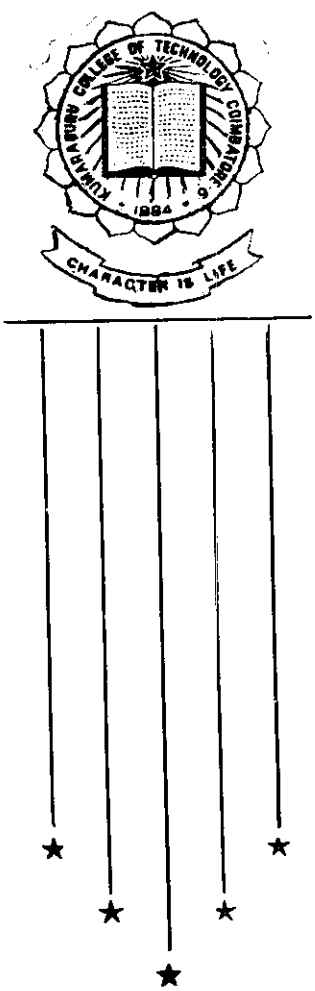
Viewing Transformations of a Six Sided Object



SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF ENGINEERING IN COMPUTER TECHNOLOGY AND INFORMATICS ENGINEERING

SUBMITTED BY

Sridharan .S
Elvin Mathew Varghese
Arul .S

UNDER THE GUIDANCE OF

Miss. G. Andal Jayalakshmi, B.E.

Department of Computer Technology and Informatics Engineering

Kumaraguru College of Technology

Coimbatore-641 006

1989-90

Department of Computer Technology & Informatics Engineering

Kumaraguru College of Technology

Coimbatore-641 006

Certificate

This is to certify that the project entitled

Viewing Transformations of a Six Sided Object

has been submitted by

Mr.....

In partial fulfilment for the Award of Bachelor of Engineering
in the Computer Technology & Informatics Engineering
Branch of the Bharathiar University, Coimbatore-641 006
During the Academic Year 1989-'90

.....
Guide

Head of the Department

Certified that the candidate was Examined by us in the project work

Viva-Voce Examination held on..... and the University

Register Number was.....

.....
Internal Examiner

.....
External Examiner

CONTENTS

P-102

ACKNOWLEDGEMENT

We would like to take this opportunity to thank our beloved **Principal, Prof. Palanivelu** for having allowed us to carry out this project to successful completion.

For the timely encouragement and support, we extend our heartfelt thanks to **Prof.P.Shanmugam,** Head of the Department, CTI and **Prof. M.Ramasamy,** Assistant Professor, Department of CTI.

For her untiring effort and never-ending enthusiasm, our very special thanks are due to our project guide, **Miss.G.Andal Jayalakshmi,** Department of CTI, without whose help this project would never have reached completion.

We owe a lot to **Dr.R.Prabhakar,** Head of the Department, CTI, Coimbatore Institute of Technology, for his speedy help and for his beautiful rendering of a matrix procedure, we'll always remember.

Finally we are indebted to all our college-mates as well as to students of various other colleges who happened to join us in our discussions especially our close friend **Sathyamoorthy.**

FOREWORD

Computer Graphics is one of the most exciting and rapidly growing fields in computer science. Some of the most sophisticated computer systems in use today are designed for the generation of graphics displays. We all know that value of a picture as an effective means for communication, and the ability to converse pictorially with a computer is revolutionizing the way computers are being used in all areas.

From the display of text to the Computer aided design of bridges, from business graphics to dazzling computer art compositions, from video games to flight simulators for jet aircraft: The field of computer graphics and its application presents new vistas and challenges to these who believe in the promise of the Computer as a tool for bearning, discovering, and creativity.

The study of Computer graphics calls for an understanding of Computer and display technologies and a solid grounding in two-and three-dimensional analytic geometry. This is to be combined with a knowledge of matrix operations, data structures, and algorithms.

In this Particular Project we have sought to construct various projections and views of some 3-dimensional object e.g. a cube using simple but powerful mathematical techniques as well as the dexterity of the 'C' programming language.

INTRODUCTION

1.1 DEFINITION

Computer Graphics is the creation, storage and manipulation of modules of objects and, their picture via computer, Interactive computer graphics is the important case in which the pictures' contents, format size or color on a display screen are dynamically controlled.

1.2 APPLICATIONS

Computer Graphics is used in many different areas of industry business, Government education, entertainmant and in advertising. The list of application is enormous and growing rapidly as simple display devices become routinely affordable Complex data in the computer are effectively communicated to user through computer graphics.

Graphics are most frequently used to draw two dimension (2D) or 3D graphics of mathematical, physical and economic functions, histograms, bar and pie charts, task scheduling charts, inventory, and production charts.

