"MICRO STREAM" A CAD PACKAGE FOR
STREAMLINED EXTRUSION DIES
UTILIZING A MICROCOMPUTER

A Thesis Presented to The Faculty
of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by

M. Janaka K. Jayasuriya, B.E. (Manufacturing)
Associate Member SME (U.S.A.)

November 1985
ACKNOWLEDGMENT

The author wishes to express his sincere appreciation to Dr. J.S. Gunasekera for providing the opportunity to perform this work in U.S.A, and for his advise during the course of this study.

A word of thanks to my wife Marja, for helping me to prepare this thesis and for the encouragement and support given to me throughout my studies.
# TABLE OF CONTENTS

1. **INTRODUCTION**  
   1.1 Streamlined extrusion dies  
   1.2 The objective  

2. **ON THE USE OF A MICROCOMPUTER**  
   2.1 The trend in the CAD industry  
   2.2 Choosing a microcomputer  
       2.2.1 The operating system  

3. **THE USE OF B-SPLINES IN DIE DESIGN**  
   3.1 B-Spline space curves  
   3.2 Bezier curves Vs B-Spline curves  
   3.3 On the evaluation of B-Splines,  
       "De Boor" algorithm  

4. **A COMPUTER GRAPHICS PACKAGE, THE THEORY**  
   4.1 Two dimensional graphics  
       4.1.1 A "perfect circle" drawing algorithm  
   4.2 Three dimensional graphics  
       4.2.1 Modelling objects in three dimensions  
       4.2.2 Coordinate system  
       4.2.3 The perspective transformation  
       4.2.4 Perspective depth  
       4.2.5 Properties of the screen coordinate system  
   4.3 Rotational transformations
5. THE I.B.M. PERSONAL COMPUTER

5.1 The hardware

5.1.1 The system memory map and the addressing scheme

5.1.2 The I.B.M. ROM BIOS

5.1.3 Memory addressing scheme in the 8088

5.1.4 The color video buffer

5.2 Software considerations

5.2.1 MS DOS version 2.1

5.2.2 Interfacing assembly language routines with Turbo Pascal source code

6. THE PROGRAM STRUCTURE

6.1 Run time environment specifications

6.2 The program structure and chaining

6.3 Memory management and data structures

6.4 "Micro Stream" system and the program disk specification

7. UTILITY PROGRAMS

7.1 Clr_Scr_Gra

7.2 Strg_Read

7.3 Frame [Stream.Pas]

7.4 Box_In_Gra

7.5 Reading scan codes

8. "Stream" SOFTWARE DESIGN

8.1 Modules of "Stream"

8.1.1 Strm_Typ.Pas
8.1.2 Strm_Util.Pas
8.1.3 Init_Screen
8.1.4 Strm_Inpt
8.1.5 Strm_Cal.Pas
8.2 Global data structures in "Stream"
8.3 The use of data files in "Stream"

9. "Grphics" SOFTWARE DESIGN
9.1 Modules of "Grphics"
9.1.1 Gra_Typ.Pas
9.1.2 Gra_Util.Pas
9.1.3 Gra_Uti2.Pas
9.1.4 Gra_Uti4.Pas
9.2 Global data structures in "Grphics"

10. MISCELLANEOUS CONSIDERATIONS
10.1 Turbo Pascal [version 3.0]
10.2 Limitations in the standard I.B.M. PC hardware

11. CONCLUSION

12. REFERENCE

APPENDIX A : Running "Micro Stream"
APPENDIX B : Source code
APPENDIX C : Interrupt types BIOS ROM 10H & DOS 21H
# Table of Figures

1. B-Spline curves of varying order 12
2. Multiple vertex B-Spline curves 12
3. Formation of a "knuckle" with a B-Spline curve 13
4. Local control in a B-Spline 13
5. The four Bezier blending functions for \( n = 3 \) 15
6. The six uniform non-periodic B-Spline blending functions. \( n = 5, k = 3 \). 15
7. The window transformation 20
8. Drawing a circle 21
9. Three dimensional world coordinate system 24
10. The perspective projection of point \( P_w \) onto the screen \( (P_s) \). 25
11. The \( y_s,z_s \) plane showing details of the perspective transformation 26
12. The screen coordinate system 30
13. The sign convention for rotations 31
14. The system memory map 36
15. The division of memory into paragraphs 38
16. The high-resolution graphics memory map 40
17. DOS interrupts 42
18. A section of the 8088 stack 44
19. Stack on entering an external 45
20. Program chaining in "Micro Stream" 47
21. Main menu of "Micro stream and Micro Cutter" 48
22. Modules of "Stream" 58
23. Modules of "Strm_Inpt" 60
24. Modules of "Strm_Pg2.Pas" 61
25. Modules of "Strm_Pg3.Pas"
26. Modules of "Strm_Cal.Pas"
27. Modules B_Spline procedure
28. Data structure used for product definition
29. Data structure used for electrode definition
30. The data file <file_name>.Pro
31. The data file <file_name>.MAP
32. Modules of "Grphics"
33. Modules of "Gra_Uti2.Pas"
34. Flow chart for "Draw_Main_Proc"
35. Modules of "Gra_Uti4.Pas"
1. INTRODUCTION

1.1 Streamlined extrusion dies

Metal extrusion is a forming process, in which a billet is forced to flow through a die orifice to form a product of uniform cross section along its length. The billet is usually circular in cross section whereas the product could be a complex geometry. Two types of dies can be utilized in the extrusion process.

[a] Flat faced die (shear die)

[b] Streamlined die (converging die)

Out of the two, the flat faced die has found wide acceptance in industry, mainly due to the ease of manufacture. However, numerous functional deficiencies inherent in these dies (1-8), prevent their successful utilization in certain extrusion applications. This has become evident on attempting to extrude "difficult to extrude materials" such as whisker reinforced Aluminium alloys, utilized in the aerospace industry (6). The design of streamlined dies overcome these functional deficiencies to a great extent, enabling these dies to function successfully in areas where the shear die fails. However, due to difficulties experienced in their design and manufacture, the extrusion industry has shown a reluctance to use streamlined dies. The design concepts for these dies are now well established and have been incorporated into a dedicated CAD package known as "STREAM" (5,6,8), which is available commercially (25). It can be utilized to design any streamlined die
cavity including product geometries with re-entry shapes. In a previous study (9), the author investigated the problems associated with the manufacture of these dies. In it, a suitable method for manufacturing these dies was established and numerous software tools were developed to aid this process. The resulting CAM package known as "CUTTER", was interfaced with "STREAM" to produce a stand-alone, user friendly CAD/CAM system for streamlined extrusion dies. This software was written in FORTRAN-77 and runs on Digital Equipment Corporation's VAX 11/700 series minicomputers, under the VMS operating system. It utilizes the services of the following software packages.

[a] TEKTRONIX PLOT-10 routines

These are a set of device drivers that are utilized by "STREAM" when performing interactive graphics on TEKTRONIX terminals.

[b] "MOVIE" graphics package

This is a package capable of performing many advanced computer graphics operations on various graphic displays. It is fully described in the "MOVIE" user manual (11). The CAD package "STREAM" utilizes "MOVIE" to display the die cavity design and the CAM package "CUTTER" utilizes it to display the tool paths used for machining these dies.
1.2 The objective

The objective of this work is best stated in point format as follows.

[a] Enhancement of the design capabilities of "STREAM" through the implementation of B-Spline theory.

[b] Installation of "STREAM" on a microcomputer including the provision of full two and three dimensional graphics support.

Throughout the study, a heavy emphasis is placed on understanding the microcomputer and its programming environment. Consequently, the study will identify the limiting factors (if there are any) of such an environment.

It is appropriate to state the following with regard to the scope of this work. "STREAM" is a comprehensive CAD package offering many features for the analysis of streamlined extrusion dies. However, the scope of this work is confined to installing those functions that are essential for running the die design section of "STREAM".
2. **On the Use of a Microcomputer**

2.1 **The trend in the CAD industry**

Prior to the introduction of the 16-bit microprocessor based computers, all major software development was carried out in minicomputer or mainframe installations. The cost of these computers were high and the use of such facilities were constrained to large commercial or academic establishments. However, with the technology advances in computer architecture, and in particular the microprocessor, microcomputers now offer a powerful program development environment at a greatly reduced cost. This has prompted many software developers to consider the personal computer as a suitable machine for software development. The following, based on a study published in reference 10, discusses the current usage of microcomputers in the CAD industry.

A typical CAD/CAM system would consist of a host computer coupled with a number of work stations. Often, these work stations are capable of performing many local computations and serve to augment the power of the host computer. However, one major disadvantage is that the computing power of these work-stations are not accessible to its user for performing other non CAD related activities. Consequently, more and more users are now opting for smaller, microcomputer based systems. These systems frequently yield 70% of the benefits for approximately 20% of the cost. This trend is expected to accelerate as more powerful machines and displays are introduced towards late 1985. At the time when Daratech research (10) was performing its study, the industrial leader in microcomputer based systems Autodesk, was selling nearly 2000 copies a month
of AutoCAD™ which is designed to function on some 30 personal computer models.

Further, there is an increased demand for microcomputer based application software, by the users who already possess such hardware. In the area of CAD, the thinking of the two leading vendors are reflected in their current or near future products.

[a] I.B.M. is expected to introduce to the domestic market an I.B.M. PC AT based system similar to MICRO CADAM, the personal computer based system its Japanese company is marketing in the far east. It is expected to run on the model 5550 personal computer under MS DOS, once high performance displays are available for the ATs in late 1985.

[b] Intergraph, in mid 1985, began manufacturing engineering work stations based on the National Semiconductor 32032, 32 bit microprocessor. This multifunction engineering work station Interpro 32, can operate as a terminal on Intergraph's standard system while providing a UNIX environment for third party software and a MS DOS capability for I.B.M. personal computer applications.

Another significant factor supporting the use of microcomputers is the development of Local Area Networking (LAN) technology which has facilitated distributed processing, while providing easy access to centralized mass storage, data bases and if required, gateways to other communication networks (telephone, satellite) providing access to more powerful computers.

The above discussion identifies a definite trend in the industry
to move away from centralized mini/mainframe computer environments to
distributed microcomputer based systems. Having recognised this trend,
it is instructive to investigate the programming environment that
is offered by these computers for CAD/CAM software development. This
instigated the current work which performs this investigation through the
installation of a working CAD system "STREAM".

2.2 Choosing a microcomputer

The current choice between computers narrows down to two major
systems.

[a] Motorola 68000 (16/32. i.e. a 16 bit data bus and 32 bit
    CPU registers) based systems
[b] Intel 8086 (16/16) or 8088 (8/16) based systems

Of the two, the MC68000 is a more powerful microprocessor. It is
also a more convenient processor to use, in that it has a linear address
space as opposed to the segmented addressing of the 8086 family.
However, several microcomputers based on the 68000 has demonstrated that
the power of the CPU alone does not automatically make that computer the
correct choice. There are various other factors that are equally
important in software development. Several of these are listed below.

[a] The availability of technical information, both on hardware
    and the DOS.
[b] Accessibility of system hardware features through user software.
[c] The availability of third party software development tools and hardware add-ons.
[d] Compatibility with other popular personal computers thus facilitating the portability of software.
[e] A dedication to the continuous development of system level software (e.g., DOS) by the manufacturer.
[f] Reliability and popularity of the computer.

For the present work, the I.B.M. PC was chosen as the most appropriate microcomputer. Although it does not utilize the most powerful processor available, it does provide an excellent program development environment. If required, the reduced processor power can be increased through some excellent, proven third party add-ons and through the use of the 8087 co-processor (for which I.B.M. has included a socket in the architecture).

2.2.1 The operating system

At present, three operating systems are available for the I.B.M. PC, the UCSD P-System, I.B.M. PC-DOS and Microsoft corporation's MS DOS. Out of these, the latter two are almost identical and are far more efficient than the P-System implementation for this computer. They also provide excellent support for accessing the routines resident in the computer's ROM. Further, MS DOS is the lower end member of a family of operating systems that is offered by Microsoft corporation. At the
top is a UNIX based microcomputer operating system "Xenix". This is an
direct indication by Microsoft, of their commitment to providing better
system level software for microcomputers. Consequently all software
development in this work is carried out in the MS DOS 2.1 environment.
In the design of a streamline extrusion die, the transition geometry connecting the entry and the exit cross sections of a die, must be defined. At present, several definitions are permitted in specifying the curve type (4), the cubic spline being the most commonly utilized. However, a die requiring a different transition geometry cannot be accommodated using this set of limited curve definitions. Therefore, what is required is a more versatile method for determining the transition geometry. Two types of parametric space curves can be utilized to provide this flexibility.

[a] Bezier curves
[b] B-Spline curves

Of the two, B-Splines provide greater flexibility (see section 3.2) and is chosen to be implemented in this work. The theory of these curves is discussed below and the implementation is given in chapter 8. Also a comparison of Bezier and the B-Spline basis approaches is presented in section 3.2.

3.1 B-Spline space curves

The discussion in this section is based on the material presented in reference 13. The parametric definition of a B-Spline space curve is given by the following.

\[ P(t) = \sum_{i=0}^{n} P_i N_{i,k}(t) \]  

---------- 3.1
Where: \( t \) is a parameter such that \( 0 \leq t \leq t_{\text{max}} \).

\( P(t) \) is the position vector along the curve as a function of \( t \).

\( P_i \) are the \( n+1 \) defining polygon vertices.

\( N_{i,k}(t) \) are the weighting function of order \( k \).

(\( k \) controls the order of continuity of the curve)

Mathematically, a B-Spline curve is defined as a polynomial spline function of order \( k \) (degree \( k-1 \)) since it satisfies the following.

[a] The function of \( P(t) \) is a polynomial of degree \( k-1 \) on each interval \( x_i \leq t < x_{i+1} \). Thus a B-Spline of order four is a piecewise cubic spline.

[b] \( P(t) \) and its derivatives of order 1, 2, \ldots, \( k-2 \) are all continuous over the entire span.

The weighting functions are defined by the following recursive formulas.

\[
N_{i,1}(t) = \begin{cases} 
1 & \text{if } x_i \leq t < x_{i+1} \\
0 & \text{otherwise}
\end{cases} \quad 3.2(a)
\]

\[
N_{i,k}(t) = \frac{(t - x_i) N_{i,k-1}(t) - (x_{i+k} - t) N_{i+1,k-1}(t)}{x_{i+k} - x_i} \quad 3.2(b)
\]

Since the denominators of 3.2(b) can become 0, the convention 0/0 = 0 is adopted in evaluating these weighting functions. The values of \( x_i \) are the elements of a knot vector that relate the parameter \( t \) to the
control points. These values are simply a series of real integers $x_i$, such that $x_i \leq x_{i+1}$ for all $x_i$. For an uniform non-periodic B-Spline the elements of the knot vector can be obtained from the following set of rules (see reference 15 for uniform periodic B-Splines).

\[
\begin{align*}
    x_i &= 0 & \text{if } i < k \\
    i - k + 1 & \text{ if } k \leq i \leq n \\
    n - k + 2 & \text{ if } i > n
\end{align*}
\]

It is important to understand the use of the knot vector since it provides the inherent added flexibility of B-Splines. This of course, is in addition to the variability provided by the position of the control vertices. In general, the limits of the parameter is set using the knot vector. For example, if the knot vector is $x_0, x_1, \ldots, x_{\text{max}}$ then $x_0 \leq t \leq x_{\text{max}}$. The number of intermediate knot values depend on the number of spans in the defining polygon. A duplicated intermediate knot value indicates a multiple vertex (a span of zero length) in the defining polygon. A triplet of intermediate knot values indicate three coincident vertices (two zero length spans) in the defining polygon.

In addition to the knot vector elements, the order $k$ must be specified. $k = 2$ is of special interest since the resulting B-Spline becomes a segmented straight line that passes through all the vertices. When there are no multiple vertices, the parameter $t$ varies from $0$ to $n - k + 2$ over the entire curve. Therefore, for a third order curve defined using 5 vertices, $k = 3$ and $t_{\text{max}} = 3$. The corresponding knot vector is $[0 0 0 1 2 3 3 3]$. Note that the order of the curve is reflected in the knot vector through the specification of knots of multiplicity $k$ (=3) at both ends of the knot vector. For a second order curve, $k = 2$, with
the same defining polygon, the knot vector is \([0 \ 0 \ 1 \ 2 \ 3 \ 4\ 4]\). Some specific features of B-Splines are illustrated through figures 1-4 (13).

Figure 1. B-Spline curves of varying order

Figure 2. Multiple vertex B-Spline curves
Figure 3. Formation of a "knuckle" with a B-Spline curve

Figure 4. Local control in a B-Spline (Fifth order seven-point polygon. No multiple vertices)
In figure 1, there are no multiple vertices and the only variable is the order k. As the order increases, the curve moves away from the defining polygon and is said to become "tighter". Note that the slope of the two ends of all curves are the same and is governed by the slope of the two end spans of the defining polygon. In figure 2, multiplicity at point [3,9] increases and the order of the curve is made equal to the total number of vertices. As an example, in the fourth order curve 4/4, k = 4 and equals the number of vertices 4. In 5/5, k = 5 and consequently there is a double vertex at [3,9] and so on for the others. This figure illustrates, as the number of coincident vertices increase, the curve gets drawn towards that point. In figure 3, there is double vertex at [7,3]. Notice now, the third order curve forms a "knuckle" through this point. Likewise, to obtain a "knuckle" for a fourth order curve, a triplet of vertices are required at [7,3]. The last figure 4 illustrates local control (also see section 3.2) that is possible with B-Splines.

