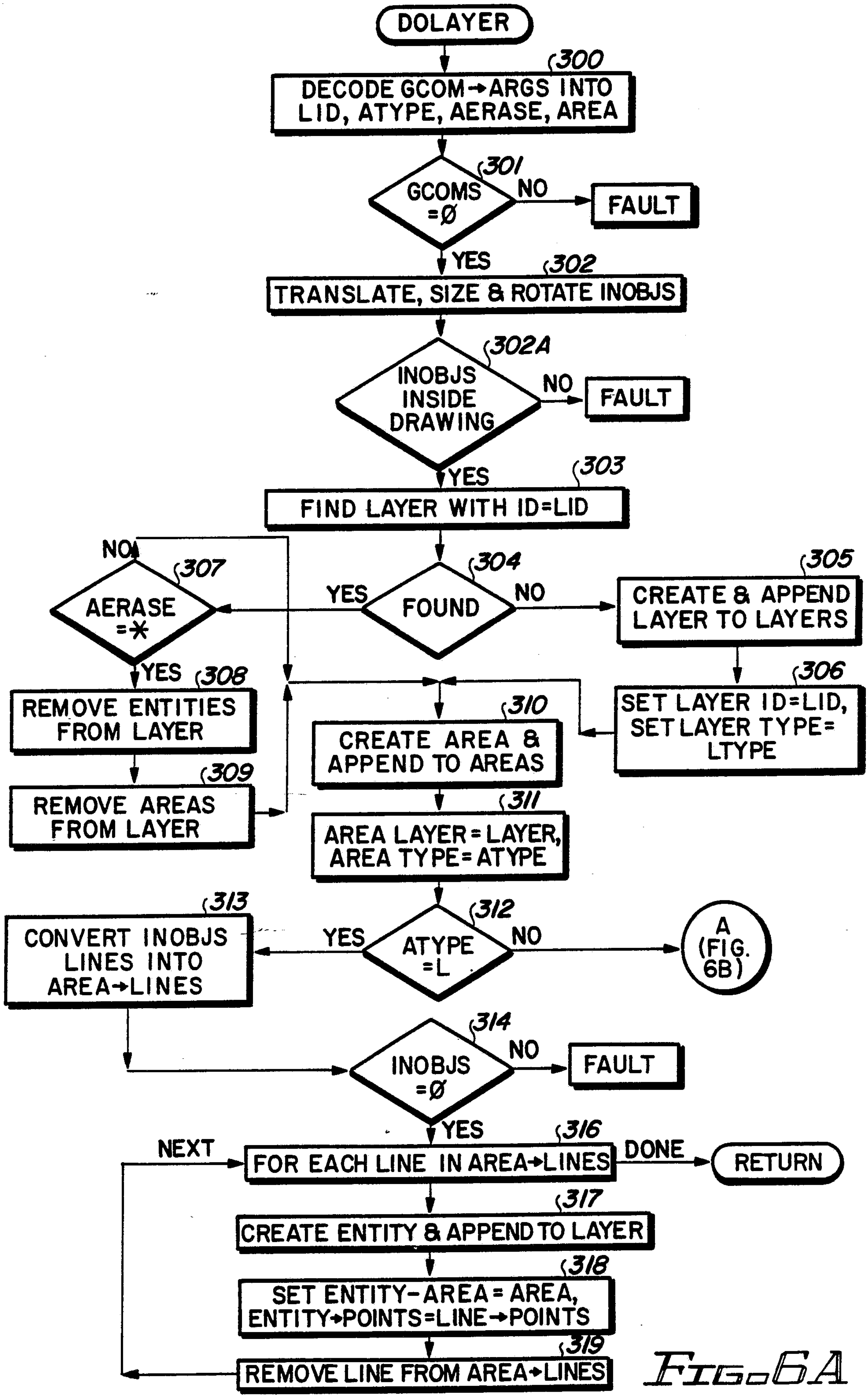

DOLAYER
300
DECODE GCOM→ARGS INTO LID, ATYPE, AERASE, AREA
301
GCOMS =Ø
NO
FAULT
YES
302
TRANSLATE, SIZE & ROTATE INOBJS
302A
INOBJS INSIDE DRAWING
NO
FAULT
YES
303
FIND LAYER WITH ID=LID
304
FOUND
YES
NO
305
CREATE & APPEND LAYER TO LAYERS
306
SET LAYER ID=LID, SET LAYER TYPE= LTYPE
307
AERASE =*
NO
YES
308
REMOVE ENTITIES FROM LAYER
309
REMOVE AREAS FROM LAYER
310
CREATE AREA & APPEND TO AREAS
311
AREA LAYER=LAYER, AREA TYPE=ATYPE
312
ATYPE =L
YES
NO
A (FIG. 6B)
313
CONVERT INOBJS LINES INTO AREA→LINES
314
INOBJS =Ø
NO
FAULT
YES
316
FOR EACH LINE IN AREA→LINES
NEXT
DONE
RETURN
317
CREATE ENTITY & APPEND TO LAYER
318
SET ENTITY-AREA=AREA, ENTITY→POINTS=LINE→POINTS
319
REMOVE LINE FROM AREA→LINES

FIG. 6A

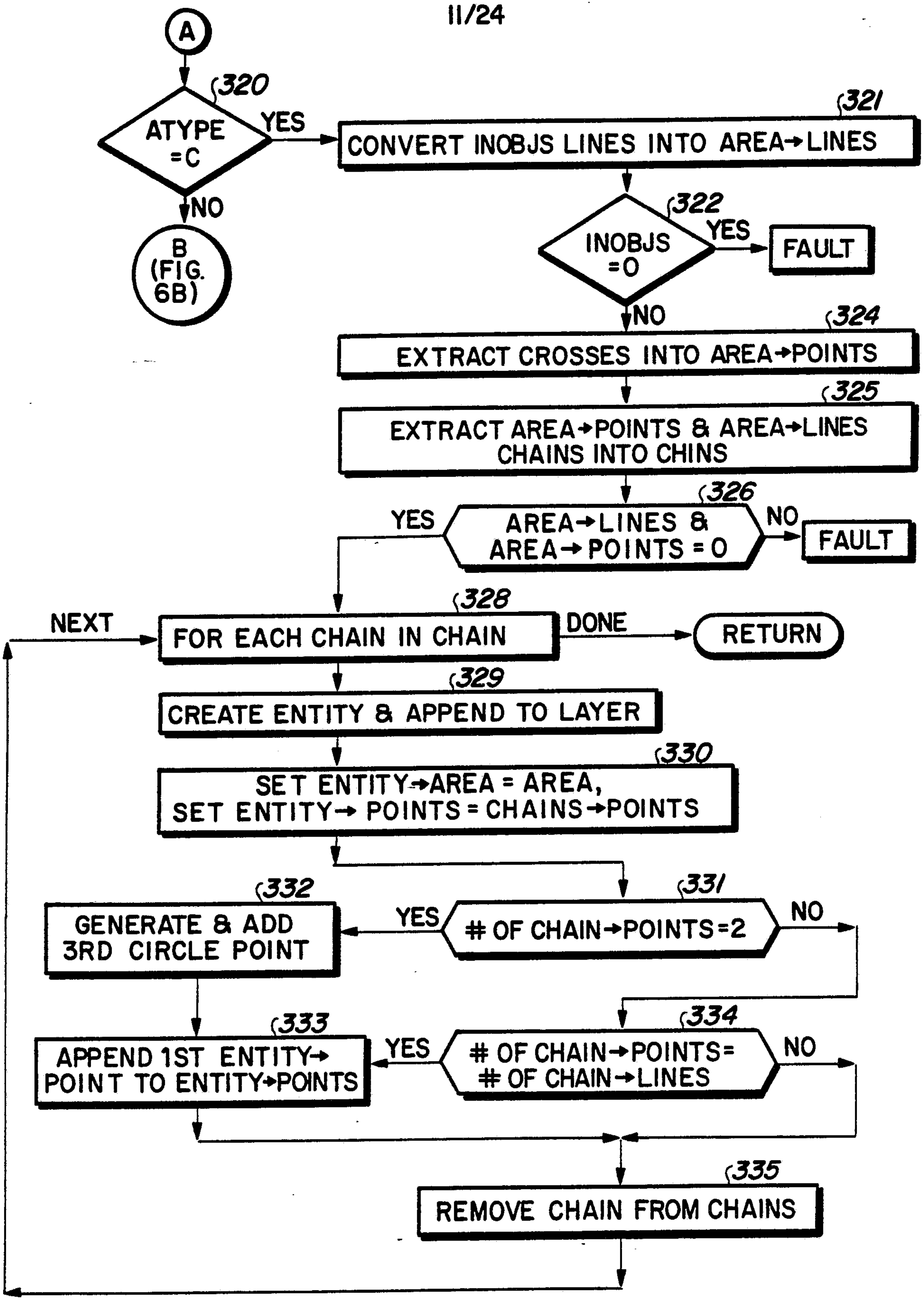

A
ATYPE =C
320
YES
NO
B (FIG. 6B)
CONVERT INOBJS LINES INTO AREA→LINES
321
INOBJS =0
322
YES
FAULT
NO
EXTRACT CROSSES INTO AREA→POINTS
324
EXTRACT AREA→POINTS & AREA→LINES CHAINS INTO CHINS
325
AREA→LINES & AREA→POINTS = 0
326
YES
NO
FAULT
NEXT
FOR EACH CHAIN IN CHAIN
328
DONE
RETURN
CREATE ENTITY & APPEND TO LAYER
329
SET ENTITY→AREA = AREA, SET ENTITY→ POINTS = CHAINS→POINTS
330
GENERATE & ADD 3RD CIRCLE POINT
332
OF CHAIN→POINTS=2
331
YES
NO
APPEND 1ST ENTITY→ POINT TO ENTITY→POINTS
333
OF CHAIN→POINTS= # OF CHAIN→LINES
334
YES
NO
REMOVE CHAIN FROM CHAINS
335