Computer Graphics is also concerned with the synthesis

pictures of real on imaginary objects, the related field of image processing treats the converse process, the analysis of scenes on reconstruction of two or three dimension objects from their picutres . Sub areas of image processing are called image enhancement, pattern detection and recognition on scene analysis and computer vision depending on the major objectives. These includes improving the image by eliminating 'noise' enhancing contrast, detecting and classifying standard patterns, finding deviations from standard patterns, or even recognizing three dimensional models on objects in a scene from several two dimensional image. The high resolution raster display makes interactive image processing to transformation of continuous tone images.

1.3 PICTURE CREATION

The picture is created by defining important points in the object and applying operation on the points like connection face formulation, surface formulation etc. The sealing parameters are applied to make the real system fit on the computer.

1.4 USER INTERACTION

Input devices such as keyboard tablet, joystick lightpen

and mouse are used to communicate to the system interactively. Output devices such as high resolution matrix printer, plotter, film recorder are extensively used to store graphical information permanently.

1.5 REASONS FOR CHOOSING THE LANGUAGE'C'

Among all the computer languages, dead and alive 'C' holds a unique place - people either love it or detest it distraction. There are no neutral parties here! 'C' has become defacto standards in many government and engineering fields, and it has gradually moved from minicomputers and mainframes into the computer arena as C PU have become more powerful and capable of supporting larger memories.

'C' offers access to the machine level with a rare blend of efficiency and elegance but sometimes the conciseness of the language encourages a cryptic cleverness that hinders maintainability.

The portability of 'C' programs stems from 'C's use of function libraries for such machine - dependent operation as I|O an area that bedeviled the growth of single standard pascal. Since 'C' is a small core - language unlike say, PL|1 or Ada there are surprisingly few keywords to learn. On the other hand

'C' is richer in operators than most languages. Particularly there are several operators that work at the bit level.

1.6 SUMMARY

In this introduction, definition of computer graphics, application on various fields have been dealt with. An introduction to designing objects, and the advantages of using 'C' has been given. Some of the interactive devices are also listed.

CHAPTER 2

OVERVIEW OF GRAPHICS SYSTEMS

2.1 DISPLAY DEVICES

Computer systems can be adopted to graphics applications in various ways, depending on the hardware and software capablities. Interactive systems use some type of video monitor as the primary output devices. The operation of most video monitors is based on the standard cathode Ray Tube (CRT) design, but several other technologies exist. The different display devices in use are: Refresh cathode Ray Tubes, Random-Scan and Raster-Scan Monitors, Color CRT Monitors Direct-view storage Tubes, Plasma-Panel Displays, LED and LED monitors, Laser Devices and Three - Dimensional Monitors.

2.2 HARD - COPY DEVICES

Many graphics systems are equipped to produce hard-copy output directly from a Video monitor in the form of 35 mm slides on overhead transparencies. Hard copy pictures can also be obtained by directing graphics output to a printer on plotter.

a. Printers

Printers produce output by either impact or nonimpact methods. Impact printers press formed character faces against an inked ribbon on to the paper. The familier line printer is an example of an impact device. Nonimpact printers are faster and quieter and often use a dot-matrix to print characters or draw lines.

b. Plotters

These devices produce hard - copy line drawings. The most commonly encountered plotters use ink pens to generate the drawings. The different types of plotters in use are: flatbed plotters, drum plotter and beltbed plotter.

2.3 DISPLAY PROCESSORS

Interactive graphics systems typically employ two or more processing units. In addition to the Central processing unit or CPU, a special purpose display processor is used to interact with the CPU and control the operation of the display devices.

With refresh CRT systems, the display processor may also be required to cycle through the picture definition, refreshing the screen often enough to eliminate flicker. The picture definition is kept in a refresh storage area. On many systems this screen - refreshing task can be assigned to an additional processor called the display controller. This allows the display processor to devote full time to other functions. Different types of display processor systems in use are Random-Scan systems DVST systems and Raster scan systems.

2.4 GRAPHICS SOFTWARE

Programming commands for displaying and manipulating graphics output are designed as extensions to existing languages.

a. Coordinate Represenations

Most graphics packages are designed to use cartesian coordinate systems. The Coordinates referenced by a user are called works coordinates, and the Coordinates used by a particular output device are called device coordinates or Screen Coordinates in the case of video monitor.

b. Graphics Functions

A general purpose graphics package provides users with a variety of functions for creating and manipulating pictures. These routines can be categorized according to whether they deal with output, input, attributes, segment transformations, viewing or general control.