3.2 Bezier curves Vs B-Spline curves

The basis of the Bezier curve is the Bernstein polynomial which has two characteristics that limits the flexibility of the resulting curves. First, the number of polygon vertices fixes the order of the resulting polynomial which defines the curve. For example, a cubic curve must be defined by a polynomial with four vertices. A polynomial with six vertices will always produce a fifth degree curve. However, the order of the B-Splines can be changed without changing the number of vertices.
The second limiting characteristic is due to the global nature of the Bernstein basis. This can be best understood by observing the behavior of these weighting functions (12).

Figure 5. The four Bezier blending functions for $n=3$.

Figure 6. The six uniform non-periodic B-Spline blending functions. $n=5$, $k=3$

In figure 5, the four blending functions are spread over the entire span of the parameter, thus their effect is felt at any value of $t$. However, in figure 6, at most only three functions (equal to the order) are non-zero within a span, thus not all the blending functions
influence all B-Spline polynomials. Consequently local control not possible with Bezier curves can be achieved with B-Splines.

3.3 On the evaluation of B-Splines, "De Boor" algorithm

[a] Equation 3.1 suggests an immediate method for the evaluation of B-Splines. Given the number of vertices $(n + 1)$ and $k$, all the numbers $N_{i,k}(t)$ are evaluated using equations 3.2. As an example, let $k = 4$ and $n + 1 = 4$. Then the blending functions that require to be evaluated are:

$$
N_{0,4}(t), \ldots, N_{3,4}(t) \\
N_{0,3}(t), \ldots, N_{4,3}(t) \\
N_{0,2}(t), \ldots, N_{5,2}(t) \\
N_{0,1}(t), \ldots, N_{6,1}(t)
$$

Since all these are functions of the parameter $t$, they need to be evaluated at each parameter increment. Once these values are known, equation 3.1 is utilized to evaluate $P(t)$. This algorithm requires the explicit evaluation of the basis functions $N_{i,k}(t)$. The following algorithm is computationally more efficient and numerically stable.

[b] "De Boor" algorithm

The complete derivations of this algorithm is given in reference 16. The following presentation concentrates on those details that must be understood prior to the implementation of this algorithm. Equation 3.1 can be re-written as follows:
Now, from equation 3.2(a), \( N_{i,l}(t) = 1 \) for \( x_i \leq t < x_{i+1} \) and zero otherwise. Therefore it follows:

\[
P(t) = \sum_j A_i^{[j]}(t) N_{i,k-j}(t) \quad \text{--------- 3.3(a)}
\]

Where

\[
A_i^{[j]}(t) = \begin{cases} 
A_i & \text{j=0} \\
\frac{t-x_i}{x_i+k-j-x_i} A_i^{[j-1]}(t) + \frac{x_{i+k-j}-t}{x_i+k-j-x_i} A_i^{[j-1]}(t) & \text{j>0}
\end{cases} \quad \text{--------- 3.3(b)}
\]

Now, from equation 3.2(a), \( N_{i,l}(t) = 1 \) for \( x_i \leq t < x_{i+1} \) and zero otherwise. Therefore it follows:

\[
P(t) = A_i^{[k-1]}(t) \quad x_i \leq t < x_{i+1} \quad \text{--------- 3.4}
\]

Now then, having found \( i \) such that \( t \in [x_i, x_{i+1}) \), generate all the entries in the following table using equation 3.3(b).

\[
A_{i-k+1}(t) \\
A_{i-k+2}(t) \quad A_{i-k+2}(t) \\
\vdots \quad \vdots \\
A_{i-1}(t) \quad A_{i-1}(t) \ldots \quad A_{i-1}(t) \\
A_i(t) \quad A_i(t) \ldots \quad A_i(t) \quad A_i(t)
\]

Then the right-most entry is the desired number \( P(t) \). Simplifying the notation yields the following relationships which are more convenient for implementation.
Set \( A(r,s) = A_{i-k+r}^{[s-1]}(t) \) \( r = s, \ldots, k; \ s = 1, \ldots, K \) \( ----- 3.5 \)
\[
DP(r) = x_{i+r} - t \\
DM(r) = t - x_{i-k+r} \quad r = 1, \ldots, K
\]

Then
\[
A(r,1) = A_{i-k+r} \quad r = 1, \ldots, K \quad ----- 3.6(a)
\]
\[
A(r,s+1) = (DM(r) \times A(r,s) + DP(r-s) \times A(r-1,s))/(DM(r) + DP(r-s)) \\
r = s+1, \ldots, K; \ s = 1, \ldots, K-1 \quad ----- 3.6(b)
\]

Note that
\[
DM(r) + DP(r-s) = t - x_{i-k+r} + x_{i+r-s} - t \\
= x_{i+r-s} - x_{i-k+r} \geq x_{i+1} - x_i > 0
\]

Therefore coincident points cause no additional difficulties if as assumed, \( i \) is chosen such that \( x_i \leq t < x_{i+1} \).
Various theoretical details of computer graphics, required for the development of a graphics package are presented in this chapter. It is based on the guidelines outlined in reference 12. The implementation details of the complete graphics package is presented in chapter 9.

The design of a graphics system can be greatly facilitated by defining a functional domain for the system. Then, it is possible to examine each function set separately to achieve the desired completeness within an individual set. This approach allows the development of the system in an orderly manner, and will yield a well defined functional domain for the resulting graphics system, facilitating the addition of subsequent extensions. For the current package, the initial function set is limited to the following.

[a] Graphic primitives. These are used to display straight lines, text strings and circular arcs.

[b] Viewport function. The provision of this function allows the user to be detached from the details of the screen coordinate system and to follow a convenient object space (world) coordinate system.

[c] Three-dimension transformation function for rotation of the model in the object space.

Note: Although a hidden line removal function is not implemented, it is given due consideration during the construction of other functions. This will facilitate the addition of this function at a subsequent time.
4.1 Two dimensional graphics

Handling two-dimensional graphics is particularly easy, since both the world and screen coordinate systems are essentially two dimensional. The world to viewport transformation can be achieved by the use of the following equations. Define the edges of the window to be at \( x = W_{xl}, x = W_{xr}, y = W_{yb} \) and \( y = W_{yt} \) (see Figure 7), all measured in world coordinates. The corresponding edges of the viewport are at \( x = V_{xl}, x = V_{xr}, y = V_{yb} \) and \( y = V_{yt} \), all measured in screen coordinates.

\[
\begin{align*}
W_{yt} & - & - & - & - & W_{yt} \\
\text{window} & - & - & - & - & \text{viewport} \\
\text{screen} & - & - & - & - & \text{screen} \\
V_{yt} & - & - & - & - & V_{yt} \\
V_{yb} & - & - & - & - & V_{yb} \\
W_{xb} & - & - & - & - & W_{xb} \\
W_{xt} & - & - & - & - & W_{xt}
\end{align*}
\]

Figure 7. The window transformation

Here, use of the window is to define what is required to be displayed and the viewport specification serve to determine where to place it on the screen. In the current package, the window is fixed and completely encloses the user defined geometry (the model) in the object space, at all times. Consequently a clipping function is not required.

Now, the point \( x_w, y_w \) in world coordinates transforms into the
point $x_s, y_s$ in the screen coordinates as follows.

$$
x_s = \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}} (x_w - w_{xl}) + V_{xl} \quad 4.1(a)
$$

$$
y_s = \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}} (y_w - w_{yb}) + V_{yb} \quad 4.1(b)
$$

4.1.1 A "perfect circle" drawing algorithm

The most common method of calculating the $x, y$ coordinates on the perimeter of a circle is to resolve the radius in the $x$ and $y$ directions [see figure 8(a)].

$$
x_n = r \cos(\theta) \quad y_n = r \sin(\theta)
$$

![Diagram of a circle with coordinates](image-url)

Angle measurements counter clockwise $> 0$

Figure 8. Drawing a circle

However, in this method, the evaluation of each coordinate pair requires the computation of $\sin(\theta)$ and $\cos(\theta)$ ($\theta$ being
a continuous variable), which can considerably decrease the speed of computation. An improvement on this would be a method that required the calculation of the trigonometric functions only once during the circle drawing. The following incremental method satisfies this requirement.

From figure 8(b):

\[ x_{n+1} = r \cos(\delta + \theta) \]
\[ y_{n+1} = r \sin(\delta + \theta) \]

expanding these gives:

\[ x_{n+1} = r[\cos(\delta) \cos(\theta) - \sin(\delta) \sin(\theta)] \]
\[ y_{n+1} = r[\sin(\delta) \cos(\theta) + \cos(\delta) \sin(\theta)] \]

However \( \cos(\theta) = x_{n} / r \) and \( \sin(\theta) = y_{n} / r \)

Substituting these into equations 4.2(a) and 4.2(b):

\[ x_{n+1} = x_{n} \cos(\delta) - y_{n} \sin(\delta) \quad \text{(4.3a)} \]
\[ y_{n+1} = x_{n} \sin(\delta) + y_{n} \cos(\delta) \quad \text{(4.3b)} \]

Now, in evaluating equations 4.3(a) and 4.3(b), \( \cos(\delta) \) and \( \sin(\delta) \) need to be evaluated only once, outside the loop.

4.2 Three dimensional graphics

The fundamental difference between two-dimensional and three dimensional graphics is that, an object space defined in a three dimensional world coordinate system need to be transformed onto a two dimensional
screen coordinate system. This can be implemented through a suitable projection. In order to retain the realism of the object space scene, such a technique must include the depth information in the transformation. The perspective transformation conveys the depth information by making distant objects smaller than the near ones. Further, the properties of the perspective transformation facilitates image space hidden line removal techniques.

4.2.1 Modelling objects in three dimensions

An appropriate graphics primitive to be used in three dimensions is the polyhedron. This is due to the fact that by increasing the number of faces, it is possible to construct a polyhedron that will approximate any solid object.

A polyhedron can be modelled by defining its faces, each face a planar polygon, that can in turn be modelled by an ordered listing of the vertices of the polygon or by a similar list of its edges. Then, the scene can be displayed as a "wire frame" model by connecting all the edges. In hidden line removal, the faces of these polygons take a primary importance in determining hidden surfaces.

Now, it is possible to identify two important classes of information in three dimensional models. One is the geometry which is concerned with measurement such as the location of a point or dimensions of an object. The other is the topology which records the structure of the scene, i.e. how points form a polygon, polygons form objects and objects form scenes.
4.2.2 Coordinate system

In order to define the geometry of the object, the right handed cartesian coordinate system shown in figure 9 is used.

![Coordinate system diagram](image)

Figure 9. Three dimensional world coordinate system

As indicated in this figure, in order to generate three dimensional scenes, a viewpoint, viewing direction (line of sight) and an aperture must be specified. These parameters are analogous to the adjustment made by a photographer when taking a picture of a real scene. A position for the camera, a direction in which to point it and the lens specification determines how much of the scene will be included in the picture. The application of these parameters in computer graphics, will become clear during the construction of the equation for perspective transformation.

4.2.3 The perspective transformation

It is convenient to perform the perspective transformation in an eye coordinate system. Therefore, the eye coordinate system is
defined to be a left handed cartesian system as shown in figure 10.

Consequently it is necessary at some point to convert from the object description in the world coordinates to the eye coordinate system. If it is assumed that \( z_e \) is the line of sight and is always parallel to \( z_w \), and the viewpoint is at the origin of the eye system, the following transformation achieves the desired result.

\[
[x_e, y_e, z_e \, 1] = [x_w \, y_w \, z_w \, 1] \, T
\]

where

\[
T = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -1 & 0 \\
0 & 0 & t_z & 1
\end{bmatrix}
\]

This translates the world origin, a distance of \( t_z \) along the \( z_w \).
axis. Also the resulting z values are inverted so that they are correct relative to the left handed eye coordinate system. No change to $x_w$ or $y_w$ is required. Notice that the assumptions stated above do not result in any loss of generality since any viewpoint can be transformed to coincide with the above defined eye coordinate system through the application of a set of appropriate transformations. Now, the perspective display can be generated by simply projecting each point of an object onto the plane of the display screen (see figure 10). Please refer to figure 11 during the following constructions.

Figure 11. The $y_s,z_s$ plane showing details of the perspective transformation.

Triangles $OQ'P'$ and $OQP$ are similar. Therefore:
Likewise a similar construction in the $x_e, z_e$ plane yields:

$$x_s = \left(\frac{x_e}{z_e}\right) D$$

The numbers $x_s$ and $y_s$ can be converted to dimensionless numbers by dividing by the screen size $S$.

Therefore

$$x_s = \left[\frac{x_e}{(S * z_e)}\right] D \quad \text{-------- 4.4(a)}$$

$$y_s = \left[\frac{y_e}{(S * z_e)}\right] D \quad \text{-------- 4.4(b)}$$

These numbers can be mapped into a predefined viewport by using equations 4.1(a) and 4.1(b). Note that this transformation ensures that straight lines in object space are mapped to straight lines in the image space. Therefore, in order to reconstruct a straight line in the image space, it is only required to transform the two end points of a straight line segment in the object space, which greatly reduces the number of points transformed.

The ratio $D/S$ in equation 4.4, defines the aperture. If it is small the aperture will be broad, thus producing an image similar to that of a wide-angle lens. Large ratios of $D/S$ specifies a narrow aperture, corresponding to a telephoto view ($S$ is a constant). With reference to the window (see section 4.1), a small $D/S$ ratio moves the window towards the viewpoint, thus enclosing more details of the world scene within the viewframe and a large $D/S$ moves the window away from the viewpoint, thus enlarging the details of the world scene and enclosing less details within the viewframe.
4.2.4 Perspective depth

The function of hidden line removal requires the depth information of each point to be preserved during the transformation. This is achieved by ensuring that planes in the object space transform into planes in the screen system. Using the general equation of a plane, we can write the following.

\[ ax_e + by_e + cz_e + d = 0 \] \[ \Rightarrow a'x_s + b'y_s + c'z_s + d' = 0 \]

Reordering this equation and substituting for \( x_s \) and \( y_s \) from equation 4.4 gives:

\[ z_s = \frac{\alpha + \beta}{z_e} \]

where \( \alpha = \frac{-d'}{c'} \)

\[ \beta = -\frac{(a'x_eD) / (Sc')} + (b'y_eD) / Sc' \]

From equation 4.6, it is clear that the choice of alpha and beta will influence the depth resolution. Let \( D \leq z_e \leq F \). Now, it is possible to maximize the depth resolution by choosing alpha and beta so that \( z_e = D \) maps into smallest \( z_s \) value and \( z_e = F \) into the largest. For convenience let \( 0 \leq z_s \leq 1 \).

Then using equation 4.6

\[ 0 = \alpha + \beta / D \quad \text{and} \]

\[ 1 = \alpha + \beta / F \]

Solving for alpha and beta yields:

\[ \alpha = \frac{F}{(F - D)} \]

\[ \beta = -\frac{(F \times D)}{(F - D)} \]
Substituting in 4.6:

\[ z_s = \frac{F}{(F - D)} - \frac{F \cdot D}{((F - D) \cdot z_e)} \]

This can be rewritten as

\[ z_s = \frac{S(z_e/D - 1)}{(1 - D/F)(Sz_e/D)} \] 4.7

Now let \( w = Sz_e/D \) and using equations 4.4 and 4.7, the perspective transformation that accomplishes the mapping of a three dimensional eye model to the two dimensional image space can be written as follows.

\[
\begin{align*}
X_s &= \frac{x_e}{w} = \frac{y_e}{w} = \frac{S(z_e/D - 1)}{(1 - D/F)w} \\
W &= \frac{Sz_e}{D}
\end{align*}
\] 4.8

Note that \( w \) is directly proportional to the depth of a point from the viewpoint along the \( z_e \) axis. If a point lies in the plane of the viewpoint \( (z_e = 0) \), \( w = 0 \). For points in front of the viewpoint \( (z_e > 0) \), \( w > 0 \) and for points behind the viewpoint \( (z_e < 0) \), \( w < 0 \). Consequently it is appropriate to consider \( w \) as a fourth coordinate that contain the "perspective information".

4.2.5 Properties of the screen coordinate system

In order to understand how the perspective transformation aid the hidden line removal, the properties of the screen coordinate system must be understood. From equation set 4.8, it is clear that \( z_e = 0 \) is undefined in the screen system, but can be thought of as being at infinity in the \(-z_s\) direction. Now then, the viewpoint is at \( z_e = 0\),
thus it can considered to be at infinity relative to the screen coordinate system. Consequently all rays emanating from the eye are parallel to the $z_s$ axis. Now, referring to figure 12, it is easy to understand how this facilitates the hidden line removal process.

Here, any point $P$ inside the "hidden region" generated due to polygon $Q$, will not be visible. This observation is correct only because the rays emanating from the eye can be considered to be parallel to $z_s$.

![Figure 12. The screen coordinate system](image)

It is difficult to extent this reasoning to the eye coordinate system since the rays are emanating from the eye at various angles.

### 4.3 Rotational transformations

In the preceding discussion on three dimensional graphics, the line of sight (viewing direction) was determined to be fixed along the $z_e$ axis. In order to enable the user to view the object from any direction, three dimensional rotation capability is implemented through the rotation
transformation. These rotations must be done in object space (i.e. in the world coordinate system) to ensure that the subsequent view transformations will not yield an object in the eye coordinate system, that will not be visible from the viewpoint (as an example, the object might transform to a region behind the viewpoint). The sign convention depicted in figure 13 is adopted in all rotation transformations.