FIG. 6B

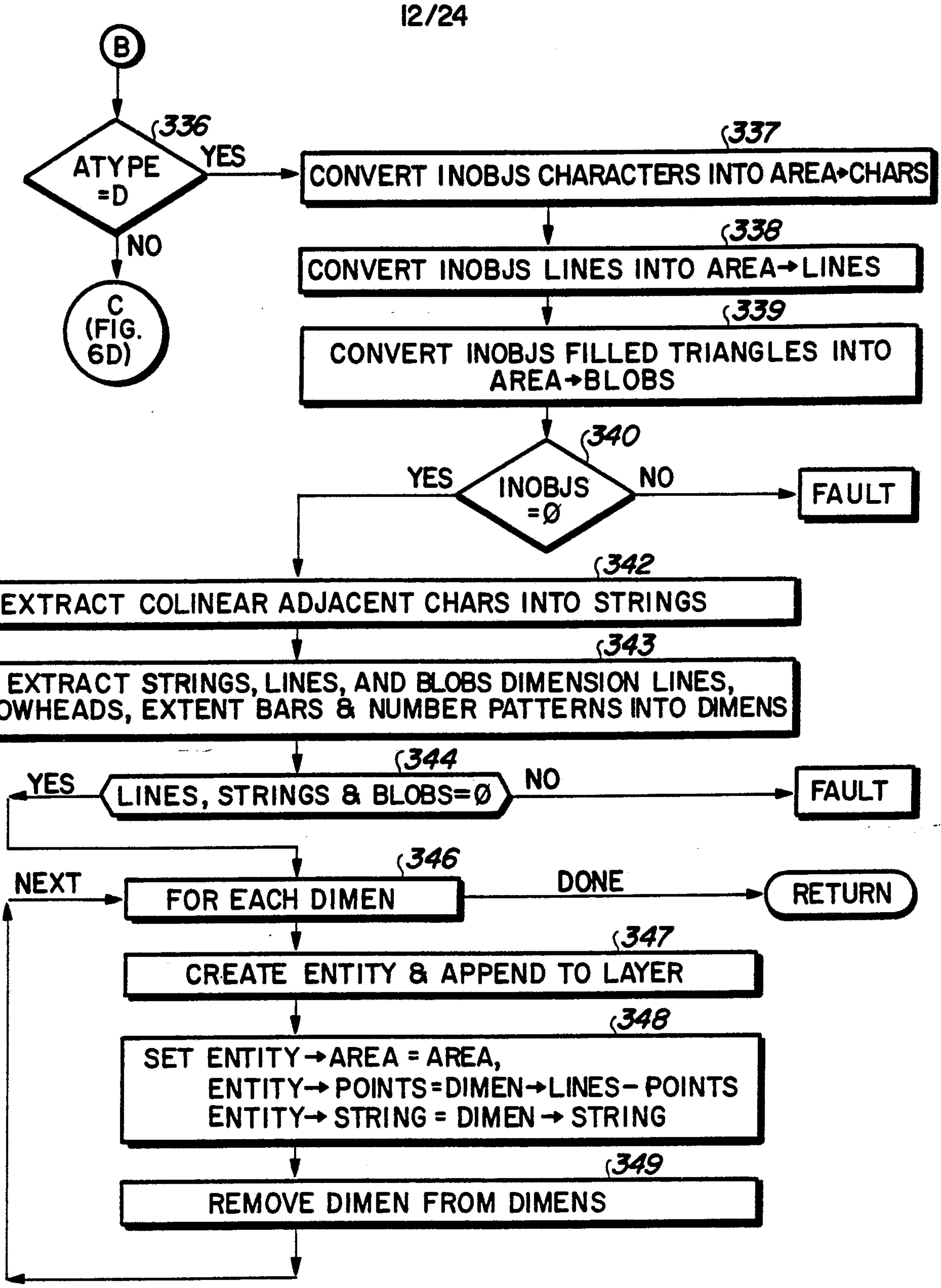
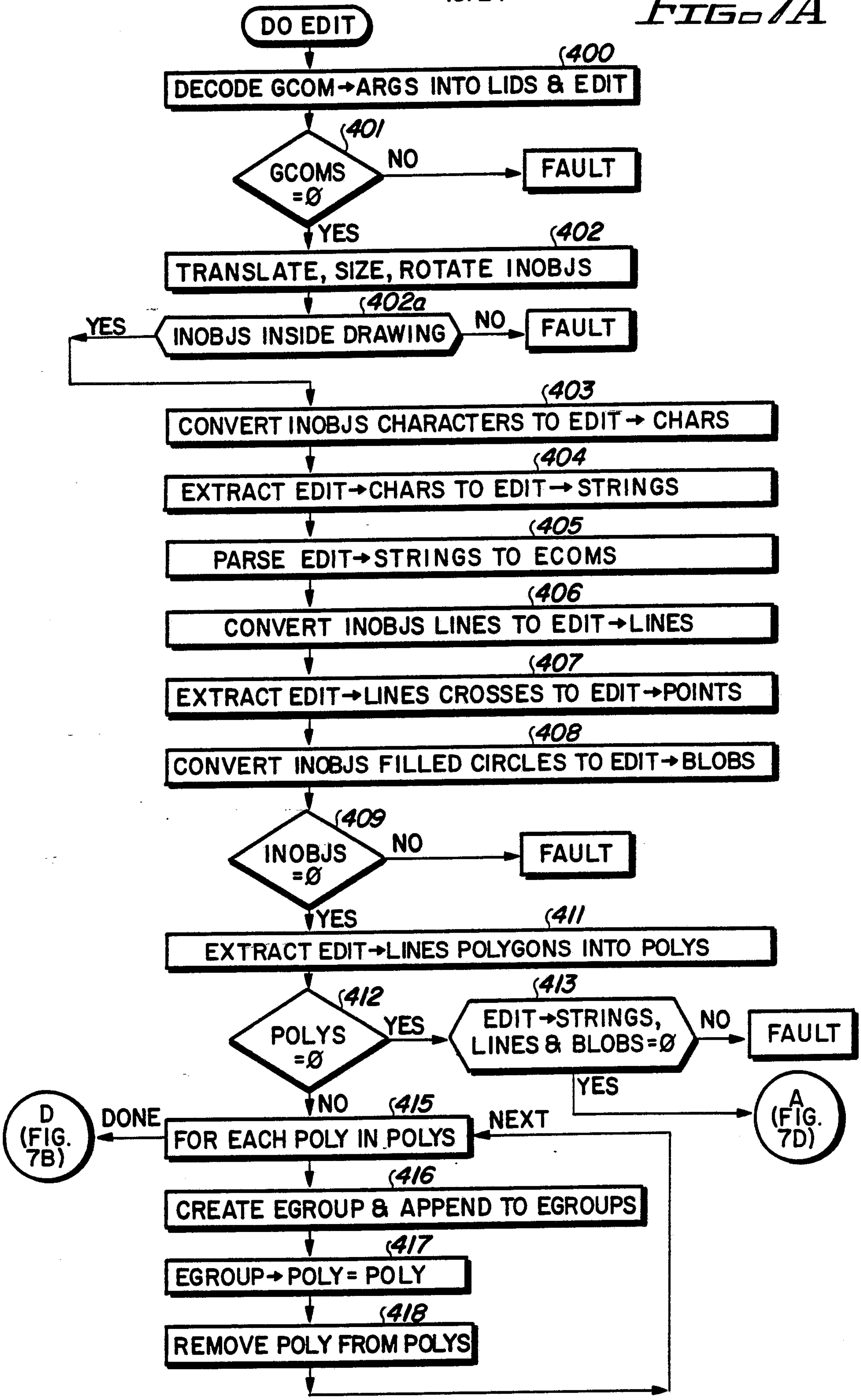

B
336
ATYPE =D
YES
NO
C (FIG. 6D)
337
CONVERT INOBJS CHARACTERS INTO AREA→CHARS
338
CONVERT INOBJS LINES INTO AREA→LINES
339
CONVERT INOBJS FILLED TRIANGLES INTO AREA→BLOBS
340
INOBJS =Ø
YES
NO
FAULT
342
EXTRACT COLINEAR ADJACENT CHARS INTO STRINGS
343
EXTRACT STRINGS, LINES, AND BLOBS DIMENSION LINES, ARROWHEADS, EXTENT BARS & NUMBER PATTERNS INTO DIMENS
344
LINES, STRINGS & BLOBS=Ø
YES
NO
FAULT
346
NEXT
FOR EACH DIMEN
DONE
RETURN
347
CREATE ENTITY & APPEND TO LAYER
348
SET ENTITY→AREA = AREA,
ENTITY→POINTS=DIMEN→LINES–POINTS
ENTITY→STRING = DIMEN → STRING
349
REMOVE DIMEN FROM DIMENS

FIG. 6C

FIG. 7A


DO EDIT
400
DECODE GCOM→ARGS INTO LIDS & EDIT
401
GCOMS =Ø
NO
FAULT
YES
402
TRANSLATE, SIZE, ROTATE INOBJS
402a
INOBJS INSIDE DRAWING
YES
NO
FAULT
403
CONVERT INOBJS CHARACTERS TO EDIT→CHARS
404
EXTRACT EDIT→CHARS TO EDIT→STRINGS
405
PARSE EDIT→STRINGS TO ECOMS
406
CONVERT INOBJS LINES TO EDIT→LINES
407
EXTRACT EDIT→LINES CROSSES TO EDIT→POINTS
408
CONVERT INOBJS FILLED CIRCLES TO EDIT→BLOBS
409
INOBJS =Ø
NO
FAULT
YES
411
EXTRACT EDIT→LINES POLYGONS INTO POLYS
412
POLYS =Ø
YES
413
EDIT→STRINGS, LINES & BLOBS=Ø
NO
FAULT
YES
A (FIG. 7D)
NO
415
D (FIG. 7B)
DONE
FOR EACH POLY IN POLYS
NEXT
416
CREATE EGROUP & APPEND TO EGROUPS
417
EGROUP→POLY= POLY
418
REMOVE POLY FROM POLYS

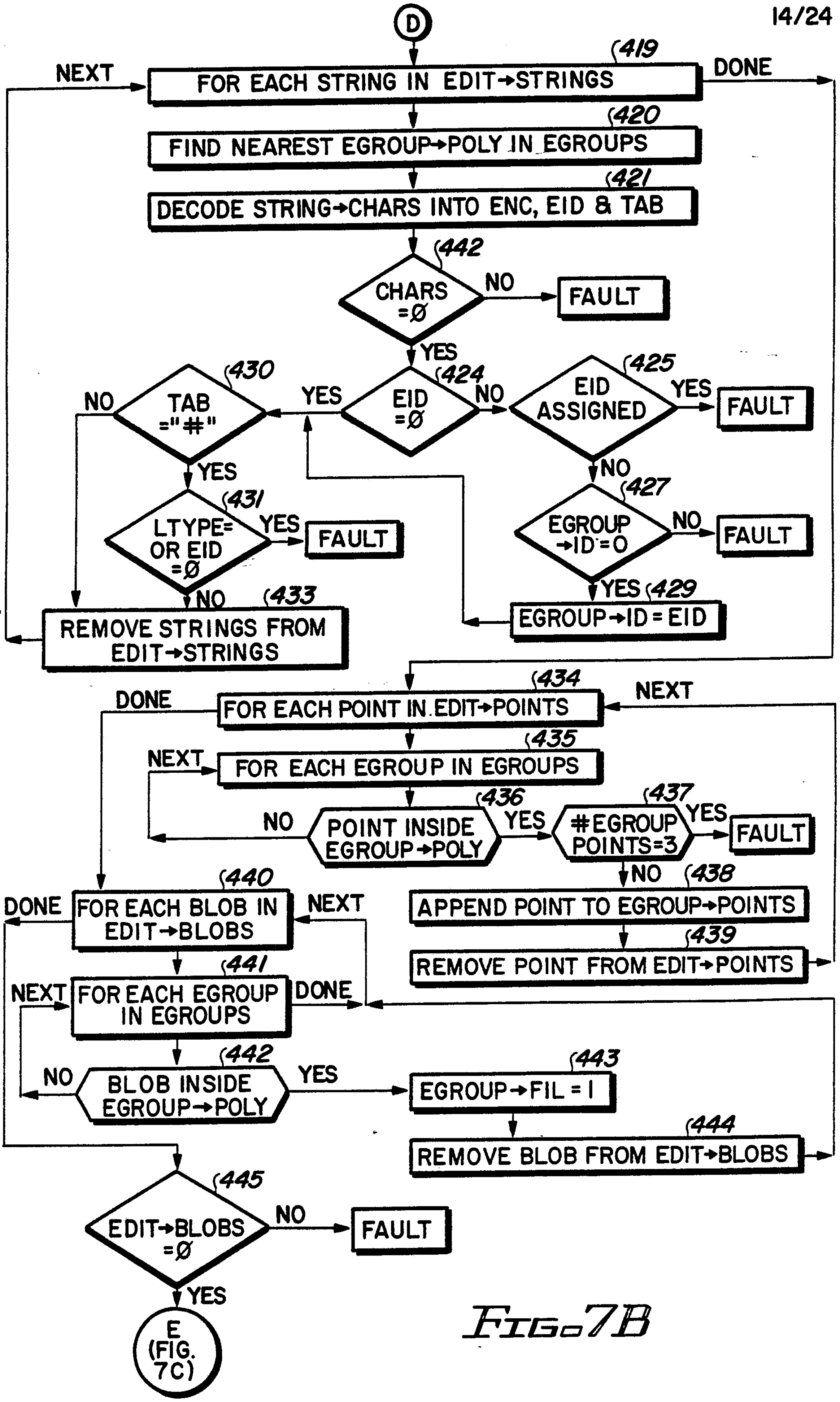

D
419
FOR EACH STRING IN EDIT→STRINGS
NEXT
DONE
420
FIND NEAREST EGROUP→POLY IN EGROUPS
421
DECODE STRING→CHARS INTO ENC, EID & TAB
442
CHARS =Ø
NO
FAULT
YES
424
EID =Ø
YES
NO
430
TAB ="#"
NO
YES
425
EID ASSIGNED
YES
FAULT
NO
431
LTYPE= OR EID =Ø
YES
FAULT
NO
433
REMOVE STRINGS FROM EDIT→STRINGS
427
EGROUP →ID=0
NO
FAULT
YES
429
EGROUP→ID=EID
434
FOR EACH POINT IN EDIT→POINTS
DONE
NEXT
435
FOR EACH EGROUP IN EGROUPS
NEXT
436
POINT INSIDE EGROUP→POLY
NO
YES
437
#EGROUP POINTS=3
YES
FAULT
NO
438
APPEND POINT TO EGROUP→POINTS
439
REMOVE POINT FROM EDIT→POINTS
440
FOR EACH BLOB IN EDIT→BLOBS
DONE
NEXT
441
FOR EACH EGROUP IN EGROUPS
NEXT
DONE
442
BLOB INSIDE EGROUP→POLY
NO
YES
443
EGROUP→FIL=1
444
REMOVE BLOB FROM EDIT→BLOBS
445
EDIT→BLOBS =Ø
NO
FAULT
YES
E (FIG. 7C)

FIG. 7B

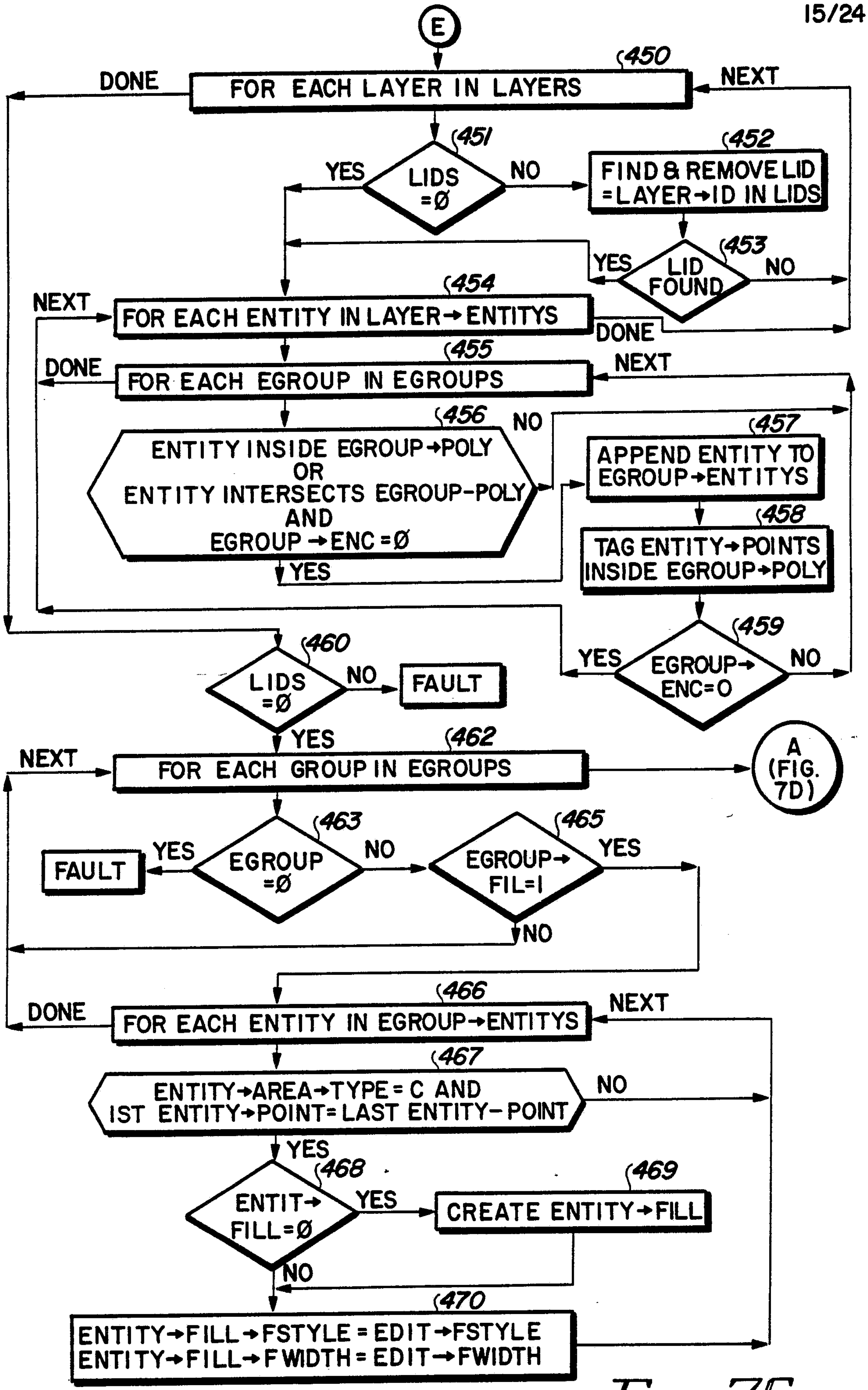

E
FOR EACH LAYER IN LAYERS
450
DONE
NEXT
LIDS =Ø
451
YES
NO
FIND & REMOVE LID =LAYER→ID IN LIDS
452
LID FOUND
453
YES
NO
FOR EACH ENTITY IN LAYER→ENTITYS
454
NEXT
DONE
FOR EACH EGROUP IN EGROUPS
455
DONE
NEXT
ENTITY INSIDE EGROUP→POLY
OR
ENTITY INTERSECTS EGROUP-POLY
AND
EGROUP →ENC =Ø
456
NO
YES
APPEND ENTITY TO EGROUP→ENTITYS
457
TAG ENTITY→POINTS INSIDE EGROUP→POLY
458
EGROUP→ ENC=0
459
YES
NO
LIDS =Ø
460
NO
FAULT
YES
FOR EACH GROUP IN EGROUPS
462
NEXT
A (FIG. 7D)
EGROUP =Ø
463
YES
FAULT
NO
EGROUP→ FIL=1
465
YES
NO
FOR EACH ENTITY IN EGROUP→ENTITYS
466
DONE
NEXT
ENTITY→AREA→TYPE = C AND
1ST ENTITY→POINT= LAST ENTITY- POINT
467
NO
YES
ENTIT→ FILL=Ø
468
YES
CREATE ENTITY→FILL
469
NO
ENTITY→FILL→FSTYLE = EDIT→FSTYLE
ENTITY→FILL→FWIDTH = EDIT→FWIDTH
470