![Figure 13. The sign convention for rotations](image)

The three transformations in the x, y and z axis are as follows.

\[
T_x = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\theta_x) & -\sin(\theta_x) \\
0 & \sin(\theta_x) & \cos(\theta_x)
\end{bmatrix}
\]

\[
T_y = \begin{bmatrix}
\cos(\theta_y) & 0 & \sin(\theta_y) \\
0 & 1 & 0 \\
-\sin(\theta_y) & 0 & \cos(\theta_y)
\end{bmatrix}
\]
In a rotation function, the result achieved through the successive application of these transformations can be obtained with a single transformation matrix by concatenating the sequence. For this, the order of application of the transformations must be predetermined and preserved during the concatenation. If the order of rotation is fixed to be about $x$, $y$ then $z$ axis, the following single matrix can be obtained by the orderly multiplication of $T_x$, $T_y$ and $T_z$.

\[
\begin{bmatrix}
\cos(\theta_x) & -\sin(\theta_x) & 0 \\
\sin(\theta_x) & \cos(\theta_x) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
T_{\text{rotational}} = T_x \ T_y \ T_z
\]

and

\[
[x' \ y' \ z'] = [x \ y \ z] \ T_{\text{rotational}} \quad \text{---------} \quad 4.9
\]

The components of the rotational matrix $a_{i,j}$ are given by the following.

\[
a_{1,1} = \cos(\theta_y) \ \cos(\theta_z)
\]
\[
a_{1,2} = -\cos(\theta_y) \ \sin(\theta_z)
\]
\[
a_{1,3} = \sin(\theta_y)
\]
\[
a_{2,1} = [\sin(\theta_x) \ \sin(\theta_y) \ \cos(\theta_z)] + [\cos(\theta_x) \ \sin(\theta_z)]
\]
\[
a_{2,2} = -[\sin(\theta_x) \ \sin(\theta_y) \ \sin(\theta_z)] + [\cos(\theta_x) \ \cos(\theta_z)]
\]
\[
a_{2,3} = -\sin(\theta_x) \ \cos(\theta_y)
\]
\[ a_{3,1} = -[\cos(\theta_x) \sin(\theta_y) \cos(\theta_z)] + \\
[\sin(\theta_x) \sin(\theta_z)] \]
\[ a_{3,2} = [\cos(\theta_x) \sin(\theta_y) \sin(\theta_z)] + \\
[\sin(\theta_x) \cos(\theta_z)] \]
\[ a_{3,3} = \cos(\theta_x) \cos(\theta_y) \]

The trigonometric functions of \( \theta_x \), \( \theta_y \) and \( \theta_z \) need to be computed only once per rotation and can be done prior to evaluating equation 4.9. This will facilitate the speed of the rotational transformation.
5. THE I.B.M. PERSONAL COMPUTER

This chapter provides a brief introduction to the I.B.M. PC's programming environment. This material will facilitate the understanding of subsequent chapters on program development. Sections 5.1.2 and 5.2.1 should be of particular interest since these provide information regarding the programming tools I.B.M. has provided for software development. The contents of this chapter is based on reference 20, 22, 23 and 24.

5.1 The hardware

The microprocessor that is used in the PC is the Intel 8088 which is a 8/16 chip, i.e. internally it can manipulate 16 bits but all external data communication is done 8 bits at a time. The 20 bit address bus permits the PC to directly address 1024K of RAM. 64K of this memory is resident in the PC's system board. It also has 5 expansion slots which is supported through a 62 bit I/O channel. Although the 8087 co-processor does not come with the computer, there is a readily available socket for it, provided by the side of the 8088. To a programmer, three other chips within the PC are of importance. They are the 8259A interrupt controller which is used to supervise the operation of interrupts, NEC PD765 floppy disk controller and the Motorola 6845 CRT controller chip. It is possible to issue commands directly to the latter two chips, and in particular, a knowledge of programming the 6845 is very useful.
5.1.1 The system memory map and the addressing scheme

The system memory map is presented in figure 14. As evident from this diagram, the first 1024 bytes of memory is reserved for the interrupt vectors [it is important to understand the use of these interrupts in order to gain access to the system resources (also see section 5.1.2 & 5.2.1)]. There are 256 interrupt vectors stored in this 1k of lower end memory. Each of these occupies 4 bytes, 2 for the segment address and 2 for the offset address (see section 5.1.3), pointing to the associated interrupt service routines. Each of these vectors is assigned an interrupt number (0,...,FFH). Referring to figure 14, the important interrupt numbers to a programmer are as follows.

[a] BIOS (Basic Input Output System) entry points (10H through 1FH)
[b] DOS interrupts (20H through 3FH)
[c] User interrupts (40H through 7FH)

A summary of the BIOS and the more useful DOS interrupts (also see section 5.2.1), types are given in appendix C.

5.1.2 The I.B.M. BIOS ROM

The I.B.M. BIOS ROM resident routines provides an excellent set of services which are accessible to the programmer through the above mentioned interrupt set [a]. A complete listing of these routines is given in appendix A, of reference 20. Out of these, the most useful routines are those allowing video control. The interrupt type 10H provides these services and is given in appendix C of this thesis.
<table>
<thead>
<tr>
<th>Segment</th>
<th>Description</th>
<th>Base Address</th>
<th>Size</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>16K - 64K ON BOARD MEMORY</td>
<td>00000</td>
<td>00080</td>
<td>BIOS INTERRUPT VECTORS (0H-1FH)</td>
</tr>
<tr>
<td>10000</td>
<td>192K MEMORY EXPANSION AREA</td>
<td>00080</td>
<td>00100</td>
<td>DOS INTERRUPT VECTORS (20H-3FH)</td>
</tr>
<tr>
<td>40000</td>
<td>384K MEMORY FUTURE EXPANSION AREA</td>
<td>00100</td>
<td>00200</td>
<td>USER INTERRUPT VECTORS (40H-7FH)</td>
</tr>
<tr>
<td>A0000</td>
<td>16K RESERVED</td>
<td>00200</td>
<td>00400</td>
<td>BASIC INTERRUPT (80H-FFH)</td>
</tr>
<tr>
<td>A4000</td>
<td>112K GRAPHICS/VIDEO DISPLAY BUFFER</td>
<td>00400</td>
<td>00500</td>
<td>BIOS DATA AREA</td>
</tr>
<tr>
<td>C0000</td>
<td>192K MEMORY EXPANSION AREA</td>
<td>00500</td>
<td>00600</td>
<td>BASIC &amp; DOS DATA AREA</td>
</tr>
<tr>
<td>F0000</td>
<td>16K RESERVED</td>
<td>00600</td>
<td>062.5K</td>
<td>62.5K USER MEMORY</td>
</tr>
<tr>
<td>F4000</td>
<td>8K USER AREA</td>
<td></td>
<td></td>
<td>R/W USER MEMORY</td>
</tr>
<tr>
<td>F6000</td>
<td>32K CASSETTE BASIC ROM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FE000</td>
<td>8K BIOS ROM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 14. The system memory map
5.1.3 Memory addressing scheme in the 8088

The 8088 is capable of accessing $2^{20} = 1024K$ (1 megabyte) of memory. Therefore it needs to address a range of memory locations 0 through FFFFFH. However, the 8088 is a 16 bit machine, consequently the largest address it can evaluate within it is $2^{16}-1 = 65,535$. (i.e. 65536 locations which is 64K). Thus, in order to provide access to a total of 1024K, the 8088 utilizes a sophisticated memory addressing scheme that involves memory segments. This is done as follows. As pointed out earlier, a 16 bit register can address 64K locations. Now, if we define a 16-byte allocation of memory to be a paragraph, then a single 16-bit register can be utilized to point to 64K paragraphs. Using this scheme allows the 8088 to address 1024K ($64 \times 16$) of memory space. The division of the memory space into paragraphs is shown in figure 15 (22).

Note that since each segment is capable of addressing 16-bytes allocations, a segment address always begins at a paragraph boundary. Consequently the least significant digit of these addresses will always be zero. The notation used in specifying the address is as follows.

Segment-Add : Offset-Add

The implementation of this addressing scheme in the 8088 is carried out through the use of segment registers and an offset register. There are four segment registers, the code segment (CS), stack segment (SS), data segment (DS) and the extra segment (ES) register. The offset address is always in the instruction pointer (IP).
Figure 15. The division of memory into paragraphs (22)

Now then, the IP is a 16-bit register, so it too can address a range of 64K addresses. Therefore, once a segment register is set to a particular paragraph boundary, a 64K address space pivoted about that paragraph boundary, can be reached by manipulating the IP. Given the segment:offset address pair, it is easy to evaluate the corresponding absolute address. For example, consider the segment:offset address pair 2915:0100. Since the segment can address only paragraph boundaries, the first manipulation is to multiply the segment addresses by 10H. Then addition (or the subtraction) of the offset address from it gives the absolute address.
5.1.4 The color video buffer

The ROM-BIOS supports essentially two graphics modes, a 320 by 200 pixels mid-resolution mode and a 640 by 200 pixel high-resolution mode. The latter is used for all graphics display in this work. The I.B.M. PC uses memory mapped graphics. In the high-resolution mode, each bit has a corresponding location in memory and the status of this location determines whether the pixel is on (1) or off (0). Therefore, a byte of memory contains enough information to determine the state of eight adjacent pixels. Since each pixel is represented by only one bit, it is not possible to have color in this mode and there is no means of controlling the contrast (thus shading is not possible). The complete image will be represented by one color. In this mode, the complete screen requires a total of 640 by 200 bits in memory. After rounding up to an even binary, this means 16K of memory is required to support this color graphic screen. However, referring to figure 14, there is a total of 112K allocated for the video buffer, starting at A4000H. The general feeling is that I.B.M. intends to produce a display with much higher resolution and the extra memory allocated here is intent for such a display. The high-resolution graphics mode memory map is given in figure 16 (22). Referring to this figure, the graphics memory is laid out in two banks of 80-byte lines, 100 lines per bank. The even number lines begin at offset 0 and the odd numbered lines at offset 2000H. The
segment address is B800H. Each byte is displayed with the most significant dot on the top line of the high-resolution screen, bit 6 to its right and so forth.

<table>
<thead>
<tr>
<th>Offset (hex)</th>
<th>Line (dec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0050</td>
<td>2</td>
</tr>
<tr>
<td>00A0</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1EF0</td>
<td>198</td>
</tr>
</tbody>
</table>

Odd-Line bank

<table>
<thead>
<tr>
<th>Offset (hex)</th>
<th>Line (dec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
</tr>
<tr>
<td>2050</td>
<td>3</td>
</tr>
<tr>
<td>20A0</td>
<td>5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3EF0</td>
<td>199</td>
</tr>
</tbody>
</table>

Figure 16. The high-resolution graphics memory map (23)
An understanding of the video memory map and the 8088 addressing scheme (section 5.1.3) is essential for any direct data transfer from memory to the video buffer. This will be required for the implementation of an image space hidden line removal algorithm.

The most annoying aspect of the I.B.M. color graphics monitor is the difference between the resolution in X & Y directions. A convenient approach to follow is to regard the display as a logical screen with equal resolution in both axis. Then, most of the work can be carried out independent of the display device. Finally, the mapping from the logical to physical space can be carried out during the viewport transformation.

5.2 Software considerations

5.2.1 MS DOS (version 2.1)

A primary function of an operating system is to provide the user a convenient method for accessing the computer resources. In most cases, this is performed through various system level software routines. Methods of accessing these routines vary from system to system. In MS DOS, the method of access is through the execution of interrupts (see section 5.1.1). Referring to figure 17, the most useful DOS interrupt type is 21H. This interrupt vector provides access to a variety of functions that can be very useful in a software development environment. A description of these functions is given in appendix C.
5.2.2 Interfacing assembly language routines with Turbo Pascal source code.

There are certain situations where the use of assembly language programming is warranted. One is where the execution speed of the object code is of main concern and another is the requirement of a method to directly access the ROM-BIOS services. Once such a routine is developed, then it must be properly interfaced with the high level language source code. The following example illustrates this process with Turbo Pascal as the high level language.

Example: Scan Codes.

A majority of the I.B.M. PC’s keyboard produces an ASCII code whenever a key is pressed. However, a certain group of keys do not have an ASCII representation. One such group is the set of function keys numbered F1 - F10. On the other hand, all the keys on the keyboard (except shift key, ctrl, caps lock and alt) has a "Scan Code" associated with them. Consequently whenever a key is pressed two codes are trans-
ferred back to the system. In register A, "AL" has the ASCII code and "AH" has the Scan code. For keys that have only scan codes, AL=0. As an example, the set of scan codes obtained for function keys F1 - F10 is given below.

<table>
<thead>
<tr>
<th>F1</th>
<th>3B</th>
<th>F6</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>3C</td>
<td>F7</td>
<td>41</td>
</tr>
<tr>
<td>F3</td>
<td>3D</td>
<td>F8</td>
<td>42</td>
</tr>
<tr>
<td>F4</td>
<td>3E</td>
<td>F9</td>
<td>43</td>
</tr>
<tr>
<td>F5</td>
<td>3F</td>
<td>F10</td>
<td>44</td>
</tr>
</tbody>
</table>

Therefore, whenever it is required to program the function keys to perform a certain task, the scan codes must be obtained by the program. The most convenient way is to write an assembly language routine to perform this function (see section 7.5). Let us now examine the details of interfacing Turbo Pascal source code with an assembled, assembly language routine.

In Turbo Pascal, an assembly language routine is known as an "external". Although MS DOS allows two types of executable files, EXE and COM, Turbo Pascal restricts the use of object code files to COM files. Consequently, any assembly language routine must be fully contained within one segment.

Now, let us examine the Turbo-external interface in details. Two critical functions are included within this interface. One is the passing of control to and from, and the other is the passing of parameters to and from the external routine. These are discussed in detail below.
[a] Passing control to and from an external

On calling an external routine, Turbo pushes the return address onto the stack. This address contains only the offset (1 word), since Turbo expects the external to be a COM file. Further, the registers BP, CS, DS and SS must be saved and restored by the external routine. The external returns control by executing a "near" RET (since only the offset was pushed onto the stack).

[b] Passing parameters to and from an external

All parameters passing is performed using the 8088 stack. A section of this stack is shown in figure 18. Several features peculiar to this stack are as follows.

(i) It grows from high to low addresses

(ii) Pop and push operations always work on 2 bytes

(iii) The stack pointer (SP) always points to the top most item on the stack. (NOT to the next available memory address).

With variable (var) parameters, the address (segment : offset) of the first byte of the parameter is pushed on the stack. With value parameters, the data transferred on the stack corresponds to the internal
data representation of Turbo Pascal (see page 223 in reference 24). The external routine must use the base pointer (BP) register to reference the parameters on the stack. Prior to this, the entry value of BP must be saved using the following instructions.

```
PUSH BP ; save entry value
MOV BP,SP ; transfer stack pointer
```

Now, BP can be used within the external. On exit the following must be done to restore this register.

```
MOV SP, BP ; restore stack pointer
POP BP ; restore base pointer
```

The listing 2 given in appendix B illustrate the use of an external function to read scan codes.

```
Low addresses  < Return address > <--- SP
<            >
<            >   } parameters
<            >
High addresses  <
```

Figure 19. Stack on entering an external (24)
6. THE PROGRAM STRUCTURE

This chapter introduces the reader to the structure of "Micro Stream". Subsequent chapters will concentrate on individual components and present details of their design. The material presented in this and the following chapters is expected to complement the program listings provided in appendix B of this thesis. The running of the package is described in appendix A.

The complete package is designed to run in 256K of user RAM. Allowing 50K for code files leaves a maximum of 206K for design files. Of the 50K, approximately 41K is available for code files, since, during compilation, any Turbo Pascal program inherits 9K of Pascal run-time library routines. The required run-time environment specifications are summarized in section 6.1.

Now, the code files resulting from the total package would far exceed the 41K limit set above. Consequently the package is developed as separate programs and linked together through "program chaining" (see section 6.2). This approach also facilitates future extensions required to complete "Micro Stream & Micro Cutter". "Micro Cutter", a microcomputer version of "Cutter" (see section 1.1 and reference 9), can be developed as a separate program and called through the main menu of the package. It can even be developed in a different computing language and be linked through the "Micro Stream" main menu. Further, this approach allows foreign programs to utilize the "Graphics" routines, provided the interface requirements are fulfilled.
6.1 Run time environment specifications.

. I.B.M. PC with 256K of user RAM, or a true compatible (particularly ROM compatibility is important). Two tried examples are the AT&T 6300 and the ITT Xtra.
. I.B.M. Graphics card and the color monitor
. Two disk drives. "Micro Stream" system disk is to be in drive A and the data file disk in drive B.

6.2 The program structure and chaining

Figure 20 presents the interaction of all the different programs that constitutes "Micro-Stream".

Autoexec.BAT

```
|-------- Title.COM
|----- Main.BAT
|----- Strt_Up.COM
|----- Batch.BAT
|-------- (stream.COM)
|-------- (Grphics.COM)
|-------- (others)
```

Figure 20. Program chaining in "Micro Stream"

First a word about batch (BAT) files in MS DOS. For these who are familiar with Digital VMS environment, the BAT files are analogous to the VMS "command" files. They allow the programmer to group and execute DOS commands. In figure 20 above, "Autoexec", "Main" and "Batch" are all
BAT files. "Autoexec" is a facility provided by MS DOS to automatically execute programs. On system start up, after the bootstrap process, DOS searches the default disk for an "Autoexec.BAT" file. If found, it immediately begins execution of this file. This allows the programmer to customize the PC environment to his personal needs. Now then, referring to figure 20, "Autoexec" calls a Pascal code file "Title" which display an opening message. Then it calls another batch file "Main.BAT". This implies that MS DOS allows chaining of batch files. However, BAT files cannot be nested, i.e. after executing the called file, control cannot be returned to the original batch file. Nevertheless, Pascal code files can be nested within batch files. Further, "Autoexec" cannot be called by other batch files.

Once called, "Main.Bat" assumes total control of "Micro Stream". It calls a Pascal code file "Strt_Up", which displays the main menu shown in figure 21.

![Main menu of "Micro Stream and Micro Cutter"](image)}
Once the user has made a choice from this menu, a mechanism must be available for the current Pascal program to call another Pascal program (i.e. to load it from disk and transfer control to it). Turbo Pascal provides a procedure "execute" [see page 193 of reference(24)], which can be used to activate other Turbo Pascal programs. However, this procedure does not alter the memory allocation state, i.e. the segment address and sizes of the code, data and stack segments are not changed. Consequently it is not the most appropriate program chaining mechanism for this package. Instead, in this package, the programs are chained using MS DOS batch file facilities.

Returning to figure 20, "Strt_Up" creates a new file "Batch.BAT" on disk A, which contains the DOS commands for executing the user requested program. Now, since "Strt_Up" is a COM file, on completion of its execution, control passes back to the original BAT file "Main" which activates the newly created BAT file "Batch". This ensures the execution of the users choice from the main menu. However, since BAT files cannot be nested, on completion of the execution of "Batch", it needs to call "Main" to complete the loop. The mechanism for terminating the loop (<esc>) is provided in "Strt_Up".

It must be mentioned that other systems such as the UCSD P-system provides a more convenient mechanism for allowing an user program to activate other programs on disk, through I/O redirection and by the use of the operating systems command line buffer.
6.3 Memory management and data structures

When designing software requiring the use of various lists during execution, the limitation of available RAM must be given careful thought. Proper management of this memory space can significantly contribute towards the success of such a package. In a CAD package, the amount of memory required for each design cannot be predetermined. However, it is desirable to have an arrangement where the design is not seriously inhibited due to limitations in the design of the software.

Two types of variables can be utilized for representing a list, one being the array and the other, the pointer variable. For this project, the use of pointer variables have two major advantages over the array representation.

[a] Being a dynamic variable, no fixed specification is required for size of the lists at compilation. Instead, the lists can be managed at execution time. Thus unwanted lists can be discarded to make room for new ones. This clearly maximizes the efficient use of memory, specially in a limited RAM environment.

[b] At compilation Turbo Pascal allocates a segment of 64K as the global data area. Clearly, this space is NOT adequate if global arrays were used as the list for this project. On the other hand, the size of the heap (the dynamic memory structure utilized by the compiler to manage pointer variables) is only limited by the size of the available memory in the computer. However, due to the approach followed by Turbo Pascal, a single pointer variable is limited to a maximum size of 64K (i.e. confined to one segment), but several variables, each of 64K can of
course co-exist on the heap.

It is important to notice an overhead involved with the use of pointer variables. Each node of the list requires at least one pointer variable which occupies 4 bytes of memory. However, this overhead memory requirement becomes insignificant as the size of the record represented by the pointer variable increases.

All significant list structures in this package are implemented as linked lists using dynamic pointer variables. Details of these data structures are discussed in individual chapters that are devoted to describing separate programs of "Micro Stream".

6.4 "Micro Stream" system & program disk specification

Following files must be resident on the system disk. The numbers in square brackets indicates the size of the files in bytes.

. Autoexec.BAT   [ 59]   . Title.COM   [11758]

Further, for "Autoexec.BAT" to be activated after the bootstrap process, DOS must also be placed on this disk. Since it is desirable to obtain graphics screen dumps, the COM file "Graphics.COM" resident on the original MS DOS disk must also be placed on "Micro Stream" system disk. It is not possible to include these features during development since Microsoft corporation owns the rights to DOS. A listing of all the source files resident on the "Micro-Stream" program disk is as follows.
. Strm_Utl.Pas [12019]
. Strm_Ut2.Pas [2990]
. Strm_typ.Pas [704]
7. **UTILITY PROGRAMS**

This chapter describes various utility programs that were developed to facilitate I/O and screen operations. These are grouped into two "include" files, "Strm_Utl.Pas" in "Stream" and "Gra-Utl.Pas" in "Grphics". In the following discussion, a routine that belongs to a particular program (i.e. to "Stream" or Grphics") will have the corresponding program name included in the section heading. Likewise, a heading without the associated program name indicates a common routine to both programs. Please refer to the source listings in appendix B during the following discussions.

7.1 **Clr_Scr_Gra**

During viewport operations in the graphics mode, it is useful to have a means for selectivity clearing the screen. Given the viewport coordinates, this routine performs that function by executing a BIOS-ROM routine through INT 10H. The following register allocation is required.

\[
\begin{align*}
(AH) &= 6 \\
(AL) &= \text{number of lines to be blanked. 0 means blank entire viewport.} \\
(CH,CL) &= \text{Row, column of lower right corner of viewpoint} \\
(BH) &= \text{Attribute to be used on blank line. 0 in graphic mode gives background.}
\end{align*}
\]
7.2 **Strg_Read**

The standard input procedures "read" and "readln" supported by Pascal is not very suitable due to the limited editing permitted on the input string, during input. In order to provide a very user friendly environment, the input driver "Strg_Read" is implemented. The following pseudo-code presents the basic technique utilized in this procedure.

```
Repeat
  Repeat
    read (single-character)
  until character in valid-set

  case character of
  <back space> : if string(length) <> 0 then
    back-space.
    else
      ignore request;
  <return> : if legal then
    Exit
    else
      ignore request;
  else {any other character entered
    is not an edit character}
    if string (length) < maximum allowable then
      add character to string
    else
      ignore entry since no space in buffer;

until Exit.
```

Fundamental to the technique are the two variables "string" and "valid-set". Each entry from the keyboards is first checked against the set variable "valid-set". If the entry is a component of the set then it is accepted as a legal entry. Then, if the entry is an edit command, the appropriate command is carried out provided it is a legal operation.
on the current "string" (for example, it is not possible to backspace on an empty string). If the entry is not an edit command, then providing space is available, it is placed in the "string" buffer.

This process is repeated until a terminator <return> is entered. Then, the entry is passed out to the caller as a character variable. The interpretation of this set of characters is left to the caller. The excellent string operations provided by Turbo Pascal enables easy conversion of correct string data into numerical values (floating point and integers). Some typical examples are as follows.

My-file : No conversion is required by the caller since this is expected to be a file name.

-23.502 : A floating point number entered as a character string. The caller converts this to a numerical value through the standard Turbo procedure "Val" (see page 70 of reference 24)

50 : A integer value entered as a character string. Conversion by the caller as above.

It is important to notice the great flexibility of this input driver due to the use of the set variable "valid-Set". The programmer can include any valid base type in this variable prior to calling "strg-read". Also within "strg read" it is possible to perform subtraction and additions on this set variable. For example, when entering a negative value, the minus sign has to be the very first character entered. If not, the number must be positive. Therefore on receiving
the first entry, "strg-read" will remove the minus sign (if it is in "valid-set") from "valid-set" thus further entries of minus signs are not permitted. A similar course of action is followed for the decimal point. Also another useful function is the correct formatting of numerical data. For example, if only 3 digits are allowed after the decimal point, then it is possible to force the decimal point into the "string" at the appropriate place, thus insuring that the entry conforms to the format requirement. Clearly, this procedure can be used to eliminate invalid data entry by the user (totally transparent to the user), thus providing a very friendly input environment.

7.3 Frame [Stream.Pas]

The I.B.M. PC's extended character set provides many graphic characters that can be used in the text mode for drawing purposes. This routine uses the following characters to draw a double line box frame of any size, in the text mode.

BAH, DDH, D8H, D9H, BCH, BBH

7.4 Box_In_Gra

In the graphics mode, the characters are formed from a character generator image maintained in the system ROM. Only the first 128 characters are maintained there. Therefore the graphics characters usable in text mode (section 7.3) is no longer directly available in the graphic mode. Consequently Box_In_Gra is utilized to draw a frame of any size, in high resolution graphics.
7.5 Reading scan codes

Turbo Pascal does not provide a convenient method for reading the scan codes from the keyboard (see section 5.2.2). Therefore an "external" function was written in 8088 assembly language for this purpose. It is implemented by calling a BIOS-ROM routine through the execution of int 16H. The settings for the registers are as follows.

\( (AH) = 0 \) Read next ASCII character structure from keyboard
\( (AL) = \) Has the ASCII code on return
\( (AH) = \) Has the scan code on return
8. "Stream" SOFTWARE DESIGN

This chapter presents the software details of the design package "Stream". The discussion concentrates on the design and the data structures of this program. First, the outline of the complete package is given, followed by a description of individual modules. Appendix B contains the source code of the "B_Spline" routine.

8.1 Modules of "Stream"

Stream

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Strm_typ.Pas</td>
<td>I</td>
</tr>
<tr>
<td>Strm_Utl.Pas</td>
<td>I</td>
</tr>
<tr>
<td>Init_Screen</td>
<td></td>
</tr>
<tr>
<td>Strm_Inpt</td>
<td></td>
</tr>
<tr>
<td>Strm_cal.Pas</td>
<td>I</td>
</tr>
</tbody>
</table>

Figure 22. Modules of "Stream"

Referring to the above figure, "Stream" calls a number of different modules to perform its functions. Some of these are Pascal procedures and are characterized by the lack of the "Pas" extension in the above figure. Others are separate Pascal programs resident in different disk files. These have the extension "Pas", and also the letter "I" indicates that they are included in "Stream" during compilation. The use of this "include" facility helps to compensate for some of the disadvantages brought about by the lack of a "linker" in Turbo program development environment. A description of the individual modules are given
8.1.1 Strm_typ.Pas

The following global data type declarations are in this file.

```pascal
Line_Displacement = String [80]  -----[a]
Str_1 = String [16]  -----[b]
Characters = Set of char  -----[c]

Ptr = ` Node  -----[d]
Node = record
  Num, Mode : byte
  X,Y,I,
  Billet_X, Billet_Y : real;
  F_link, B_link : Ptr
end;

Dsg_Ptr = ` Dsg_node  -----[e]
Dsg_node = record
  X,Y,Z : real;
  F_link : Dsg_Ptr
end;
```

Of these, [a] and [b] are required since structured types cannot be used in procedure headings. [c] is used in "Strg_Read" input utility program (see section 7.2). The use of types [d] and [e] are further explained in section 8.2 on data structures.

8.1.2 Strm_Util.Pas
The contents of this program was discussed in sections 7.1 - 7.5.

8.1.3 Init_Screen

This routine sets up the initial screen for "stream", defines the window and draws the frame.

8.1.4 Strm_Inpt

All "Stream" input is obtained through this procedure. Its module structure is given in figure 23.

```
Strm_Inpt
|---------- Strm_Pgl.Pas I
|---------- Strm_Pg2.Pas I
|---------- Strm_Pg3.Pas
```

Figure 23. Modules of "Strm_Inpt"

Referring to the above figure, "Stream" acquires its input through three pages. Each of these page programs are implemented as an "include" file. They are described in the following sections.

[a] Strm_Pgl.Pas

The product file name is obtained from the user through this routine. It uses the "Strg_Read" utility procedure for this purpose. Once a valid name is obtained (this is assured by "Strg_Read"), it checks drive B disk directory for the file. If found, it reads the product data and forms a "doubly linked" link list of it (see section 8.2 on data
structures). Otherwise it assumes a new file and returns a "nil" product list.

[b] Strm_Pg2.Pas

The details on electrode geometry is obtained through this program. Its module structure is given in figure 24.

Page_2

Figure 24. Modules of "Strm_Pg2.Pas"

(i) The input mode

If the current file is new, then "page-2" automatically activates its input mode where the user is allowed to enter fresh data into the menu. If it is an old file, the data resident in the product definition list is used to fill the data menu. Once complete, it enters an edit mode allowing the user to change any of the data entered.

(ii) The edit mode

In this mode, the user is allowed to change any of the information currently on the menu. Within this screen editor, the movement of the cursor is implemented by "Carriage_Rtn" and "replace" is utilized to enter new information on the menu.

[c] Strm_Pg3.pas
This input page differs from the other two, since it works in the high resolution graphics mode. The product definition is entered through this page. Its module structure is given in figure 25. The screen is divided into two viewports, one is used to display the product and the other for the definition of the geometry. The latter has two modes, an input and an edit mode. These two are analogous to those described in (b) above. Note that if the current file is an old file, then the program displays the old data and the product, then automatically enters the edit mode. The geometry definition viewport is divided into pages since it is not possible to know, how many nodes will be required to define a product. Each page accommodates 7 nodes. In the input mode, pages are automatically incremented so that at the end of defining the 7th node, the user is presented with a fresh page. In edit mode, the user can go back and forward through the pages using the function keys F1 and F2. This is implemented through the procedure "Pre_or_Next" which utilizes the external "Read_Key". Within each page, the screen editor allows the user to move the cursor to any entry and replace old data with a new entry (through Replace & Carriage_Rtn). Any modification to the geometry definition can be reflected in the display simply through the use of "D" key. This function is implemented through "Disp_Product".
All the calculations pertaining to "Stream" are performed within this program. The module structure of it is shown in figure 26.

Of these routines, Centroid, Map-Cal, Die_Volume and S_Area (surface area) perform calculations related to the die cavity design and analysis. The concepts used in these routines were first presented
in the original version of "Stream" (25). "Disp_Map" is used to produce the graphical output of the "billet to product" material map.

The routine \texttt{Wrt\_Map\_file}, writes a disk file containing all the product nodes and their corresponding billet nodes. (see section 8.3).

"De Boor" algorithm presented in section 3.3 is implemented in the procedure \texttt{B-spline}. It must be mentioned that in his Ph.D. thesis, Riesenfeld (15) presents this algorithm in a slightly different manner. However, the author feels that the original presentation of "De Boor" (see section 3.3) is more suitable for implementation and is used in this work.

\begin{verbatim}
B_Spline  
|--------- Streamline  
|--------- Knots
\end{verbatim}

Figure 27. Modules of B_Spline procedure

The procedure \texttt{B_Spline} calls "Streamline" to determine the defining open polygon, total number of vertices and the order of the B-Spline. Therefore it is possible to implement any type of curve merely by adding new polygon vertex definitions to "streamline". Following this, it calls the procedure "knots" which allocates the knot vector. Multiple vertices are accommodated during this process. Then B-Spline executes the "De Boor" algorithm. The following arrays are utilized as data structures.

\begin{verbatim}
X : Array [0.. 17] of real \{knot vector\}  
V-x, V-y, V-z : array [0..10] of real \{x,y,z coordinates of the defining polygon vertices\}
\end{verbatim}
A, DP, DM : array [1..7] of real \{see equation 3.5\}

Notice that the "A" is a single dimension array and needs only to be large as the highest permissible order of the B-Spline. This is possible because the algorithm allows preceding values of A \((r,s)\) (equation 3.5) to be overwritten by succeeding values. Therefore the memory requirement for implementing this algorithm is very small. Also, by redimensioning these arrays, higher order B-Splines with more vertices can be easily accommodated. This is possible since the algorithm, itself does not impose any restrictions (for numerical stability \(k \leq 20\)).

8.2 Global data structures in "Stream"

This section describes the use of the two types [d] and [e] presented in section 8.1.1. Two global linked list structures are utilized to represent related lists.

(a) A doubly linked circular list [of type (d) in section 8.1.1)] is utilized to store the product geometry information. Several features of this link list are as follows.

. Being based on pointer variables, the number of nodes on the product need not be known at compilation.

. The topology of the product is automatically taken care of by the linked list structure. Consequently the list must be circular.

. The backward link is required for implementing the "paging" feature in "Strm_Pg3.Pas" [see section 8.1.4(c)].
This data structure is present in figure 28.

(b) A forward linked circular list [of type (e) in section 8.1.1)] for storing the complete electrode definition, i.e. the coordinates of each node. Most of the comments given in (a) above are applicable here too. This list is depicted below.
8.3 The use of data files in "Stream"

Two data files are generated by this program. They are both written onto the B disk.

[a] <file-Name> . PRO

This file contains all the users defined product and electrode geometry data. It is read by "Strm_Pgl.Pas" program [see section 8.1.4(a)]. Writing this file is done by the procedure Strm_Inpt. A segment of this file with an explanation of its contents are given below. <file_Name> is any user entered text string and the extension "PRO" is automatically appended by the program.

```
electrode length  electrode diameter  no. of sections  stream line
4.0000  4.0000  5  5
1  0.0000  0.0000  0.0000
2 -0.5000  0.0000  0.0000
3 -0.5000 -1.0000  0.0000
4  0.5000 -1.0000  0.0000
5  0.5000  0.0000  0.0000
Node x y radius
```

Figure 30. The data file <file_Name>.Pro

[b] <file_Name>.MAP

This file serves as the interface between "Stream" and the "Graphics" programs. It is written by the procedure "Wrt_Map_file" in "Strm_cal.Pas" program (see section 8.1.5) and contains the electrode geometry data entered by the user and the x,y,z coordinates of the nodes
resulting from the mapping calculations (i.e. product and the corresponding billet nodes). Here, the <file_name> is the user entered product file name and the extension "MAP" is automatically appended by the procedure. A segment of this data file with an explanation of its contents is given below.