FIG. 7C

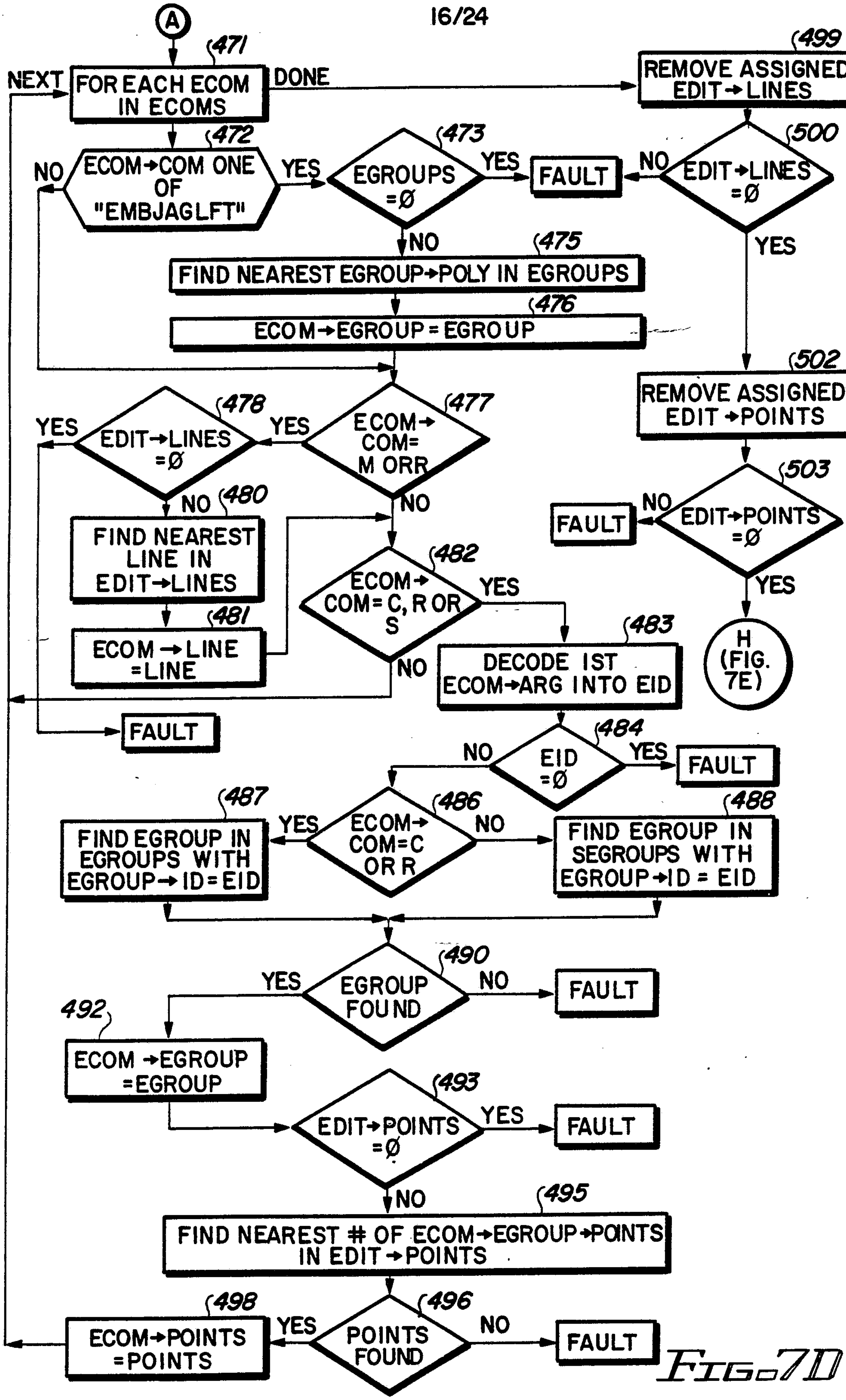

A
471
NEXT
FOR EACH ECOM IN ECOMS
DONE
499
REMOVE ASSIGNED EDIT→LINES
472
NO
ECOM→COM ONE OF "EMBJAGLFT"
YES
473
EGROUPS =Ø
YES
FAULT
NO
500
EDIT→LINES =Ø
NO
YES
475
FIND NEAREST EGROUP→POLY IN EGROUPS
476
ECOM→EGROUP = EGROUP
502
REMOVE ASSIGNED EDIT→POINTS
478
YES
EDIT→LINES =Ø
YES
477
ECOM→COM= M ORR
NO
480
NO
FIND NEAREST LINE IN EDIT→LINES
503
NO
FAULT
EDIT→POINTS =Ø
YES
482
ECOM→COM= C, R OR S
YES
481
ECOM→LINE =LINE
NO
483
DECODE IST ECOM→ARG INTO EID
H (FIG. 7E)
FAULT
484
NO
EID =Ø
YES
FAULT
487
486
488
FIND EGROUP IN EGROUPS WITH EGROUP→ID=EID
YES
ECOM→COM=C OR R
NO
FIND EGROUP IN SEGROUPS WITH EGROUP→ID = EID
490
YES
EGROUP FOUND
NO
FAULT
492
ECOM →EGROUP =EGROUP
493
EDIT→POINTS =Ø
YES
FAULT
NO
495
FIND NEAREST # OF ECOM→EGROUP→POINTS IN EDIT→POINTS
498
496
ECOM→POINTS =POINTS
YES
POINTS FOUND
NO
FAULT

FIG. 7D

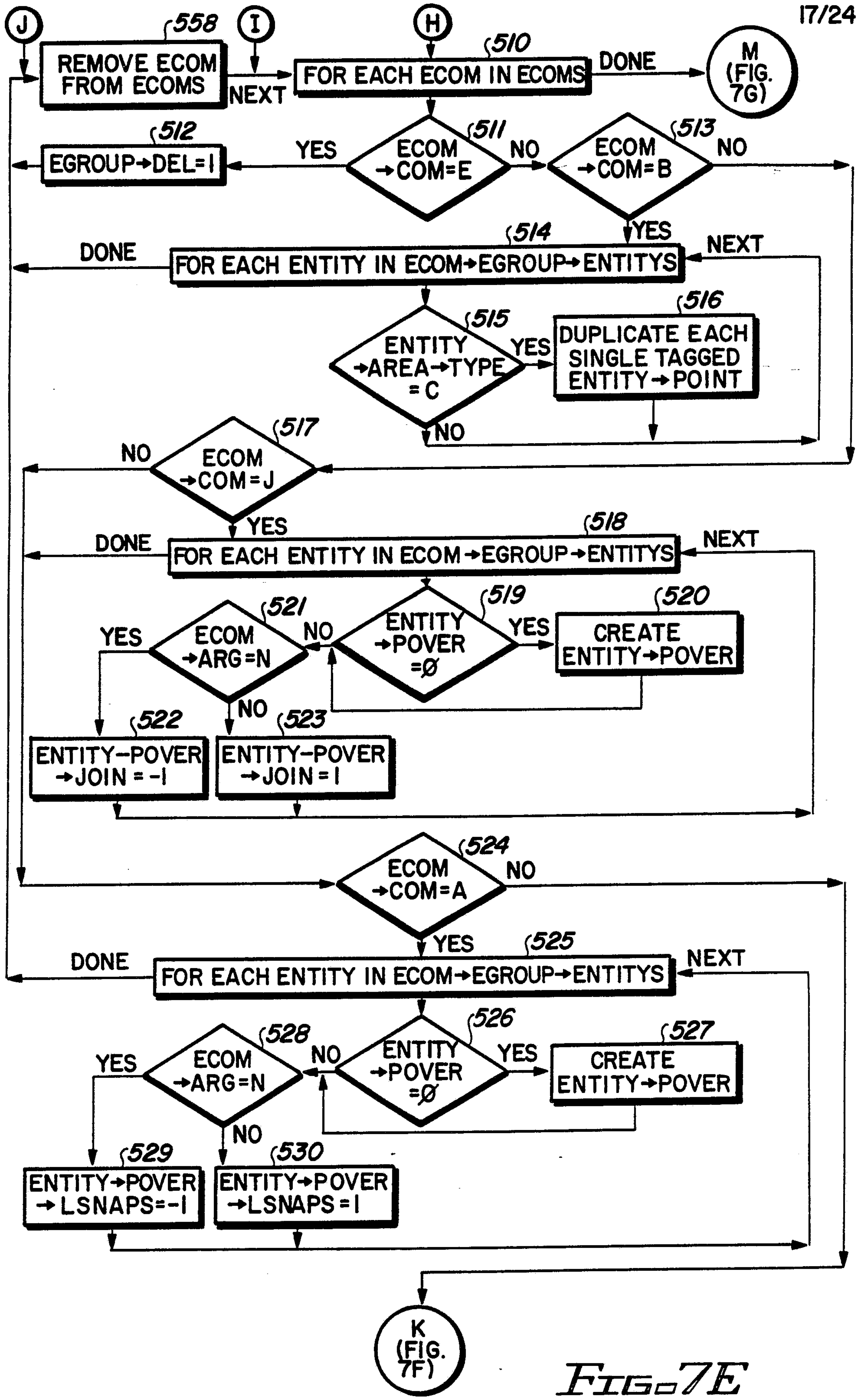

J
I
H
558
REMOVE ECOM FROM ECOMS
NEXT
510
FOR EACH ECOM IN ECOMS
DONE
M (FIG. 7G)
512
EGROUP→DEL=1
YES
511
ECOM →COM=E
NO
513
ECOM →COM=B
NO
YES
514
DONE
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
515
ENTITY →AREA→TYPE =C
YES
516
DUPLICATE EACH SINGLE TAGGED ENTITY→POINT
NO
517
NO
ECOM →COM=J
YES
518
DONE
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
521
YES
ECOM →ARG=N
NO
519
ENTITY →POVER =Ø
YES
520
CREATE ENTITY→POVER
NO
522
ENTITY→POVER →JOIN=-1
523
ENTITY→POVER →JOIN=1
524
ECOM →COM=A
NO
YES
525
DONE
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
528
YES
ECOM →ARG=N
NO
526
ENTITY →POVER =Ø
YES
527
CREATE ENTITY→POVER
NO
529
ENTITY→POVER →LSNAPS=-1
530
ENTITY→POVER →LSNAPS=1
K (FIG. 7F)