<table>
<thead>
<tr>
<th>electrode length</th>
<th>no. of section</th>
<th>stream line</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0000</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.5000</td>
<td>0.0002</td>
</tr>
<tr>
<td>-0.1000</td>
<td>0.5000</td>
<td>-0.3127</td>
</tr>
<tr>
<td>-0.2000</td>
<td>0.5000</td>
<td>-0.6179</td>
</tr>
<tr>
<td>-0.3000</td>
<td>0.5000</td>
<td>-0.9079</td>
</tr>
<tr>
<td>-0.4000</td>
<td>0.5000</td>
<td>-1.1755</td>
</tr>
<tr>
<td>-0.5000</td>
<td>0.5000</td>
<td>-1.4142</td>
</tr>
</tbody>
</table>

Product x Product y Billet x Billet y

Figure 31. The data file <file_name>.MAP
9."Graphics" SOFTWARE DESIGN

This chapter follows the approach taken in chapter 8, to present the "Graphics.Pas" program. The source code is given in appendix C.

9.1 Modules of "Graphics"

Graphics

Figure 32. Modules of "Graphics"

As evident from the above figure, "Graphics" program calls four other Pascal programs (through the include facility), which are resident on in separate disk files. They are described below.

9.1.1 Gra_Typ.Pas

This include program contains all the global data structures utilized in "Graphics".

Line_Disp = String [80]       ------[a]
Str_1    = String [16]         ------[b]
Characters = Set of char       ------[c]
Dsg_Ptr  = ^ Dsg_Node;         ------[d]
Dsg_Node = Record

    x, y, z : real;
F_link : Dsg_Ptr
end.

Dsp_Ptr = ^ Dsp_Node; -------[e]
Dsp_Node = Record
    x, y : integer;
    z : real;
    F_link : Dsp_Ptr
end

Ptr_1 = ^ File_node -------[f]
File_node = record
    Pro_x, Pro_y,
    Billet_x, Billet_y : Real;
    F_link : Ptr_1
end

Of these, the use of [a], [b] and [c] are as explained in section 8.1.1. The use of types [d], [e] and [f] are described in section 9.2 on "Grphics" data structures.

9.1.2 Gra_Util.Pas

The contents of this program was discussed in chapter 7.
9.1.3 Gra_Uti2.Pas.

Draw-Main-Proc.

```
--------  Read_file
--------  B-Spline
--------  World_to_Eye
--------  Eye_to_Scrn
--------  Map_to_Viewport
--------  Display
```

Figure 33. Modules of "Gra_Uti2.Pas"

The perspective transformation described in section 4.2.4 is implemented through this program. The flow control of this program is illustrated in figure 34. In this figure, the procedure names performing associated functions appear within square brackets.
Figure 34. Flow chart for "Draw_Main_Proc"
Referring to figure 34, if the file is resident on disk B, then the data in the .MAP file is read into the Dsg_list (see section 9.2), else the Dsg_list is returned as a "nil" list. If the latter was true, then the rest of the action takes is clear from the flow chart. However, if a new file was read, then B-splines for the design is evaluated. Following this, the perspective transformation is carried out. This is done by the series of routines listed below.

World_to_Eye : Transformation described in section 4.2.3
Eye_To_Scrn : The transformation described by equation 4.8 in section 4.2.4
Map_To_Viewport : The viewport transformation described in section 4.1
Display : The final display on the screen is done by this procedure.

Notice that the transformations carried out in the first two of these procedures are independent of the display device, i.e. the resolution of the I.B.M. display devices is not considered and the screen is treated as a logical device with equal resolution in both X and Y. The mapping from the logical screen to the physical screen is carried out by the latter two procedures. Consequently these are display device dependant. A pixel density factor of 2.1 (applicable to the I.B.M. color monitor operating in the high resolution mode) is applied as required to the viewport definition.

Gra_Uti4.Pas

--- Split-Screen
--- Full-Screen
--- Change-screen
--- Rotate

Figure 35. Modules of "Gra_Uti4.Pas"

The "Grphics" program implements two screen modes, full screen and the split screen. These functions are implemented by the first three routines listed in figure 35 above. The "Rotate" procedure implements the transformation described in section 4.3.

9.2 Global data structures in "Grphics"

The following three forward link list structures are utilized in this program.

[a] Dsg_list : Holds the data read from the MAP file.
[b] World_list : Hold the coordinates of all the nodes resulting from fitting the B-splines.
[c] Disp_List : The image space object data is held in this circular list.

Of these, [b] and [c] are global and [a] is local to Draw_Main_Proc. Their structure is analogues to that shown in figure 29. The non circular lists [a] and [b] above are terminated by 'nil' pointers.
10. MISCELLANEOUS CONSIDERATIONS

10.1 Turbo Pascal [version 3.0]

[a] Advantages

Turbo Pascal offers many excellent features which are not in standard Pascal. This is particularly true, when it operates in the MS DOS environment. These features are summarized below.

- String manipulation procedures and functions.
- Graphics operations including turtle graphics. However, the circle drawing routine in the extended graphics package does not draw a perfect circle in the high resolution graphics mode.
- Color options both in text and graphics modes.
- Extended file handling capabilities including the provision of a series of logical devices predefined as files.
- I/O redirection compatible with MS DOS 2.1.
- "Mark" and "release" procedures for managing the heap which are faster than the standard "dispose" procedure.
- Extended set of functions providing low level access to system hardware.

[b] Disadvantages

- The lack of a linker facility. Consequently files with relocatable information (with the file extension EXE) can not be accommodated by Turbo programs.
Code and data segments are limited to a maximum of 64K segments. This can be overcome through program chaining, although this will result in a decreased program execution speed due to the overhead of reloading programs whenever they are required to be executed.

Due to the implementation procedures followed by Turbo Pascal, a single pointer (dynamic) variable is limited to 64K. This is a serious limitation particularly in a link list implementation utilizing dynamic variables. In such an implementation, the size of each individual node (usually large) will determine the maximum number of nodes in the list.

Difficulty in interfacing "externals" with Turbo source code. This was particularly true, when the external accessed BIOS-ROM routines through interrupt vectors. Also, the Turbo documentation in this area is poor.

Fixed size real number representation (6 bytes).

Procedure "new" will not accept variant record specifications.

10.2 Limitations in the standard I.B.M. PC hardware

The concepts utilized in the implementation of "Micro Stream" requires extensive floating point number processing. However, the 8088 processor is limited to integer number processing. This limitation can be overcome through the use of the 8087 math chip, for which I.B.M. PC has a readily available socket in its architecture. This processor expands the instruction set of the 8088 to provide it with 80-bit floating point operations. This includes arithmetic operations such as
additions, subtraction, multiplication, division and calculation of trigonometric functions. Consequently, the use of the 8087 chip will greatly enhance the execution speed of "Micro Stream".

Another feature contributing to the loss of execution speed is the 8-bit data bus that the 8088 utilizes for outside processor communications. This means, every instruction requiring 16 bits from memory requires two 8-bit memory fetches. Although this action is transparent to the programmer, it can slow down the execution of the program. The AT&T 6300 utilizing the 8086 (16/16) runs "Micro Stream" faster.

Due to the program chaining utilized in this package, a hard disk would enhance the speed of program loading from disk.
11. CONCLUSION

The CAD package "STREAM" (including the provision of fully three dimensional graphic support) has been successfully implemented on a standard I.B.M. personal computer. Many features were added to enhance the user friendliness of this program. The incorporation of B-Spline theory removed the limited design environment that was available in the original version of "STREAM".

Throughout the design of the software, provision has been made to facilitate future expansion of this package. In particular, the chaining facility allow additions such as "Micro Cutter" to be developed independently, as completely separate programs. Subsequently, it can be chained to the current package through the main menu. Likewise, since the graphics package is implemented as a separate program, its routines are available to program other than "Micro Stream".

The resolution of the currently available I.B.M. color monitor is not adequate for high level graphics application. A high resolution screen particularly with a pixel density of 1:1 would have greatly facilitated the development of the graphics routines.

The programming environment offered by the I.B.M. PC is a very pleasant one, mainly due to the provision of the BIOS-ROM routines and the DOS services (available in DOS 2.1). The accessibility to these services are excellent. However, certain features that are transparent to the software developer in a minicomputer environment, needs careful thought in the microcomputer environment, the main one being the effective memory utilization. Also the computer language to be utilized
for a particular project must be given careful consideration. Some compilers enforce severe limitations, others provide extensions to the standard definition of the language and some are not proven and consequently not reliable. The use of non standard features will facilitate the coding process, but will hinder the portability of the program.

Concluding, it can be said that the development of major software on a microcomputer, is feasible and practical. Although the current 16-bit computers require more programming effort (than the minicomputers), the new 32-bit virtual memory machines running the UNIX™ operating system (eg. Interpro 32) should remove most of the differences that currently exist between these two environments.
12. REFERENCE

V. Nagpal, T. Altan and R. Gagne

J.S. Gunasekera
ASME, Int. Computer Technology Conference, San Francisco
pp 452-459 (1980)

V. Nagpal, C.F. Billhardt, T. Altan

J.S. Gunasekera
Symposium, CAD/CAM in the 80's and integration through data bases. Melbourne, pp 20-22 (October 1980)

J.S. Gunasekera and S. Hoshino
J.S. Gunasekera, H.L. Gegel (et. al.)

[7] Computer Aided Design and Manufacture for Extrusion of Aluminium, Titanium and Steel Structural Parts (phase 1)
V. Nagpal and T. Altan
AVSCOM report No. 76-12, AMMRC CRT 76-6, U.S. Army (March 1976)

J.S. Gunasekera, H.L. Gegel (et. al)
Annals of the CIRP, Vol. 33/1/1984

[9] Computer Aided Manufacturing of Streamlined Extrusion Dies
M.J.K. Jayasuriya
Department of Mechanical Engineering
Monash University, Clayton, Victoria 3168
Australia

[10] CAD/CAM: Plotting the Impact of Shakeout
Donald Williamson
MOVIE.BYU (MOVIE.Brigham Young University)

A general purpose computer graphics system


All correspondence should be addressed to:

Hank Christiansen
Civil Engineering
368 cb byu
Provo, Utah 84602
(801) 328 - 6325

Principles of Interactive Computer Graphics 2nd Ed.
William M. Newman  Robert F. Sproull (1979)
McGraw-Hill International Book Company
Sydney Australia

Mathematical elements of Computer Graphics
David F. Rogers  J. Alan Adams (1976)
McGraw-Hill Book company
New York

Numerical Control, Mathematics and applications
P.E. Bezier (1972)
John Wiley & Sons Inc., London
R.F. Riesenfeld
Ph.D Dissertation, Syracuse University (1973)
Computer Science

[16] On Calculating with B-Splines
Carl De Boor
Journal of Approximation Theory 6, 50-62 (1972)

M.G. Cox
Journal of Institute of Mathematics and its Applications 10, 134-149 (1972)

Ray Duncan
Dr. Dobb's Journal, Number 69, July 1982

Jack Bresenham
Communications of the ACM February 1979
volume 20 number2, pp 100-106
    "Technical Reference"

    Disk Operating System (version 2.1)
    Microsoft Corp.

[22] 8088 Macro Assembler Programming
    Dan Rollins

[23] I.B.M. PC Assembly Language. A guide for programmers
    Leo J. Scanlan
    Robert J. Brady Co, A prentice-Hall Publishing and
    Communication Company, Bowie, MD 20715 (1985)

    (version 3.0) 2nd Ed. (June 1985)
    Borland International Inc.
    4585 Scotts Valley Drive
    Scotts Valley, CA 95066

[25] Direct all inquiries regarding "STREAM" to :
    Dr. Jay Gunasekera
    Supertechnology International Inc.
    P.O. Box 2305, Athens, Ohio 45701, U.S.A.
APPENDIX A

Running "Micro Stream"
1. STARTING OFF

Two diskettes are required to run "Micro Stream". One is the program disk and the other can be any formatted MS DOS (version 2.1) compatible disk. The first requirement is to place the system files and Graphics.Com (this code file is resident on the original DOS disk and permits the user to obtain screen dumps in the graphics mode) onto the program disk (refer the DOS manual). Then placing this disk in drive A and rebooting the system will activate "Micro Stream". The second disk should be in drive B and is utilized for storing data files generated by the design program, "Stream".

After a brief message (figure B-1), the main menu appears on the screen (figure B-2). Here, the user can make a choice by a single key entry, i.e. <rtn> not required (most command line entries require only single key entries). At present, the choice is limited to "Stream" or "Graphics".

1.1 "Stream"

Choosing "S" from figure B-2 would result in the display of input page 1 (figure B-3) of "Stream" (it has a total of four pages). Here, top most line (the status line) always reports the active module, the current page and the active product file name. The bottom line is the command line.

THE PRODUCT FILE: This is the file which contains the product geometry coordinates (figure B-3). It is created on disk B by "Stream",
once the user has completed the product definition.

Micro STREAM and Micro CUTTER

CAD / CAM

for Streamlined Extrusion Dies

version 1.

All rights reserved: M.J.K. Jayasuriya
Jay S. Gunasekera

Main menu

Micro STREAM and Micro CUTTER

Module selection (enter single character) >>>>

Module: Stream  Input Page:  Product file =

File name [≤ 8 chrs: Alpha/Numeric and #] >>>>  sqr#l
NEW DESIGN: Enter a valid name of a file that is currently not resident on disk B. "Stream" assumes a new file, prints a brief message to that extent and enters the input mode permitting the user to enter electrode geometry data into input page 1 (figure B-4). On completing the data entry, the program will automatically enter the edit mode (see section below on "Old-File").

RECALLING AN OLD FILE: Enter a valid name of a file that is resident on disk B. "Stream" will read this file, fill the input page 1 with this data and enter the edit mode (figure B-5). The command line now displays valid edit commands. Here, changes to existing data can be made. On completion, <esc> will cause an exit from input page 1.

Notes: [a] The field width at each entry point is indicated by the series of underscores on the screen. As new entries are made, the underscores are removed to indicate the left over space in the entry field. Likewise, as entries are deleted, underscores are added to the entry field.

[b] The field format of the first two entries allows positive real numbers with up to four significant digits and four decimal points. The third entry allows a positive integer with up to three significant digits and the fourth a single digit.

[c] The product file name is displayed in the status line.

[d] Each data entry must be followed by a <enter>. 
Module: Stream  Input Page_1  Product file = new

Types of stream lines

Electrode length >>>> 4.0000  [1] Straight Converging die
Number of sections >>>> 5  [3] Concave-drawing type
[4] Parabolic
[5] Cubic Streamline
[6] Third order area basis

< crt > [move cursor] R (replace) < esc > [to quit]
PRODUCT GEOMETRY: The second page of "Stream" is reserved for product geometry definition (figures B-6 & B-7). If the file name is an old file, the page is filled with the data in the file, the product displayed and the program enters the edit mode. This mode is analogous to that described in the previous section with one exception. The left viewport, which is used to enter the product coordinates is further divided into separate pages. Each page accommodates 7 nodes and an unlimited number of pages are available. Moving between pages is done using the function keys F1 and F2 (see command line in figure B-6). Any modification to the geometry can be immediately displayed using the "D" key.

If the design is new, on entry into this page the input mode is activated (figure B-7) which allows the user to enter x,y coordinates of each product node and the corner radii (the latter feature is currently not implemented). On entering 7 nodes, the page automatically increments to a fresh page. Entering "Q" quits from the input mode and enters the edit mode (see above section).

NOTES: [a] The field width of x, y and r entries allow real numbers with up to four significant digits and four decimal points.

[b] Each data entry must be followed by an <enter>.
Module: stream

Product definition

Product File = new

<table>
<thead>
<tr>
<th>Node</th>
<th>x</th>
<th>y</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>-0.5000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>3</td>
<td>-0.5000</td>
<td>-1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>4</td>
<td>0.5000</td>
<td>-1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>5</td>
<td>0.5000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

F1 (Next_Pg) F2 (Pre_Pg) R (replce)
D (display (crt) (to move cursor)
Q (quit)

E (exit from input mode)
THE MAPPING: On exiting from page two, the user is presented with two choices.

C (ontinue <esc> (to main menu)

Entering "C" will cause "Stream" to proceed with calculations. However, pressing <esc> enables the user to terminate further processing and return to the main menu. Prior to either of these function, the following is displayed.

Save product design file? [y/n] >>>>

Entering "y" will cause "Stream" to write the current product and electrode geometry into a data file <file_name>.Pro, on disk B.

Now, if "Stream" is to continue, the next page displays the "mapping" resulting from the first stage of design calculations (figure B-8). The empty space in the left viewport is reserved for future extensions. On exiting from this page, the user is presented with the following.

Save map file? [y/n] >>>>

Entering "Y" will cause "Stream" to write the current mapping data into a data file <file_Name>.MAP, on disk B. This file serves as input to the "Graphics" module. Therefore, in order to view the electrode geometry, this file must be created on disk B.

DESIGN CALCULATION: The results from the second stage design calculations are displayed in this last page of stream (figure B-9).
Module: Stream  mapping  Product file = sqr#1

Press any key to continue

B - 8

Module: Stream  Design Calculations  Product file = sqr#1

Product perimeter = 4.00
X-Sec area of billet = 12.57
X-Sec area of product = 1.00

Volume inside die = 24.3362
Surface area = 35.6794
Extrusion ratio = 12.5664

Run Stream again [y/n] >>>>>
1.2 "Graphics"

Choosing "G" from the main menu (figure B-2) will activate the graphics module. The status line at the top reports the active module and the display file. The bottom line is the command line. Single key entries are sufficient to activate any of the commands. Entering "Q" will cause an exit from this module to the main menu.

SCREEN MODES: Two screen modes are provided. One uses the full screen (figure B-10) and the other splits it into two viewports (figure B-11). It is possible to freely swap from one mode to another (see the corresponding command lines in figures B-10 and B-11). In the split mode, there is an additional command "C (change)", which allows the user to toggle the active viewport. The active viewport is indicated in the status line, right hand corner (in figure B-11, active viewport is the right one, indicated by "R"). The two functions, draw and rotate operates on the active viewport.

DRAWING A DESIGN: Entering "D" results in the following prompt.

File name [<= 8 chars : Alpha/Numeric and #] >>>> _______

There are two input possibilities here. Entering a valid file name resident on disk B will cause "Graphics" to read this file and display the geometry on the active screen. Note that "Graphics" assumes an file extension "MAP". If the file is not found in the drive B disk directory, the following message is displayed.
B: <file_name>.MAP is not resident on disk Press any key to continue

The other input possibility is to press <enter> for a file name. This indicates to "Graphics", that the user intends to work on the current display file already resident in memory. However, if no display file is currently active, the following error message will be displayed.

No display file Press any key to continue

Note that it is possible to have two different display files in the split screen mode. The left viewport can be displaying file_1.MAP and the right, file_2.MAP (figure B-12). This facility is provided so that the electrode design and the corresponding tool paths generated by "Micro Cutter" (a future extension) can be viewed simultaneously on one screen.

ROTATION: The rotate function permits the user to view the display from any angle (figures B-11 & 12). Three angle values, x,y and z in that order are requested from the user (the rotations are carried out in the same order). Each entry is allowed to be a positive/negative real number with up to three significant digits and two decimal digits. Each entry must be terminated with an <enter>. A null entry is permitted and is interpreted as a zero degree rotation. Counter clockwise rotations about the positive axes are considered positive. If a rotation is requested when there is no active display file, the following error message is displayed.

No display file Press any key to continue
Module: Graphics
Display file = B:sqr#1.MAP

S (plit D (raw V (iew R (otate Q (uit

B - 10

Module: Graphics
Display file = B:hex.MAP

F (ull C (hange D (raw V (iew R (otate Q (uit

B - 11
Module: Graphics
Display file = B:sqr.MAP

Angles of rotation. < rtm > for null entry > > > > x = __________

B - 12
APPENDIX B

Source code

- Frame.Pas
- Scan_Cod.ASM
- Grphics.Pas (B_Spline on page 109)
procedure Frame (Up_Row, Up_Colmn, Low_Row, Low_Colmn : byte);

{ ...............................................................
Given the window coordinates draw a frame around it. Uses
standard 1.B.M. characters.

input : window coordinates called by : Main_Menu in
Strt_Up.pas.

output : draw frame

{ ...............................................................
}

var I : byte;  { Index }
begin  { frame }

[ draw top line ]
  TextColor (14); GotoXY (Up_Colmn, Up_Row); write (chr(201));
  for I := (Up_Colmn + 1) to (Low_Colmn - 1) do
    write (chr(205)); write(chr(187));

[ draw verticles ]
  for I := (Up_Row + 1) to (Low_Row - 1) do
    begin
      GotoXY (Up_Colmn, I); write (chr(186));
      GotoXY (Low_Colmn, I); write (chr(186))
    end;

[ draw bottom line ]
  GotoXY (Up_Colmn, Low_Row); write (chr(200));
  for I := (Up_Colmn + 1) to (Low_Colmn - 1) do
    write (chr(205));
  write (chr(188))

end;  { frame }
Read_Key: reads the scan code of any key.

Returns the value in AL. Routine is meant to be an external and a function of the type byte.

```
Key_In   equ 16h ; keyboard I/O ROM call

Input_Rl segment
assume CS: Input_Rl

Read_Key proc near
; ------ entry requirements
push BP
mov BP,SP
push DS
push ES
push SS

; ------ source code
xor AH,AH ; read char. function
int Key_In
mov AL,AH
xor AH,AH

; ------ exit requirements
pop SS
pop ES
pop DS
mov SP,BP ; restore SP to entry level
pop BP
ret 1 ; remove function result from stack [type byte : 1 byte]

Read_Key endp

Input_Rl ends

end Read_Key ; end of program
```
program Graphics (input, output);

{ This is the driver for all graphics display routines.

input : - called by : Batch.BAT

output : - global types :

}

{ Global type definitions used in the graphics program, Graphics.Pas

}

type Line Disp = string [80];
Str_1 = string [16];
Characters = set of char;

Dsg_Ptr = ^ Dsg_Node; { type for electrode node data list }
Dsg_Node = record
  x, y, z : real;
  F_Link : Dsg_Ptr
end;

Dsp_Ptr = ^ Dsp_Node; { type for image space object data list }
Dsp_Node = record
  X, Y, Z : integer;
  F_Link : Dsp_Ptr
end;

Ptr_1 = ^ File_Node; { type for product data list read from disk }
File_Node = record
  Pro_x, Pro_y, Billet_x, Billet_y : real;
  F_Link : Ptr_1
end;

Hrk_Ptr = ^ integer;

var Coands : Characters;
Exit, Active_Scrn : boolean;
S : Line Disp;
Key : char;
V_Center_x, V_Center_y : integer; { center of the viewport }
V_x, V_y : integer; { absolute dimensions of (length & height) of viewport }
Elec_Lenth : real;
World_List : Dsg_Ptr; { electrode node data }
Disp_List : Dsp_Ptr; { image space data }
procedure Clr_Scr_Gra (Strt_Row, Strt_Col, End_Row, End_Col : integer);

{ Clears an active window in the graphics mode by executing a ROM interrupt. All registers are preserved. The origin is 0,0 and is located at the upper left corner of the window. The coordinates must be specified as rows and columns, NOT in pixel notation. eg. 0..24, 0..79 clears the entire screen irrespective of the window defined by Graph Window.

input : Strt_Row, Strt_Col, End_Row called by : [Read_File, Display, Draw_Main_Proc] in Gra_Uti2.pas
        [Split_Scrn, Full_Scrn, Rotate] in Gra_Uti4.pas

output : none global types : none

)                                               

  type Reg_Pack = record
     ax,bx,cx,dx,bp,di,si,ds,es,flag : integer
  end;

  var al, cl, dl : byte;
  registers     : Reg_Pack;

  Begin { Clr_Scr_Gra }
      al := 0; cl := Strt_Col; dl := End_Col;
        with registers do
          begin
             ax := 6 shl 8 + al;
             cx := Strt_Row shl 8 + cl;
             dx := End_Row shl 8 + dl;
             bx := 0
          end;
      intr ($10, Registers);
  end;   { Clr_Scr_Gra }

procedure Box_in_Gra (X_Up, Y_Up, X_Down, Y_Down : integer);

{ Draw a frame defined by input parameters. Module to be used in HiRes graphics mode.

input : X_Up, Y_Up, X_Down, Y_Down called by : Graphics.pas (wide usage)

output : none global types : none

)
begin { Box_In_Gra }
  draw (X_Up, Y_Up, X_Up, Y_Down, l);
  draw (X_Up, Y_Down, X_Down, Y_Down, l);
  draw (X_Down, Y_Down, X_Down, Y_Up, l);
  draw (X_Down, Y_Up, X_Up, Y_Up, l)
end; { Box_In_Gra }

procedure Disp_Line (S : Line_Disp; Colmn, Row : byte);
  begin { Disp_Line }
    GotoXY (Colmn,Row);     (screen coordinates)
    write (S)
  end; { Disp_Line }

procedure Strg_Read (var In_Str : Str_l; Valid_Chrs : Characters;
  Max_Lenth, Colmn, Row : byte);
  begin { Strg_Read }
    GotoXY (Colmn,Row);     (screen coordinates)
    write (S)
  end; { Strg_Read }

const cr    = 13;
bs           = 8;
Fld_Chr     = '\_';
Dot          = '
';
three        = 3;
var i : byte;
Entry : char;
Exit : boolean;
Ok_Set : Characters;

begin  { Strg_Read }
In_Str := ''; Exit := false;  { initialize }
Ok_Set := Valid_Chrs + [chr(bs), chr(cr)];
GotoXY (Colnm,Row);  { start of entry field }
for i := 1 to Max_Lenth do
  write (Fld_Chr); GotoXY (Colmn,Row);
repeat
  repeat read (kbd, Entry); until Entry in Ok_Set;
  if ((ord(In_Str[0]) = 0) and ('-' in Ok_Set)
     and (ord(Entry) <> bs)) then
    Ok_Set := Ok_Set - ['-'];  { Disnable multiple entries of - sign }
    case ord(Entry) of
      bs : begin
        i := ord(In_Str[0]);
        if i = 1 then  { If needed add - sign back
          into set variable }
          if '-' in Valid_Chrs then
            Ok_Set := Ok_Set + ['-'];
          if i <> 0 then  { can delete }
            begin
              Delete (In_Str, i, 1);
              GotoXY (Colmn + i - 1, Row);
              write (Fld_Chr);
              GotoXY (Colmn + i - 1, Row)
            end
        else
          { no more characters in In_Str }
        end;
      cr : if Chr(cr) in Valid_Chrs then
        Exit := true
      else
        if ord(In_Str[0]) <> 0 then
          Exit := true
        else
          { empty string cannot be returned to
            caller. Go back to input of data }
        end;
    else  { valid character to be inserted to string }
      if ord(In_Str[0]) < Max_Lenth then  { there is space }
        begin
          write (Entry);  { echo to screen }
          In_Str := In_Str + Entry;
        end;
    end;
  end;
end;
if (ord(ln-Str[0]) = three) and 
(Dot in Valid_Chrs) then 
begin 
  \{ determine whether the decimal 
  point has already been added to 
  the string by the user \} 
  if Pos(Dot, In_Str) = 0 then 
  begin 
    write (Dot); 
    In_Str := In_Str + Dot 
  end 
end 
else 
  \{ no space in In_Str. Ignore extra entries \} 
begin sound(440); delay(550); nosound end 
end; \{ case \} 
until Exit 
end; \{ Strg_Read \} 

procedure Draw_MainProc (var Disp_List : Dsp_Ptr; var World_List : Dsg_Ptr; 
  V_Center_x, V_Center_y : integer; 
  V_x, V_y : real; Status : Line_Disp; 
  var Elec_Lenth : real; var Sections, Strm_Line : 
 byte; var Heap_At_Strt, Heap_At_Eye : Mrk_Ptr); 
{ ----------------------------------------------- 
 Reads a *.MAP file from disk b: and fits B_Splines to this data. Then a full 
3-0 to 2-0 perspective transformation is carried out to obtain real screen 
coordinates. These are then mapped to a screen view port. The depth info. 
is preserved for hidden line removal purposes. Also the heap space is 
managed. 

input : All parameters except Heap_At_Eye \hspace{1cm} \text{called by: Graphics.pas} 
output : Heap_At_Eye. This variable must be \hspace{1cm} \text{global types : [Dsp_Ptr,} 
maintained accurately during the entire \hspace{1cm} Dsg_Ptr,
execution of Graphics.pas. Therefore it \hspace{1cm} Line_disp 
is a global. \hspace{1cm} Mrk_Ptr] in 
gra_typ.pas 

----------------------------------------------- 
} 

const Max_Vertices = 10; \{ 0..10 = 11 vertices for defining 
\hspace{1cm} \text{polyon in B-Spline} \} 

type Vertices = array [0..Max_Vertices] of real; \{ therefore n = 10 \} 

var Dsg_List, Lead_P, Tra_P : Ptr_I;
  Max_z, Min_z, Max_x, Min_x,
  Max_y, Min_y, C_I 
  : real;
  Valid_Chrs : Characters;
  Eye_List : Dsg_Ptr;
  i : byte;
  On_Disk : boolean;
procedure Read_File (var Dsg_List : Ptr_1; var Elec_Lenth : real; var Sections, Strm_Line : byte; var On_Disk : boolean; var File_Name : Str_1; var Heap_At_Strt : Mrk_Ptr);

{
  Reads a text file from disk b:. First a check is made to determine whether the file exists on disk b:. If not found a null string is passed back to the caller with On_Disk negated. Also reinitializes the heap space for a new job

  input : Heap_At_Strt. this is a pointer to
  very start of the heap space. initialized
  in Graphics.pas

  output : data in disk file < File_Name.MAP >. Sections, Strm_Line, Elec_Lenth and Dsg_List. Also status indicator On_Disk
  and the File_Name.

  global types : [Ptr_1, Str_1, Mrk_Ptr] in Graphics
  ---------------------------------------------------------------------------
}

const Cr = 13;
Max_Lenth = 8;

var Colmn, Row : byte;
S : Line_Disp;
Valid_Chrs : Characters;
File_Var : text;
Temp_P : Ptr_1;

function Exist (File_Name : Str_1) : boolean;

{
  Determine whether file Str_1 is resident on disk.

  input : Str_1                    called by : [Read_File]

  output : function result Exist        global types : none
  ---------------------------------------------------------------------------

}

var Fil : File;      { untyped }

begin { Exist }
assign (Fil, File_Name);

reset (Fil);

Exist := (10result = 0);
close (Fil)
end;  ( Exist )

begin  ( Read_File )

File_Name := ''; On_Disk := false;
S := 'File name [ <= 8 chrs: Alpha/Numeric and # ] >> > ';
Clr_Scr_Gra (24,0,24,79);   [ erase previous entry ]
Disp_Line (S,1,25);
Valid_Chrs := ['A'..'Z', 'a'..'z', '#', '0'..'9'] + [chr(Cr)];
Strg_Read (File_Name, Valid_Chrs, Max_Lenth, 53, 25);
Dsg_List := nil; Clr_Scr_Gra (24,0,24,79);

if ord (File_Name[0]) <> 0 then
begin

File_Name := '8:' + File_Name + '.HAP1';
if Exist (File_Name) then
begin

{ new job. reinitialize all heap space }
release (Heap_At_Strt);

GotoXY (50,1); write ('reading file');
assign (File_Var, File_Name); On_Disk := true;
reset (File_Var);   [ prepare for reading only ]
readin (File_Var, Elec_Lenth, Sections, Strm_Line);
Temp_P := nil;
while not eof (File_Var) do
begin
if Temp_P = nil then
new (Temp_P)
else
begin
new (Temp_P ':.F_Link);
Temp_P := Temp_P ':.F_Link
end;
Temp_P ':.F_Link := nil;

with Temp_P do
readln (File_Var, Pro_x, Pro_y,
       Billet_x, Billet_y);
if Dsg_List = nil then
Dsg_List := Temp_P
end;  ( while .... do )
close (File_Var);
GotoXY (50,1); Clr_Scr_Gra (0,50,0,70); write (File_Name)

end
end}  ( Read_File )
procedure Streamline (Strm_Line : integer; var n_Plus_1, k : integer; Elec_Lenth : real; var V_x, V_y, V_z : Vertices ;Temp_D_P : Ptr_i);
{
Define the open polygon vertices for the defining polygon.

input : Strm_Line, Elec_Lenth, Temp_D_P called by : B_Spline

output : V_x, V_y, V_z, n_Plus_1, k

{ array index }
begin ( Streamline ) for i := 0 to Max_Vertices do { initialise }
begin
V_x [i] := 0;
V_y [i] := 0;
V_z [i] := 0
end;

{ assign vertices }
case Strm_line of
5 : { cubic streamline } begin
k := 4; n_Plus_1 := 4;
for i := 0 to 3 do

end
end}
end { case }
end; { Streamline }

procedure B_Spline (Dsg_List : Ptr_l; var World_List : Dsg_Ptr; Sections,
     Strm_Line : integer; Elec_Lenth : real);

{ Implementation of the De Boor algorithm for the evaluation of B-Splines. 
  No restrictions are imposed, i.e. the algorithm as described in chapter 
  3 of the thesis is implemented. Multiple vertices are allowed. }

input : Dsg_List, Strm_Line, Elec_Lenth, Sections  called by : Draw_Main_
     Proc

output : World_List  global types : [Ptr_l, Dsg_Ptr] in Gra_ 
         typ.Pas

const  Max_Order  = 7;

type  Knot_Vector = array [0..17] of real;  { 0..(n+k) }
    axis  = (x_axis, y_axis, z_axis);

var  Temp_W_P : Dsg_Ptr;
    Temp_D_P : Ptr_l;
    Increment, t : real;
    Co Ordinate : axis;
    n_Plus_l,
    k,
    i, l, S, r, Index : integer;
    A, DP, DM : array [1..Max_Order] of real  { see De Boor }; 
    X : Knot_Vector;
    V_x, V_y, V_z : Vertices;

procedure Knots (var X : Knot_Vector; n_Plus_l, k : byte;
                   V_x, V_y, V_z : Vertices);

{ Given the information in the interface, calculate the knot vector 
  X. Multiple vertices are allowed }

input : n_Plus_l, k, V_x, V_y, V_z  called by : B_Splines

output : X (the knot vector)  global types : [Vertices] in Draw 
                      Main_Proc
                    [Knot_Vector] in B
                      -Splines
var n, j : byte;

begin { Knots }

n := 0;

for j := 0 to (n_Plus_1 - 1) + k do

X [j] := 0;

for j := 0 to k - 1 do

X [j] := 0;

if n_Plus_1 = k then

X [j] := 1;
else

begin

for j := k to n_Plus_1 do

begin

n := n + 1;

if (abs (V_x [n] - V_x [n-1]) <= 0.01) and
(abs (V_y [n] - V_y [n-1]) <= 0.01) and
(abs (V_z [n] - V_z [n-1]) <= 0.01)) then

X [j] := X [j-1]
else

X [j] := X [j-1] + 1
end;

for j := (n_Plus_1 + 1) to (n_Plus_1 - 1) + k do

X [j] := X [j-1]

end; { Knots }

begin { B_Spline }

for r := 1 to Max_Order do

begin

A[r] := 0;
DP[r] := 0;
DM[r] := 0
end;