FIG-7E

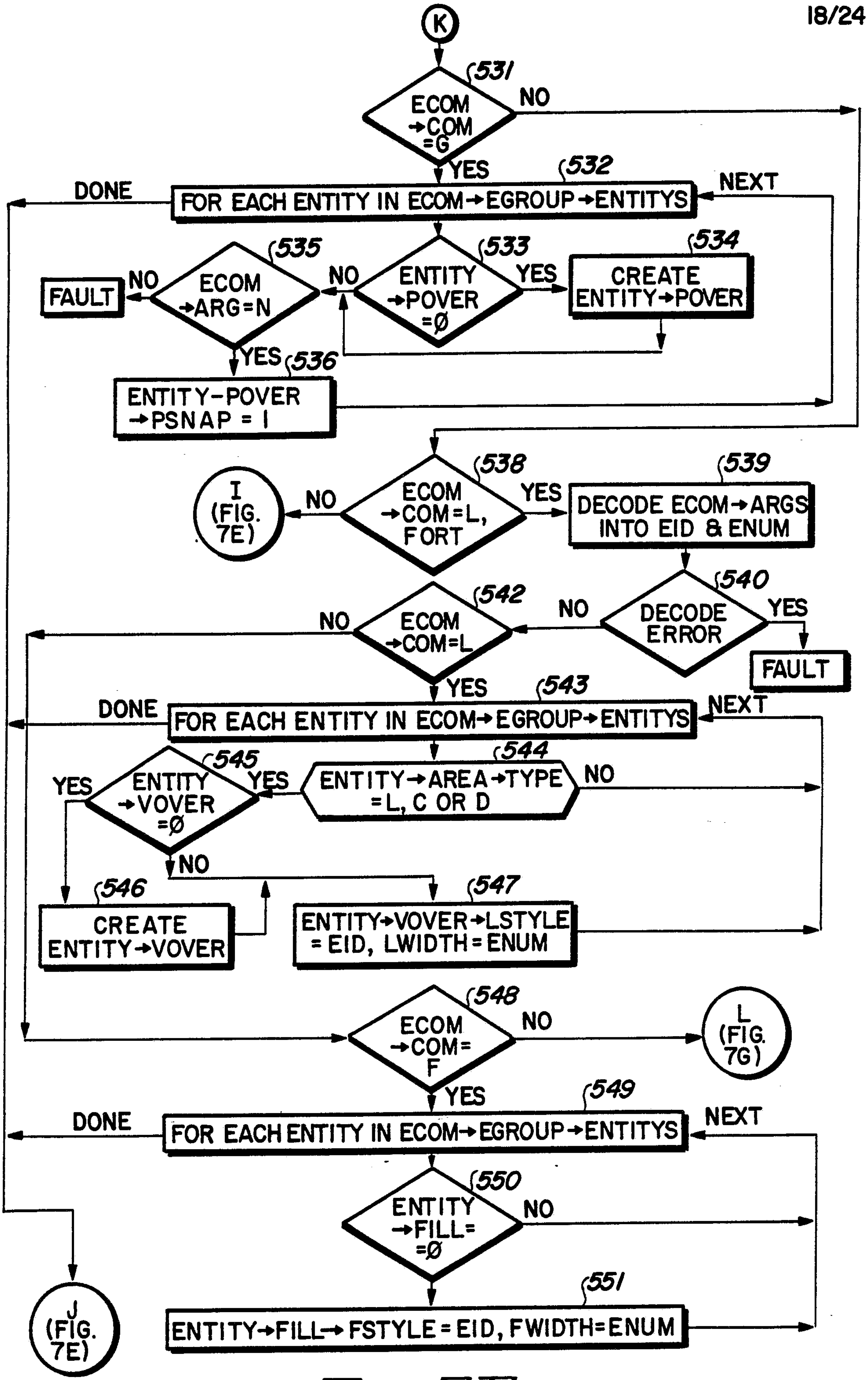

K
531
ECOM →COM =G
NO
YES
532
DONE
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
535
FAULT
NO
ECOM →ARG=N
NO
533
ENTITY →POVER =Ø
YES
534
CREATE ENTITY→POVER
YES
536
ENTITY-POVER →PSNAP = I
538
I (FIG. 7E)
NO
ECOM →COM=L, FORT
YES
539
DECODE ECOM→ARGS INTO EID & ENUM
540
DECODE ERROR
YES
FAULT
NO
542
NO
ECOM →COM=L
YES
543
DONE
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
545
YES
ENTITY →VOVER =Ø
YES
544
ENTITY→AREA→TYPE =L, C OR D
NO
NO
546
CREATE ENTITY→VOVER
547
ENTITY→VOVER→LSTYLE = EID, LWIDTH=ENUM
548
ECOM →COM= F
NO
L (FIG. 7G)
YES
549
DONE
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
550
ENTITY →FILL= =Ø
NO
551
J (FIG. 7E)
ENTITY→FILL→FSTYLE=EID, FWIDTH=ENUM