Temp_W_P := nil; World_List := nil; Temp_D_P := Dsg_List;
repeat { for all streamlines on electrode }

Streamline (Strm_Line, n_Plus_l, k, Elec_Lenth, V_x, V_y, V_z, Temp_D_P);

Knots (X, n_Plus_l, k, V_x, V_y, V_z);

{ Note : x_o <= t < x Max. Therefore t_max = X (n+k) }
Increment := X [(n_Plus_l - 1) + k] / Sections;

\( t := 0; \)
for \( l := 0 \) to Sections do
begin
if Temp_W_P = nil then
begin
new (Temp_W_P)
end
else
begin
new (Temp_W_P ^ .F_Link);
Temp_W_P := Temp_W_P ^ .F_Link
end;
with Temp_W_P ^ do
begin
  F_Link := nil;
  x := 0; y := 0; z := 0
end;
if ((l = 0) or \( l = \text{Sections} \)) then
begin
if \( l = 0 \) then { first vertex }
begin
with Temp_W_P ^ do
begin
  \{ billet end \}
  x := Temp_D_P ^ .Billet_x;
  y := Temp_D_P ^ .Billet_y;
  z := 0
end
end
else { last vertex }
with Temp_W_P ^ do
begin
  \{ product end \}
  x := Temp_D_P ^ .Pro_x;
  y := Temp_D_P ^ .Pro_y;
  z := Elec_Lenth
end
end
else { intermediate points }
{ evaluate \( F(t) \), De Boor algorithm }
begin
i := 0; { find i }
repeat
  i := i + 1;
until ((t < X [i+1]) and \( t \geq X[i] \));
(calculate DP(r) and DM(r))
for r := 1 to k do
begin
   DP[r] := X[i + r] - t;
   DM[r] := t - X[i - k + r]
end;

for Co_Ordinate := x_axis to z_axis do
begin
   case Co_Ordinate of
      x_axis : for r := 1 to k do
                  A[r] := V_x[i - k + r];
      y_axis : for r := 1 to k do
                  A[r] := V_y[i - k + r];
      z_axis : for r := 1 to k do
                  A[r] := V_z[i - k + r]
   end;  { case }
for S := 1 to k - 1 do
begin
   Index := 1;
   for r := S + 1 to k do
      begin
         A[Index] :=
                  (((Dm[r] * A[Index + 1]) +
                    (DP[r-S] * A[Index]))) /
                  (DM[r] + DP[r-S]);
         Index := Index + 1
      end;  { for r .... }
end;  { for S ...... }

{ calculate coordinates }
case Co_Ordinate of
   x_axis : Temp_W_P.x := A[1];
   y_axis : Temp_W_P.y := A[1];
   z_axis : Temp_W_P.z := A[1]
end;  { case }
end;  { for Co_Ordinate .... }
end;  { if ... then }

if World_List = nil then
   World_List := Temp_W_P;

   t := t + Increment
end;  { for l ...... }

Temp_D_P := Temp_D_P.f_Link;

   until Temp_D_P = nil;  { do complete electrode }
end;  { B_Spline }
procedure World_To_Eye (var world_List, Eye_List : Dsg_Ptr;
    var Max_z, Min_z : real);

{ -----------------------------------------------------------------------------
Transform the origin from the world system to the viewpoint. Also
invert z since the eye system is a left handed system where as the
world system is right handed. Determine Max_z and Min_z in the eye
system. Due to this latter step, this transformation could not be
combined with the Eye_To_Scrn transformation.

input : World_list called by : [Draw_Main_Proc] in Gra_Util2.pas
    This list is linear
    thus terminated by nil.

output : Eye_List, Max_Z, Min_z  global types : [Dsg_Ptr] in Gra_TYP.pas
-----------------------------------------------------------------------------

const Eye_To_Obj = 20;  { inches. Really should be dimensionless }

var Temp_P, Lead_P : Dsg_Ptr;

begin ( World_To_Eye )
    Temp_P := World_List;
    Eye_List := nil; Lead_P := nil;
    Max_z := 0; Min_z := -(Temp_P ^.z - Eye_To_Obj);
    repeat
        if Lead_P = nil then
            new (Lead_P)
        else
            begin
                new (Lead_P ^.F_Link);
                Lead_P := Lead_P ^.F_link
            end;
        Lead_P ^.F_Link := nil;
        with Lead_P do
            begin
                { In order to increase efficiency, eye x and eye y
                are obtained from the world_list, whenever required. }

                x := 0; y := 0;
                z := -(Temp_P ^.z - Eye_To_Obj);

                if z > Max_z then
                    Max_z := z
                else
                    if z < Min_z then
                        Min_z := z;

                if Eye_List = nil then
                    Eye_List := Lead_P;
                Temp_P := Temp_P ^.F_Link
            end

end;
procedure Eye_To_Scrn (var Scrn_List, World_List : Dsg_Ptr; Max_z, Min_z : real; var Max_x, Min_x, Max_y, Min_y : real);

\{ Carryout the transformation from Eye system to the real screen system. \}

Note that this transformation does not do any scaling. The transformed
coordinates are written back into the Eye_List thus destroying the original
Eye_List. Also the following identities hold as regard to the transformation.
i.e. F = Max_z and Eye_To_Scrn = Min_z = 0.

input : Scrn_List, World_List, Max_z, Min_z called by : [Draw_Main_Proc]
Scrn_List and World list are terminated in Gra_Utilz.pas
by nil.

output : altered Scrn_List, Max_x, Min_x, global types : [Dsg ptr] in
Max_y, Min_y Gra_Typ.pas
These min-max values are signed values

\}

const Scrn_Size = 11.4; \[ S \]

var Temp_S_P, Temp_W_P : Dsg_Ptr;
\ w : real;

begin \{ Eye_To_Scrn \}
Temp_S_P := Scrn_List;
Temp_W_P := World_List;
\ w := (Scrnn_Size * Temp_S_P \'.z\) / Min_z;
Max_x := Temp_W_P \'.x\ / \ w; Min_x := Temp_W_P \'.x\ / \ w;
Max_y := Temp_W_P \'.y\ / \ w; Min_y := Temp_W_P \'.y\ / \ w;

repeat
\ with Temp_S_P \ do
begin
\ w := (Scrnn_Size * z\) / Min_z;
\ x := Temp_W_P \'.x\ / \ w;
\ y := Temp_W_P \'.y\ / \ w;
\ z := (Scrnn_Size * (z / Min_z - 1)) / ((1 - Min_z / Max_z) * \ w);

if x > Max_x then
\ Max_x := x
else
\ if x < Min_x then
\ Min_x := x;
\ if y > Max_y then
\ Max_y := y
else
\ end; (World_To_Eye)
if \( y < \text{Min}_y \) then
\[ \text{Min}_y := y; \]
\[ \text{Temp}_S \cdot \text{P} := \text{F}_\text{Link}; \text{Temp}_W \cdot \text{P} := \text{Temp}_W \cdot \text{P} \cdot \text{F}_\text{Link} \]
end;
until \( \text{Temp}_S \cdot \text{P} = \text{nil} \)
end; { Eye_To_Scrn }

procedure Map_To_ViewPort (var Eye_List : Dsg_Ptr; var Disp_List : Dsp_Ptr;  
\( V_x, V_y, \text{Min}_x, \text{Max}_x, \text{Min}_y, \text{Max}_y \) : real;  
var \( \text{C}_1 \) : real);

{ Determine modified \( V_x \) and \( V_y \) to suite the viewport. Then apply the  
viewport transformation to obtain the pixel coordinates. Also the origin  
is moved to the upper left corner of the viewport to be compatible with  
the Pascal definition.}

\text{input : Eye_List, } V_x, V_y, \text{Max}_x, \text{Min}_x, \text{Max}_y, \text{Min}_y \text{ called by : [Draw_MainProc]  
Max_y, Min_y in Gra_utl2.pas  
output : Disp_List, } \text{C}_1 \text{ global types : [Dsg_Ptr, Dsp_Ptr] in  
Disp_List is circular } Gra_Typ.pas  

)

const \( \text{C}_2 = 1; \) { assume pixel density to be 1:1 }

var Scrn_Real_x, Scrn_Real_y,  
\text{Pixel_Den, } \text{Modi}_V_x,  
\text{Modi}_V_y := \text{real};  
\text{Tra}_P := \text{Dsg_Ptr};  
\text{Temp}_P := \text{Dsp_Ptr};

begin { Map_To_ViewPort }
\text{Scrn}_\text{Real}_x := \text{abs}(\text{Min}_x) + \text{abs}(\text{Max}_x);  
\text{Scrn}_\text{Real}_y := \text{abs}(\text{Min}_y) + \text{abs}(\text{Max}_y);  
\text{C}_1 := \text{Scrn}_\text{Real}_y / \text{Scrn}_\text{Real}_x;  
if \( \text{C}_1 \) \( \text{C}_2 \) then
begin  
\text{V}_x := \text{V}_y / \text{C}_1;  
\text{Pixel_Den} := \text{V}_y / \text{Scrn}_\text{Real}_y  
end
else  
begin  
\text{V}_y := \text{V}_x \cdot \text{C}_1;  
\text{Pixel_Den} := \text{V}_x / \text{Scrn}_\text{Real}_x  
end;

{ Generate Disp_List and discard Eye_List }  
\text{Tra}_P := \text{Eye_List}; \text{Temp}_P := \text{nil}; \text{Disp}_\text{List} := \text{nil};
repeat
if Temp_P = nil then
  new (Temp_P)
else
  begin
    new (Temp_P ^ F_Link);
    Temp_P := Temp_P ^ F_Link
  end;
Temp_P ^ F_Link := nil;
with Temp_P ^ do
begin
  { first translate origin to upper left corner of the world window. Also invert y to align with Pascal system }
  if C_1 > C_2 then
    begin
      Tra_P ^.x := (Tra_P ^.x - Min_x) * Pixel_Den
                   * 2.4;
      Tra_P ^.y := -(Tra_P ^.y - Max_y) * Pixel_Den
    end
  else
    begin
      Tra_P ^.x := (Tra_P ^.x - Min_x) * Pixel_Den;
      Tra_P ^.y := -(Tra_P ^.y - Max_y) * Pixel_Den / 2.4
    end;
  x := trunc (Tra_P ^.x);
y := trunc (Tra_P ^.y);
z := Tra_P ^.z;
  Tra_P := Tra_P ^ F_link
end;
if Disp_List = nil then
  Disp_List := Temp_P;
until Tra_P = nil;
Temp_P ^.F_Link := Disp_List;        { Disp_List is circular }
end;    { Map_To_ViewPort } 

procedure Display (Disp_List : Dsp_Ptr; Sections : byte; V_Center_x, V_Center_y : integer; V_x, V_y, C_l : real);

{---------------------------------------------------------------}
Using parameters that define the viewport a Graphic window is declared. This allows easy reference to any viewport on the screen. Then the display is output to the standard output device.

input : all parameters called by : [Draw_Main_Proc]
in Gra_Uti2.pas
var Prev_Sec, Tra_P, Lead_P : Dsp_Ptr;
i, j : byte;  // loop variables

begin  // Display
  Clr_Scr_Gra (trunc((V_Center_y - V_y / 2) / 8),
               trunc((V_Center_x - V_x / 2) / 8),
               trunc((V_Center_y + V_y / 2) / 8),
               trunc((V_Center_x + V_x / 2) / 8));

  if C_1 > C_2 then
    V_x := V_y * 2.4 / C_1
  else
    V_y := V_x * C_1 / 2.4;

  Lead_P := Disp_List;  // Disp_list is circular
  repeat streamlines
    repeat
      for i := 1 to Sections do
        with Lead_P ^ do
          begin
            draw (x, y, F_link ^ .x, F_link ^ .y, l);
            Lead_P := F_link
          end;
        Lead_P := Lead_P ^ .F_Link;  // start of next streamline
      until Lead_P = Disp_List;  // all streamlines done
    until Lead_P = Disp_List;

  [ do circumferential elements ]
  Tra_P := Disp_List; Lead_P := Disp_List; Prev_Sec := Disp_List;

  for i := 1 to Sections + 1 do  // do all sections
    begin
      repeat  // do one section at a time
        for j := 1 to Sections + 1 do
          begin
            Lead_P := Lead_P ^ .F_Link;  // adv: to next streamline
            with Lead_P ^ do
              begin
                draw (Tra_P ^ .x, Tra_P ^ .y, x, y, l);
                Tra_P := Lead_P
              end;
          end;
        until Tra_P = Prev_Sec;
        Prev_Sec := Prev_Sec ^ .F_Link;
        Tra_P := Prev_Sec; Lead_P := Prev_Sec
      end;  // for .... do
    end;  // Display
begin { Draw_Main_Proc }
  Clr_Scr_Gra (24,0,24,79);
  Read_File (Dsg_List, Elec_Lenth, Sections,
             Strm_Line, On_Disk, File_Name, Heap_At_Strt);
  if (not(On_Disk) and (ord(File_Name[0]) = 0)) then { empty string }
    begin
      if Dsg_List = nil then { no new display needed }
        begin
          if World_List <> nil then { display the existing display }
            begin
              release (Heap_At_Eye); { reclaim all memory above
                             the current World_List }
              Eye_List := nil; Disp_List := nil;
              World_To_Eye (World_List, Eye_List, Max_z, Min_z);
              Eye_To_Scrn (Eye_List, World_List, Max_z, Min_z,
                            Max_x, Min_x, Max_y, Min_y);
              Map_To_ViewPort (Eye_List, Disp_List, V_x, V_y,
                                Min_x, Max_x, Min_y, Max_y, C_x);
              Display (Disp_List, Sections, V_Center_x, V_Center_y,
                       V_x, V_y, C_x)
            end
          else
            begin
              sound (540); delay (550); nosound;
              Msg := 'No display file' +
                    ' Press any key to continue';
              Clr_Scr_Gra (24,0,24,79); Disp_Line (Msg,1,25);
              repeat until keypressed
            end
        end
    end
else
  if On_Disk then
    begin
      { New Dsg_List. Heap memory was initialized in Read_File }
      World_List := nil; Disp_List := nil; Eye_List := nil;

      { World_List and Dsg_List are terminated by a 'nil' node }
      B_Spline (Dsg_List, World_List, Sections,
                Strm_Line, Elec_Lenth);
      mark (Heap_At_Eye); { for release in above code in
                           this section }
      World_To_Eye (World_List, Eye_List, Max_z, Min_z);
Eye_To_Scrn (Eye_List, World_List, Max_z, Min_z,
Max_x, Min_x, Max_y, Min_y);

Map_To_ViewPort (Eye_List, Disp_List, V_x, V_y, Min_x,
Max_x, Min_y, Max_y, C_l);

Display (Disp_List, Sections, V_Center_x, V_Center_y,
V_x, V_y, C_l)
end

else
begin
sound (540); delay (540); nosound;
Msg := File_Name + ' is not resident on disk' +
  'Press any key to continue';
Clr_Scr_Gra (24,0,24,79); Disp_Line (Msg,1,25);
repeat until keypressed
end

GraphWindow (0,0,639,199); Clr_Scr_Gra (24,0,24,79);
Disp_Line (Status,1,25)
end; { Draw_Main_Proc }

procedure Split_Screen (var Active_Scrn : boolean;
  var Comnds : Characters; var S : Line_Displ;) var V_Center_x, V_Center_y : integer;
  var V_x, V_y : real);

----------------------------------------------*--------------------------
Utilized as a screen op. routine. Splits the screen half and set active
screen to left. Initialize all new screen parameters.

input : none cal led by : graphics

output : modified entry parameters global types : [Line_Displ, Char-
acters] in Gra_
Typ.pas

----------------------------------------------*--------------------------

begin { Split_screen }
  Clr_Scr_Gra (2,1,22,78); Draw (319,11,319,189,1); { center line }
  Active_Scrn := true; { left screen = true }

  { define viewport for left screen } V_x := 300; V_Center_x := 158; V_Center_y := 99; V_y := 125;
  GotoXY (80,1); write ('L');
  Clr_Scr_Gra (24,0,24,79); { erase command line }
  S := 'F (ull C (hange D (raw V (iew R (otate Q (uit;'
  Disp_Line (S,1,25);
  Comnds := ['F', 'C', 'D', 'V', 'R', 'Q']
end; { Split_Screen }

procedure Full_Scrn (var Comnds : Characters;
  var S : Line_Displ; var V_Center_x, V_Center_y :
integer; var V_x, V_y: real);

{-----------------------------------------------------------------------
restores the full screen and reset all parameters
input : none               called by : graphics
output : modified entry parameters  global types : [Line_Disp, Characters]
in Gra_Typ.pas
-----------------------------------------------------------------------}

begin ( Full_screen )
  Clr_Scr_Gra (2,1,22,78);  { erase left and right screens }
  Draw (319,11,319,188,0);  { erase center line }

  { define viewport for partial full screen }
  V_x := 390; V_Center_x := 319; V_Center_y := 99; V_y := 162.5;

  Clr_Scr_Gra (0,78,0,79);  { erase ‘L’ or ‘R’ }
  Clr_Scr_Gra (24,0,24,79);
  S := ’S’; Split_D (raw V_iew R otate Q uit’;
  Disp_Line (5,1,25);
  Comnds := ['S', 'D', 'V', 'R', 'Q']
end;  { Full_Screen }

procedure Change_Screen (var Active_Scrn: boolean; var V_Center_x,
                         V_Center_y: integer; var V_x, V_y: real);