FIG. 7F

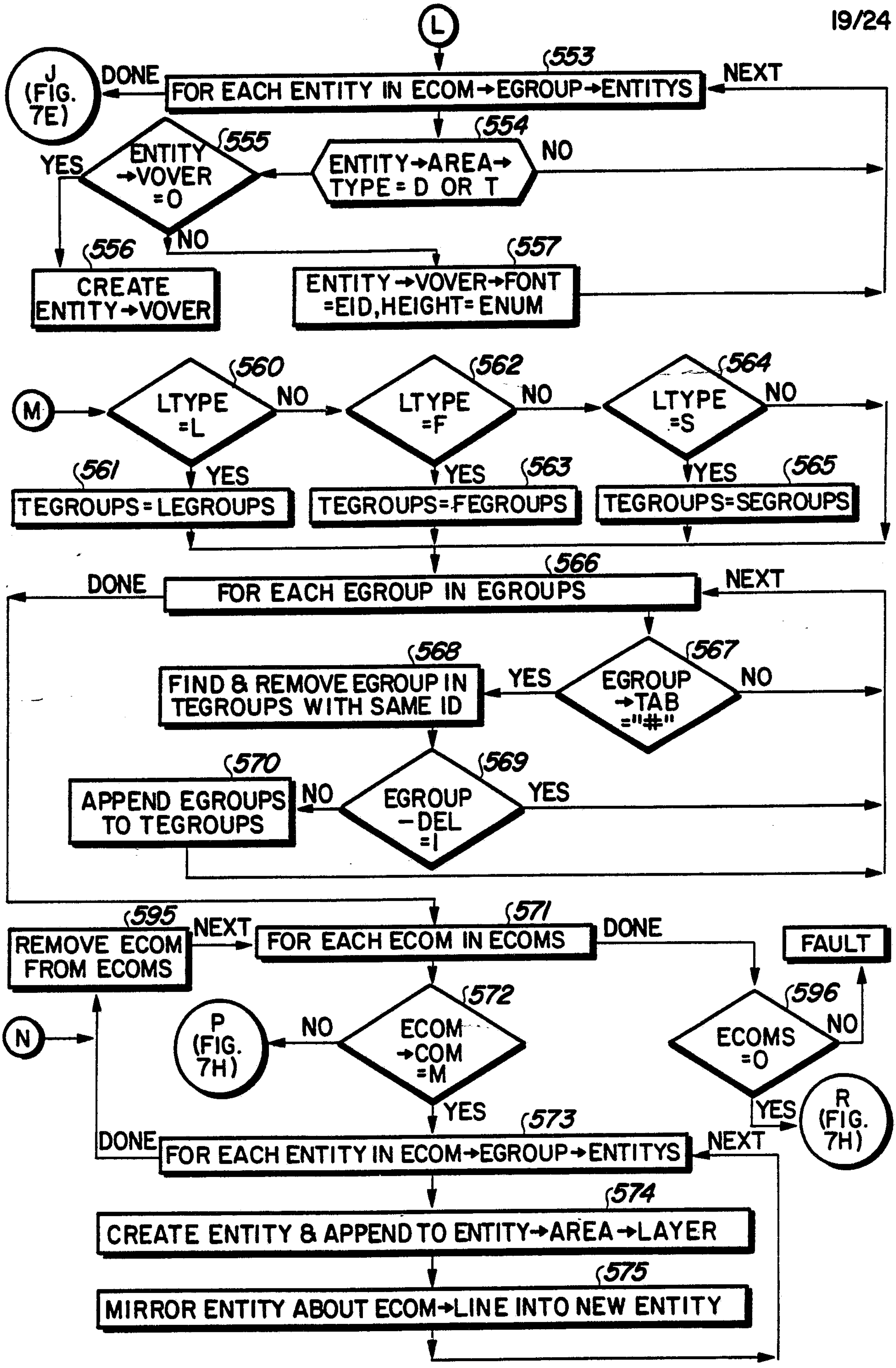

L
J (FIG. 7E)
DONE
553
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
554
ENTITY→AREA→TYPE = D OR T
NO
555
ENTITY →VOVER =0
YES
NO
556
CREATE ENTITY→VOVER
557
ENTITY→VOVER→FONT =EID, HEIGHT= ENUM
M
560
LTYPE =L
NO
562
LTYPE =F
NO
564
LTYPE =S
NO
561
YES
TEGROUPS = LEGROUPS
YES
563
TEGROUPS = FEGROUPS
YES
565
TEGROUPS = SEGROUPS
566
DONE
FOR EACH EGROUP IN EGROUPS
NEXT
568
567
FIND & REMOVE EGROUP IN TEGROUPS WITH SAME ID
YES
EGROUP →TAB ="#"
NO
570
569
APPEND EGROUPS TO TEGROUPS
NO
EGROUP −DEL =1
YES
595
571
REMOVE ECOM FROM ECOMS
NEXT
FOR EACH ECOM IN ECOMS
DONE
FAULT
N
P (FIG. 7H)
NO
572
ECOM →COM =M
596
ECOMS =0
NO
YES
YES
R (FIG. 7H)
573
DONE
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
NEXT
574
CREATE ENTITY & APPEND TO ENTITY→AREA→LAYER
575
MIRROR ENTITY ABOUT ECOM→LINE INTO NEW ENTITY

FIG. 7G

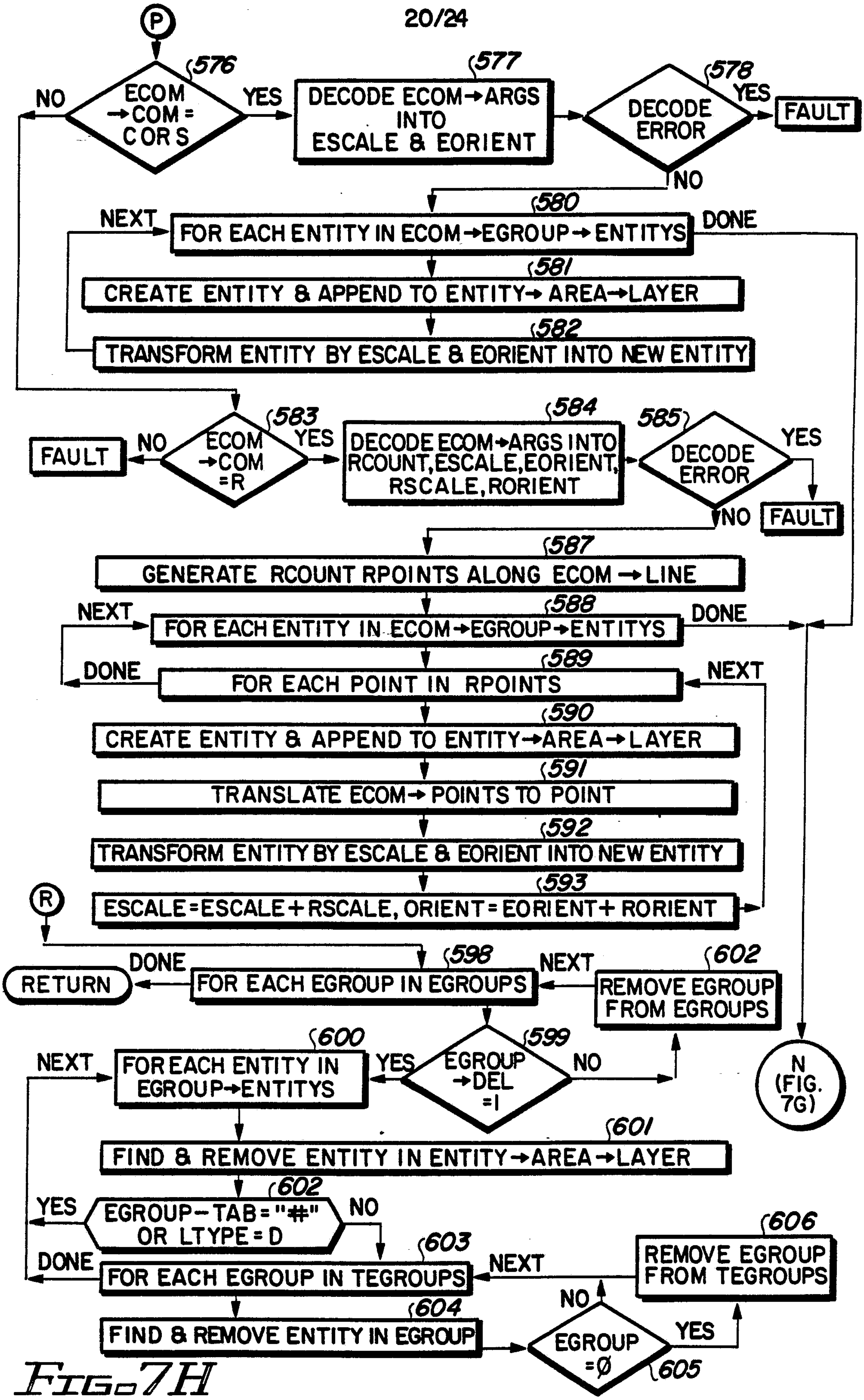

P
576
ECOM →COM = C OR S
NO
YES
577
DECODE ECOM→ARGS INTO ESCALE & EORIENT
578
DECODE ERROR
YES
FAULT
NO
580
NEXT
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
DONE
581
CREATE ENTITY & APPEND TO ENTITY→AREA→LAYER
582
TRANSFORM ENTITY BY ESCALE & EORIENT INTO NEW ENTITY
583
FAULT
NO
ECOM →COM =R
YES
584
DECODE ECOM→ARGS INTO RCOUNT, ESCALE, EORIENT, RSCALE, RORIENT
585
DECODE ERROR
YES
NO
FAULT
587
GENERATE RCOUNT RPOINTS ALONG ECOM→LINE
NEXT
588
FOR EACH ENTITY IN ECOM→EGROUP→ENTITYS
DONE
DONE
589
FOR EACH POINT IN RPOINTS
NEXT
590
CREATE ENTITY & APPEND TO ENTITY→AREA→LAYER
591
TRANSLATE ECOM→POINTS TO POINT
592
TRANSFORM ENTITY BY ESCALE & EORIENT INTO NEW ENTITY
593
R
ESCALE=ESCALE+RSCALE, ORIENT=EORIENT+RORIENT
598
DONE
RETURN
FOR EACH EGROUP IN EGROUPS
NEXT
602
REMOVE EGROUP FROM EGROUPS
600
NEXT
FOR EACH ENTITY IN EGROUP→ENTITYS
YES
599
EGROUP →DEL =1
NO
N (FIG. 7G)
601
FIND & REMOVE ENTITY IN ENTITY→AREA→LAYER
602
YES
EGROUP–TAB="#" OR LTYPE=D
NO
603
DONE
FOR EACH EGROUP IN TEGROUPS
NEXT
606
REMOVE EGROUP FROM TEGROUPS
NO
604
FIND & REMOVE ENTITY IN EGROUP
EGROUP =Ø
YES
605