{-----------------------------------------------------------------------
Toggle active screen. Applicable only in the split screen mode. Reset all
variables to the active screen.
input : none               called by : graphics
output : modified parameters  global types : none
-----------------------------------------------------------------------}

begin ( Change_Screen )
  V_Center_y := 99; V_x := 300; V_y := 125;

  if Active_Scrn then
     begin
       Active_Scrn := false;  { set to right }
       V_Center_x := 478;
       GotoXY (80,1); write ('R');  { display status }
     end
  else
     begin  { set screen to left }
       Active_Scrn := true;
       V_Center_x := 158;
       GotoXY (80,1); write ('L');  { display status }
     end
end;  { Change_Screen }
procedure Rotate (var World_List : Dsg_Ptr; Status : Line_Disp);
{
All rotations are performed in the original world coordinate system prior to any transformations. The coordinate system is: x to the right, y up and z towards the user out of the screen. Angles of rotations are read within this procedure. Order of rotation is x, y, z and is preserved. Rotations are concatenated.

input : World_List, Status called by : graphics
World_List is terminated by nil.

output : Modified World_List global types : [Dsg_Ptr, Line_Disp] in Gra_Typ.pas

-----------------------------------------------

const Max_Lenth = 6; { Decimal at 4 th place }
Cr = 13;

var Theta_x, { about x-axis }
Theta_y, { about y-axis }
Theta_z, { about z-axis }
Conv_To_Rad : real;
Temp_P : Dsg_Ptr;
S : Line_Disp;
Colmn, Row : byte;
Angle : Str_I;
Result : integer;
T_11, T_12, T_13, T_21, T_22, T_23, T_31, T_32, T_33,
T_x, T_y, T_z : real;
Valid_Chrs : Characters;

begin ( Rotate )
if World_List <> nil then { ccw rotations in the +ve axis direction > 0 }
begin
Valid_Chrs := ['0'..'9','-'',''] + [chr(Cr)];
S := 'Angles of rotation. < rtn > for null entry >>>> ';
Clr_Scr_Gra (24,0,24,79); Disp_Line (5,1,25);
S := ' x = '; Disp_Line (5,49,25); Angle := '';
Strg_Read (Angle, Valid_Chrs, Max_Lenth, 55, 25);
if ord (Angle[0]) = 0 then { null entries are special }
Theta_x := 0
else
val (Angle, Theta_x, result);
\[
\text{Angle} := 'y'; \\
\text{Clr_Scr_Gra} (24,49,24,79); S := 'y = '; \text{Disp_Line} (S,49,25); \\
\text{Strg_Read} (\text{Angle}, \text{Valid_Chrs}, \text{Max_Lenth}, 55, 25); \\
\text{if} \ \text{ord} (\text{Angle}[0]) = 0 \ \text{then} \\
\quad \text{Theta}_y := 0 \\
\text{else} \\
\quad \text{val} (\text{Angle}, \text{Theta}_y, \text{Result}); \\
\text{Angle} := 'z'; \\
\text{Clr_Scr_Gra} (24,49,24,79); S := 'z = '; \text{Disp_Line} (S,49,25); \\
\text{Strg_Read} (\text{Angle}, \text{Valid_Chrs}, \text{Max_Lenth}, 55, 25); \\
\text{if} \ \text{ord} (\text{Angle}[0]) = 0 \ \text{then} \quad \text{[null entries are special]} \\
\quad \text{Theta}_z := 0 \\
\text{else} \\
\quad \text{val} (\text{Angle}, \text{Theta}_z, \text{Result}); \\
\text{Clr_Scr_Gra} (24,0,24,79); \\
\text{Conv_To_Rad} := \pi / 180; \\
\quad \text{Theta}_x := \text{Theta}_x \times \text{Conv_To_Rad}; \\
\quad \text{Theta}_y := \text{Theta}_y \times \text{Conv_To_Rad}; \\
\quad \text{Theta}_z := \text{Theta}_z \times \text{Conv_To_Rad}; \\
\text{Temp_P} := \text{World_List}; \\
T_{11} := \cos (\text{Theta}_y) \times \cos (\text{Theta}_z); \\
T_{12} := -\cos (\text{Theta}_y) \times \sin (\text{Theta}_z); \\
T_{13} := \sin (\text{Theta}_y); \\
T_{21} := (\sin (\text{Theta}_x) \times \sin (\text{Theta}_y) \times \cos (\text{Theta}_z)) + \\
\quad (\cos (\text{Theta}_x) \times \sin (\text{Theta}_z)); \\
T_{22} := - (\sin (\text{Theta}_x) \times \sin (\text{Theta}_y) \times \sin (\text{Theta}_z)) + \\
\quad (\cos (\text{Theta}_x) \times \cos (\text{Theta}_z)); \\
T_{23} := - \sin (\text{Theta}_x) \times \cos (\text{Theta}_y); \\
T_{31} := - (\cos (\text{Theta}_x) \times \sin (\text{Theta}_y) \times \cos (\text{Theta}_z)) + \\
\quad (\sin (\text{Theta}_x) \times \sin (\text{Theta}_z)); \\
T_{32} := (\cos (\text{Theta}_x) \times \sin (\text{Theta}_y) \times \sin (\text{Theta}_z)) + \\
\quad (\sin (\text{Theta}_x) \times \cos (\text{Theta}_z)); \\
T_{33} := \cos (\text{Theta}_x) \times \cos (\text{Theta}_y); \\
\text{repeat} \\
\quad \{ \text{world_list is terminated by a nil node} \} \\
\text{with Temp_P := do} \\
\quad \begin{align*}
\text{begin} \\
\quad \text{T}_x := x; \text{T}_y := y; \text{T}_z := z; \\
\quad x := (T_x \times T_{11}) + (T_y \times T_{21}) + (T_z \times T_{31}); \\
\quad y := (T_x \times T_{12}) + (T_y \times T_{22}) + (T_z \times T_{32}); \\
\quad z := (T_x \times T_{13}) + (T_y \times T_{23}) + (T_z \times T_{33}); \\
\quad \text{Temp_P} := \text{F_Link} \\
\text{end} \\
\quad \text{until Temp_P = nil;} \\
\text{end} \\
\text{else}
begin
sound (540); delay (540); nosound;
S := 'No display file. Press any key to continue';
Clr_Scr_Gra (24,0,24,79); Disp_Line (S,1,25);
repeat until keypressed;
Clr_Scr_Gra (24,0,24,60)
end;
Disp_Line (Status,1,25)
end;  { Rotate }

begin  { Graphics }
mark (Heap_At_Strt);  { the very start of the heap }
HiRes; HiResColor (yellow);
S := 'Module : Graphics Display file = ';
Disp_Line (S,1,1);
S := 'Split 0 (raw View Rotate Quit';
Disp_Line (S,1,25);
Box_In_Gra (0,10,639,1891;  { center is 319 }
Comnds := ['S', 'D', 'V', 'R', 'Q'];
Exit := false;
World_List := nil; V_Center_x := 319; V_Center_y := 99; Disp_List := nil;
V_x := 390; V_y := 162.5; Sections := 0; Strm_Line := 0; Elec_Lenth := 0;
repeat
  repeat
    read (kbd, Key)
    until Upcase (Key) in Comnds;
  Case Upcase (Key) of
    'S' : Split_Screen (Active_Scrn, Comnds, S, V_Center_x, V_Center_y, V_x, V_y);
    'F' : Full_Scrn (Comnds, S, V_Center_x, V_Center_y, V_x, V_y);
    'C' : Change_Screen (Active_Scrn, V_Center_x, V_Center_y, V_x, V_y);
    'D' : Draw_Main_Proc (Disp_List, World_List, V_Center_x, V_Center_y, V_x, V_y, S,
      Elec_Lenth, Sections, Strm_Line, Heap_At_Strt, Heap_At_Eye);
    (* 'V' : View; *)  { future implementation } 
    'R' : Rotate (World_List, S);
    'Q' : Exit := true
  end;  { case }
  until Exit;
TextMode (C30)
end.  { Graphics }
APPENDIX C

- Table-1 BIOS-ROM interrupt 10H
- Table-2 DOS interrupt 21H

[for further information see reference 23]
### TABLE 1. VIDEO I/O OPERATIONS WITH TYPE 10 INTERRUPT.

<table>
<thead>
<tr>
<th>(AH)</th>
<th>Operation</th>
<th>Additional Input Register</th>
<th>Result Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRT Interface Routines</td>
<td>0 Set video mode</td>
<td>(AL) = 0 40x25 B/W, Alpha (Default)</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 1 40x25 Color, Alpha</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 2 80x25 B/W, Alpha</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 3 80x25 Color, Alpha</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 4 320x200 Color, Graphics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 5 320x200 B/W, Graphics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 6 640x200 B/W, Graphics</td>
<td></td>
</tr>
<tr>
<td>1 Set cursor lines</td>
<td></td>
<td>CH Bits 0-4 = Start line for cursor</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CH Bits 5-7 = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL Bits 0-4 = End line for cursor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL Bits 5-7 = 0</td>
<td></td>
</tr>
<tr>
<td>2 Set cursor position</td>
<td></td>
<td>(DH,DL) = Row, column (0,0) is upper left</td>
<td>None</td>
</tr>
<tr>
<td>3 Read cursor position</td>
<td></td>
<td>(BH) = Page number (0 for Graphics mode)</td>
<td>(DH,DL) = Row, column of cursor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(CH,CL) = Current cursor mode</td>
</tr>
<tr>
<td>4 Read light pen position</td>
<td></td>
<td>None</td>
<td>(AH) = 0 light pen switch not down or not triggered</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(AH) = 1 Valid pen values in registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(DH,DL) = Row, column</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(CH) = Raster line (0-199)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(BX) = Pixel column</td>
</tr>
</tbody>
</table>
5 Select active display Page (Alpha modes) 
   (AL) = New page value 
   (0-7 for Modes 0 and 1; 
   0-3 for Modes 2 and 3) 

6 Scroll active page up 
   (AL) = Number of lines 
   Input lines blanked at bottom of 
   window. (AL) = 0 blanks entire 
   window 
   (CH,CL) = Row, column of upper left corner 
   of scroll 
   (DH,DL) = Row, column of lower right corner 
   of scroll 
   (BH) = Attribute to be used on blank line 

7 Scroll active page down 
   (AL) = Number of lines 
   Input lines blanked at top of window 
   (AL) = 0 blanks entire window 
   (CH,CL) = Row, column of upper left corner of 
   scroll 
   (DH,DL) = Row, column of lower right corner of 
   scroll 
   (BH) = Attribute to be used on blank line
Character-Handling Routines

8 Read attribute/character at current cursor position
   (BH) = Display page (Alpha modes)  (AL) = Character read
   (AH) = Attribute of character read (Alpha modes)

9 Write attribute/character at current cursor position
   (BH) = Display page (Alpha modes)  (BL) = Attribute of character (Alpha)
   = Color of character (Graphics)  (CX) = Count of characters to write
   (AL) = Character to write

10 Write character only at current cursor position
   (BH) = Display page (Alpha modes)  None
   (CX) = Count of characters to write
   (AL) = Character to write

Graphics Interface

11 Set of color palette (320x200 graphics)
   (BH) = 10 of palette color (0 - 127)  None
   (BL) = Color value to be used with that color ID

12 Write dot
   (DX) = Row number
   (CX) = Column number
   (AL) = Color value
   If Bit 7 of AL = 1, the color value is exclusive ORed with the current contents of the dot

13 Read dot
   (DX) = Row number
   (CX) = Column number
   (AL) = Dot read

ASCII Teletype Routine for Output

14 Write character to screen then advance cursor
   (AL) = Character to write
   (BL) = Foreground color (Graphics)
   (BH) = Display page (Alpha)

15 Read current video state
   None
   (AL) = Current mode - see
   (AH) = 0 for explanation
   (AH) = Number of character columns on screen
   (BH) = Current active display page
<table>
<thead>
<tr>
<th>(AH)</th>
<th>Operation</th>
<th>Additional Input Registers</th>
<th>Result Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(AL) = Keyboard character</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(AL) = Keyboard character, if available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>= 0 if no character is available</td>
</tr>
<tr>
<td>1</td>
<td>Wait for keyboard character, then display it</td>
<td>None</td>
<td>(AL) = Keyboard character</td>
</tr>
<tr>
<td></td>
<td>(with Ctrl-Break check)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Display a character</td>
<td>(DL) = Display character</td>
<td>None</td>
</tr>
<tr>
<td>5</td>
<td>Print a character</td>
<td>(DL) = Print character</td>
<td>None</td>
</tr>
<tr>
<td>6</td>
<td>Read keyboard character</td>
<td>(DL) = OFFH</td>
<td>(AL) = Keyboard character, if available</td>
</tr>
<tr>
<td></td>
<td>(without Ctrl-Break check)</td>
<td></td>
<td>= 0 if no character is available</td>
</tr>
<tr>
<td>6</td>
<td>Display a character</td>
<td>(DL) = Display character</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>(value other than OFFH)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Wait for keyboard character, but do not display it</td>
<td>None</td>
<td>(AL) = Keyboard character</td>
</tr>
<tr>
<td></td>
<td>(without Ctrl-Break check)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Same as function 7, but with Ctrl-Break check</td>
<td>None</td>
<td>(AL) = Keyboard character</td>
</tr>
<tr>
<td>9</td>
<td>Display a string in memory</td>
<td>(DS:DX) = Address of string</td>
<td>None</td>
</tr>
<tr>
<td>A</td>
<td>Read keyboard characters into buffer</td>
<td>(DS:DX) = Address of buffer</td>
<td>None</td>
</tr>
<tr>
<td>B</td>
<td>Read keyboard status (with Ctrl-Break)</td>
<td>None</td>
<td>(AL) = OFFH if character is available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>= 0 if no character is available</td>
</tr>
<tr>
<td>C</td>
<td>Clear keyboard buffer and call a keyboard input function</td>
<td>(AL) = Keyboard function number (1,6,7,8, or A)</td>
<td>Per keyboard function</td>
</tr>
</tbody>
</table>

**Console functions**
Asynchronous Communications Functions

3 Wait for asynchronous input character
   None
   (AL) = Asynchronous character

4 Output a character to asynchronous device
   (DL) = Output character
   None

Disk Functions

0 Reset disk
   None
   None

E Select default drive
   (DL) = Drive number (0 = A, 1 = B)
   (AL) = Number of drives in system 2 for single drive system

F Open file
   (DS:DX) = Address unopened file control block (FCB)
   (AL) = 0 if file is found
   = OFFH if file is not found

10 Close file
   (DS:DX) = Address opened FCB
   Same as function F

11 Search for filename
   (DS:DX) = Address of unopened FCB
   (AL) = 0 if filename is found
   = OFFH if filename is not found

12 Find next occurrence of filename
   Same as function 11
   Same as function 11

13 Delete file
   Same as function 11
   Same as function 11

14 Read sequential file
   (DS:DX) = Address of opened FCB
   (AL) = 0 if transfer successful
   = 1 if no data in record
   = 2 if insufficient space
   = 3 if partial record is read

15 Write sequential file
   Same as function 14
   (AL) = 0 if transfer successful
   = 1 if disk is full
   = 2 if insufficient space
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Input/Output</th>
<th>Notes</th>
</tr>
</thead>
</table>
| 16 | Create a file | (DS:DX) = Address of unopened FCB | (AL) = $0$ if file is created  
= $\text{OFFH}$ if no entry is empty |
| 17 | Rename a file | (DS:DX) = Address of filename to be renamed  
(DS:DX+1H) = Address of new file name | (AL) = $0$ if rename successful  
= $\text{OFFH}$ if no match is found |
| 19 | Read default drive code | None | (AL) = Code of default drive ($0 = A$, $1 = B$) |
| 1A | Set disk transfer | (DS:DX) = Disk transfer address | None |
| 18 | Read allocation table address | None | (DS:DX) = Address of file allocation table  
(DX) = Number of allocation units  
(AL) = Records/allocation unit  
(CX) = Size of physical sector |
| 21 | Read random file | (DS:DX) = Address of opened FCB | Same as function 14 |
| 22 | Write random file | Same as function 21 | Same as function 15 |
| 23 | Set file size | (DS:DX) = Address of unopened FCB | (AL) = $0$ if file size is set  
= $\text{OFFH}$ if no matching entry is found |
| 24 | Set random record field | (DS:DX) = Address of opened FCB | None |
| 26 | Create a new program segment | (DX) = New segment number | None |
| 27 | Read random block | (DS:DX) = Address of opened FCB | (AL) = $0$ if transfer successful  
= $1$ if end-of-file  
= $2$ if wrap-around would occur  
= $3$ if last record is a partial record |
| 28 | Write random block | Same as function 27 | (AL) = $0$ if transfer successful  
= $1$ if insufficient space |
29  Parse a filename

(DS:SI) = Address of command line to parse
(ES:DI) = Address of memory to be filled with an unopened FCB
(AL) = 1 to scan off leading separators
      = 0 no scan-off

(AL) = 0 if parse successful
      = 1 if filename contains ? or *
      = OFFH if drive specifier is invalid

Date and Time Functions

2A  Get date

None

(CX) = Year (1980 - 2099)
(DH) = Month 1 - 12
(DL) = Day (1 - 31)

2B  Set date

(CX) and (DX) = Date, in same format as function 2A
(AL) = 0 if date is valid
      = OFFH if date is invalid

2C  Get time

None

(CH) = Hours (0 - 23)
(CL) = Minutes (0 - 59)
(DH) = Seconds (0 - 59)
(DL) = 1/100 seconds (0-99)

2D  Set time

(CX) and (DX) = Time, in same format as function 2C
(AL) = 0 if time is valid
      = OFFH if time is invalid

Missellaneous Functions

0  Terminate program

None

25  Set interrupt vector

(DS:DX) = Vector address
(AL) = interrupt type

None