FIG. 7H

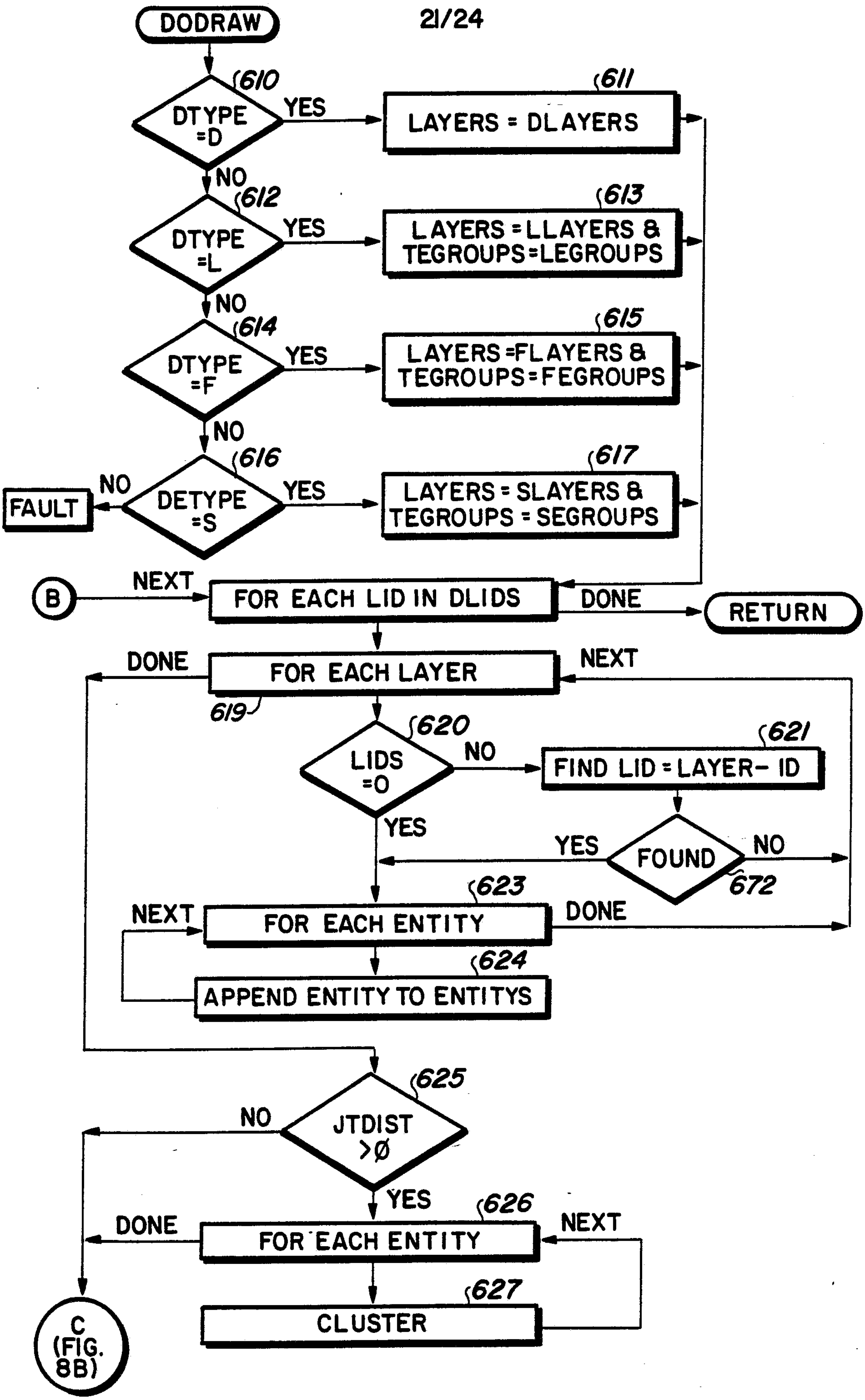

DODRAW
610
DTYPE =D
YES
611
LAYERS = DLAYERS
NO
612
DTYPE =L
YES
613
LAYERS = LLAYERS &
TEGROUPS = LEGROUPS
NO
614
DTYPE =F
YES
615
LAYERS = FLAYERS &
TEGROUPS = FEGROUPS
NO
616
DETYPE =S
YES
617
LAYERS = SLAYERS &
TEGROUPS = SEGROUPS
NO
FAULT
B
NEXT
FOR EACH LID IN DLIDS
DONE
RETURN
DONE
FOR EACH LAYER
NEXT
619
620
LIDS =0
NO
621
FIND LID = LAYER- ID
YES
YES
FOUND
NO
672
623
NEXT
FOR EACH ENTITY
DONE
624
APPEND ENTITY TO ENTITYS
625
NO
JTDIST >Ø
YES
626
DONE
FOR EACH ENTITY
NEXT
627
CLUSTER
C (FIG. 8B)

FIG. 8A

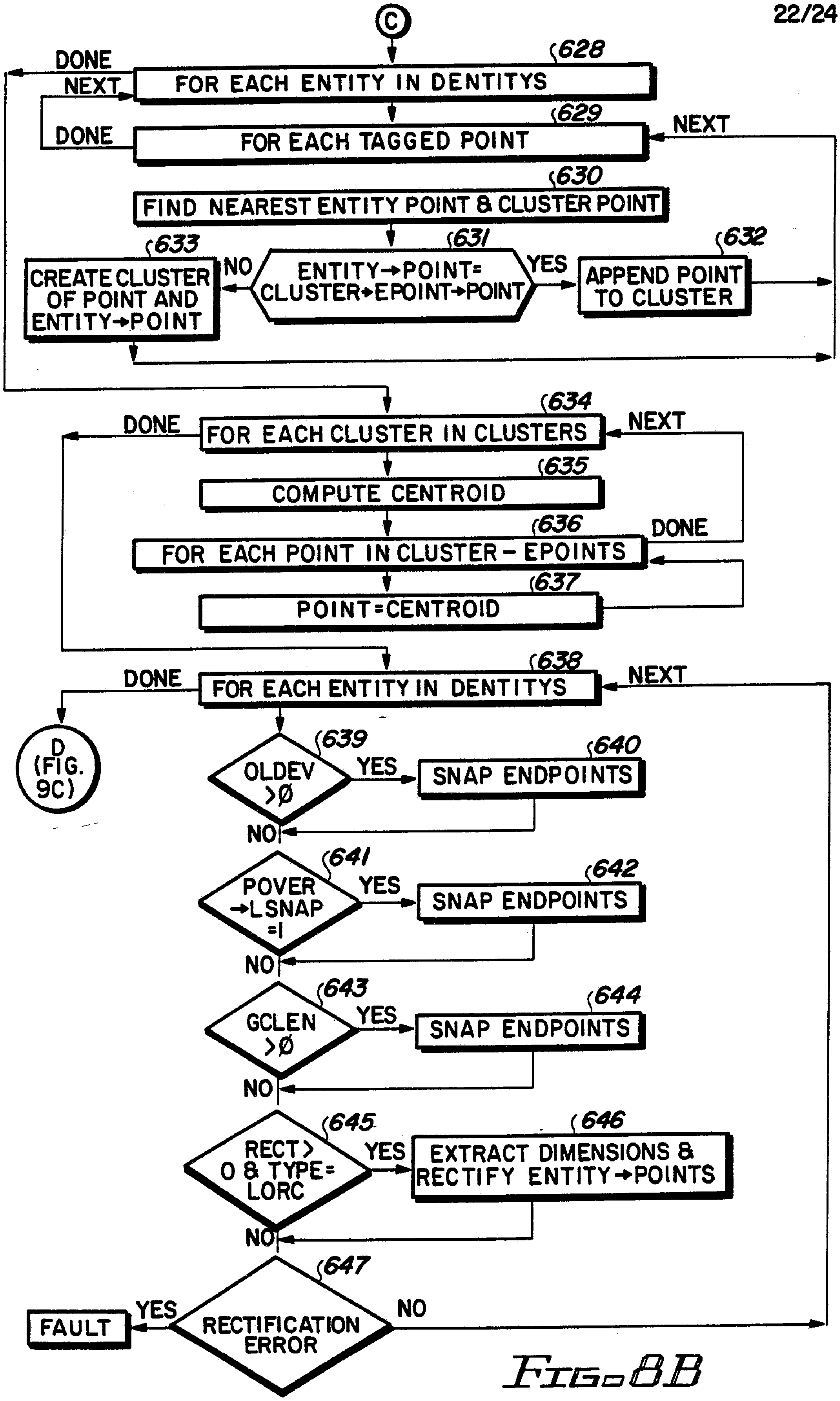

C
628
DONE
NEXT
FOR EACH ENTITY IN DENTITYS
629
DONE
FOR EACH TAGGED POINT
NEXT
630
FIND NEAREST ENTITY POINT & CLUSTER POINT
633
631
632
CREATE CLUSTER OF POINT AND ENTITY→POINT
NO
ENTITY→POINT= CLUSTER→EPOINT→POINT
YES
APPEND POINT TO CLUSTER
634
DONE
FOR EACH CLUSTER IN CLUSTERS
NEXT
635
COMPUTE CENTROID
636
DONE
FOR EACH POINT IN CLUSTER-EPOINTS
637
POINT=CENTROID
638
DONE
FOR EACH ENTITY IN DENTITYS
NEXT
D (FIG. 9C)
639
OLDEV >Ø
YES
640
SNAP ENDPOINTS
NO
641
POVER →LSNAP =1
YES
642
SNAP ENDPOINTS
NO
643
GCLEN >Ø
YES
644
SNAP ENDPOINTS
NO
645
RECT> 0 & TYPE= LORC
YES
646
EXTRACT DIMENSIONS & RECTIFY ENTITY→POINTS
NO
647
FAULT
YES
RECTIFICATION ERROR
NO

FIG. 8B

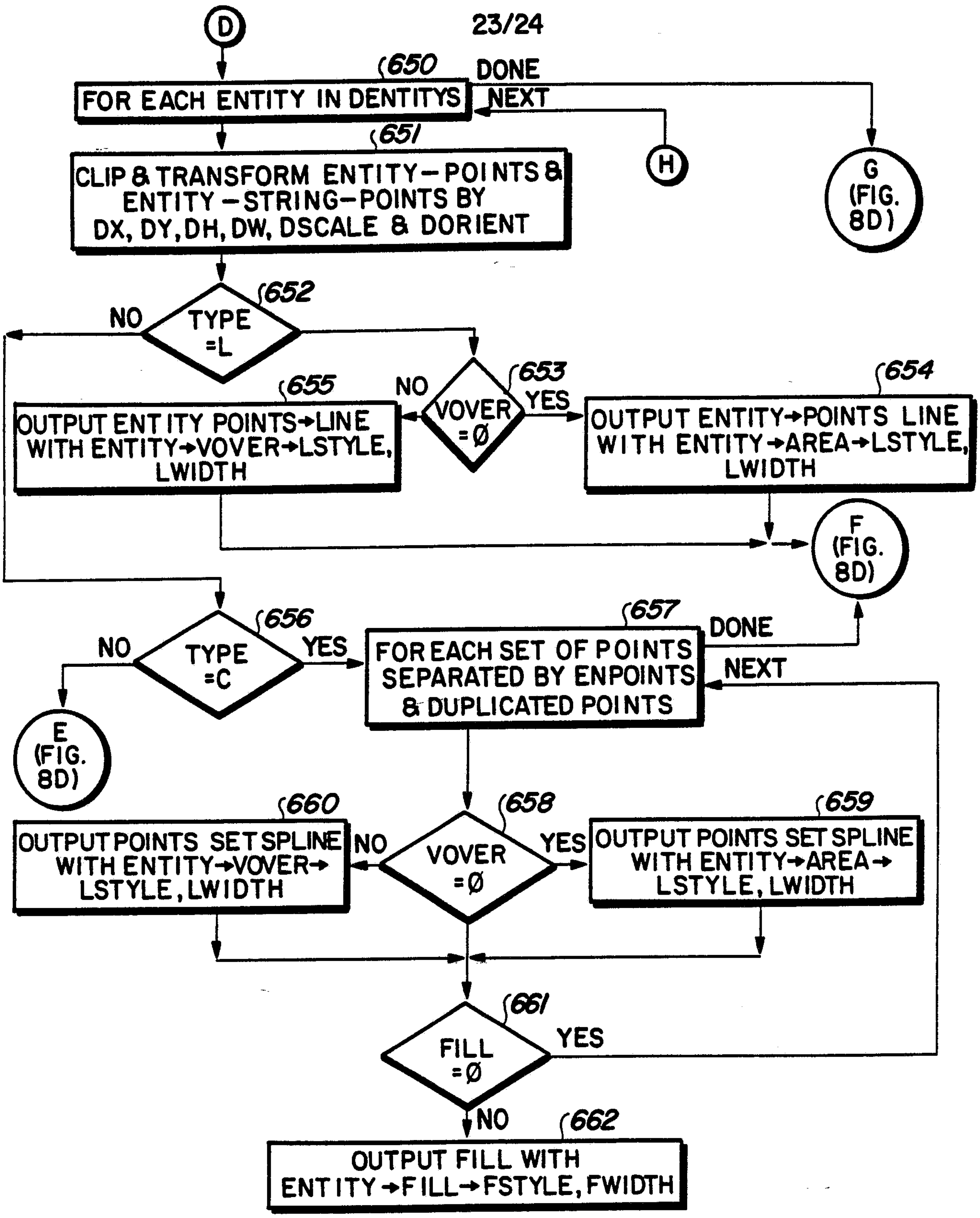

D
650
FOR EACH ENTITY IN DENTITYS
DONE
NEXT
H
G (FIG. 8D)
651
CLIP & TRANSFORM ENTITY—POINTS & ENTITY—STRING—POINTS BY DX, DY, DH, DW, DSCALE & DORIENT
652
TYPE =L
NO
653
VOVER =Ø
NO
YES
655
OUTPUT ENTITY POINTS→LINE WITH ENTITY→VOVER→LSTYLE, LWIDTH
654
OUTPUT ENTITY→POINTS LINE WITH ENTITY→AREA→LSTYLE, LWIDTH
F (FIG. 8D)
656
TYPE =C
NO
YES
E (FIG. 8D)
657
FOR EACH SET OF POINTS SEPARATED BY ENPOINTS & DUPLICATED POINTS
DONE
NEXT
658
VOVER =Ø
NO
YES
660
OUTPUT POINTS SET SPLINE WITH ENTITY→VOVER→LSTYLE, LWIDTH
659
OUTPUT POINTS SET SPLINE WITH ENTITY→AREA→LSTYLE, LWIDTH
661
FILL =Ø
YES
NO
662
OUTPUT FILL WITH ENTITY→FILL→FSTYLE, FWIDTH

FIG. 8C

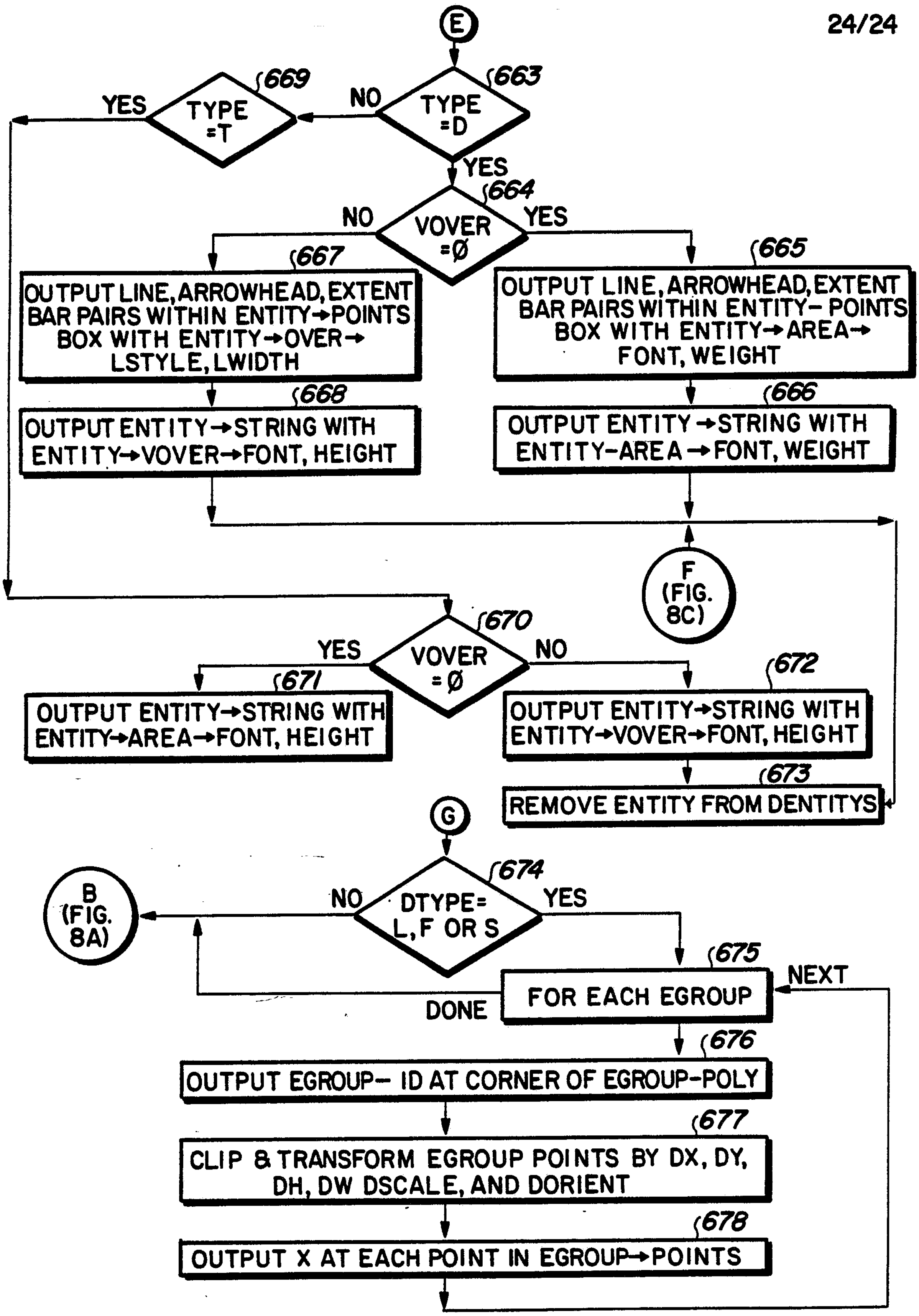

E
TYPE =D
663
NO
YES
TYPE =T
669
YES
VOVER =Ø
664
NO
YES
667
OUTPUT LINE, ARROWHEAD, EXTENT BAR PAIRS WITHIN ENTITY→POINTS BOX WITH ENTITY→OVER→ LSTYLE, LWIDTH
665
OUTPUT LINE, ARROWHEAD, EXTENT BAR PAIRS WITHIN ENTITY– POINTS BOX WITH ENTITY→AREA→ FONT, WEIGHT
668
OUTPUT ENTITY→STRING WITH ENTITY→VOVER→FONT, HEIGHT
666
OUTPUT ENTITY→STRING WITH ENTITY–AREA→FONT, WEIGHT
F (FIG. 8C)
VOVER =Ø
670
YES
NO
671
OUTPUT ENTITY→STRING WITH ENTITY→AREA→FONT, HEIGHT
672
OUTPUT ENTITY→STRING WITH ENTITY→VOVER→FONT, HEIGHT
673
REMOVE ENTITY FROM DENTITYS
G
DTYPE= L, F OR S
674
NO
YES
B (FIG. 8A)
675
FOR EACH EGROUP
NEXT
DONE
676
OUTPUT EGROUP– ID AT CORNER OF EGROUP-POLY
677
CLIP & TRANSFORM EGROUP POINTS BY DX, DY, DH, DW DSCALE, AND DORIENT
678
OUTPUT X AT EACH POINT IN EGROUP→POINTS

FIG. 8D

INTERNATIONAL SEARCH REPORT

International Application No PCT/US 90/06274

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [6]

According to International Patent Classification (IPC) or to both National Classification and IPC

IPC^5: G 06 F 15/72

II. FIELDS SEARCHED

Minimum Documentation Searched [7]

Classification System	Classification Symbols
IPC^5	G 06 F

Documentation Searched other than Minimum Documentation to the Extent that such Documents are included in the Fields Searched [8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [9]

Category [*]	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No. [13]
A	GB, A, 2044966 (DAVY INTERNATIONAL) 22 October 1980 --	
A	Zeitschrift für wirtschaftliche Fertigung & Automatisierung, vol. 82, no. 7, July 1987, Carl Hanser Verlag, (Munich, DE), H. Jansen et al.: "Handskizzierter Entwurf von CAD-Modellen mit CASUS", pages 398-404 --	
A	IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Miami Beach, Florida, 18-20 November 1985, IEEE, H. Harada et al.: "Recognition of freehand drawings in chemical plant engineering", pages 146-153 --	
A	FR, A, 2260135 (SNAM PROGETTI) 29 August 1975 ------------	

* Special categories of cited documents: [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"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)

"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

"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

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"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.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search	Date of Mailing of this International Search Report
29th April 1991	27.06.91
International Searching Authority	**Signature of Authorized Officer**
EUROPEAN PATENT OFFICE	Natalie Weinberg

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON INTERNATIONAL PATENT APPLICATION NO. US 9006274

SA 42821

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report.
The members are as contained in the European Patent Office EDP file on 10/06/91
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
GB-A- 2044966	22-10-80	None		
FR-A- 2260135	29-08-75	BE-A-	824733	15-05-75
		CH-A-	582919	15-12-76
		DE-A-	2502277	14-08-75
		GB-A-	1495791	21-12-77
		LU-A-	71703	24-06-75
		NL-A-	7501001	05-08-75

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82