2.5 GEOMETRICAL TRANSFORMATIONS

Geometrical transformations are important in graphic to change the positions, size and view. Depending on the dimension in which it is applied it is called as 2D transformation or 3D transformation. Transformations-translation rotation and scaling are used to move a point from one place to another. Scaling is applied to scale up or scale down object size by scaling point coordination. Rotation is used to rotate a point about a particular point or axis.

2.5.1. 2D Transformation

When an operation is made on a print which has two axes, then the transformation is called 2D transformation. The translations are discussed in the following sections.

Translation

Movement of x and y points by tx and ty respectively are

$$x' = x + tx$$

$$y' = y + ty$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \ldots(2.1)$$

Translation of figure 2.1 with tx = 40 and ty = 40 is shown in Figure 2.2

Rotation

Rotation of a point x, y to Θ degrees are

$$x' = x \cos - y \sin \Theta$$

$$y' = x \sin + y \cos \Theta$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Cos } \theta & -\sin \theta & 0 \\ \text{Sin } \theta & \text{Cos } \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \ldots(2.2)$$

Rotation of Figure 2.1 with θ = 30 degrees is shown in Figure 2.3.

Scaling

Scaling of a point x, y by Sx and Sy respectively are

$$x' = x\ Sx$$

$$y' = y\ Sy$$

$$\begin{bmatrix} x' \\ y' \\ \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \ldots(2.3)$$

Scaling of Figure. 2.1 with Sx = $\frac{1}{2}$ and Sy = $\frac{1}{2}$ is shown in Figure 2.4.

2.5.2 3D Transformation

When an operation is made on a point which has 3 coordinates then transformation is called 3D transformation.

Translation

Movement of x, y and z points by tx, ty and tz respectively are

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \ldots(2.4)$$

Translation of object in Figure 2.5 with tx = 5, ty = 10 is shown in Figure 2.6.

Rotation

Notation of a point x, y and z about each axis is calculated separately. The matrix representation is as follows :

About z axis :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Cos}\,\theta & -\text{Sin}\,\theta & 0 & 0 \\ \text{Sin}\,\theta & \text{Cos}\,\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \ldots(2.5)$$

About y axis :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Cos}\,\theta & 0 & \text{Sin}\,\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\text{Sin}\,\theta & 0 & \text{Cos}\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \ldots(2.6)$$

About z axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \text{Cos}\,\theta & -\text{Sin}\,\theta & 0 \\ 0 & \text{Sin}\,\theta & \text{Cos}\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \ldots(2.7)$$

ROTATION ABOUT AN ARBITRARY AXIS L DEFINED BY A DIRECTION VECTOR V AND A LOCATION POINT P :

The required transformation can be found by the following steps :

1. Translate P to the Origin.
2. Align V with the Vector k.
3. Rotate by about k.
4. Reverse steps 2 and 1.

So,

$$R_{\theta.L} = T_{-P}^{-1} \cdot A_V^{-1} \cdot R_{,k} \cdot A_V \cdot T_{-p} \qquad \ldots (2.8)$$

where $R_{,k}$ and T_{-p} refer to Rotation and Translation matrics defined earlier. Here A_V is the alignment transformation which is described in the following section :

TRANSFORMATION A_V WHICH ALIGNS A GIVEN VECTOR V WITH THE VECTOR k ALONG THE POSITIVE z - AXIS

Let $V = aI + bJ + cK$

Alignment is performed through the following sequence of transformations.

Refer Fig. 2.7, 2.8, and 2.9.

1. Rotate about the x axis by an angle Θ_1 so that V rotates into the upper half of the xz plane (as the vector V_1).

2. Rotate the vector V, about the y axis by an angle $-\Theta_2$ so that V_1 rotates to the positive z axis (as the vector V_2).

Implementing Step 1 from Figure 2.8, the required angle of rotation Θ, can be found by looking at the projection of V onto the yz plane. (by assuming b and c are not both zero). From triangle OP 'B'.

$$\text{Sin } \Theta_1 = \frac{b}{(b^2 + c^2)^{\frac{1}{2}}} \qquad \text{Cos } \Theta_1 = \frac{c}{(b^2 + c^2)^{\frac{1}{2}}}$$

The required rotation is

$$R\,\Theta_1\,I = \begin{bmatrix} 1 & 0 & 0 & \\ 0 & \frac{c}{(b^2+c^2)^{\frac{1}{2}}} & \frac{-b}{(b^2+c^2)^{\frac{1}{2}}} & \\ 0 & \frac{b}{(b^2+c^2)^{\frac{1}{2}}} & \frac{c}{(b^2+c^2)^{\frac{1}{2}}} & \\ 0 & 0 & 0 & \end{bmatrix}$$

Applying this rotation to the vector $\vec{V}$ produces the vector V_1 with the components $(a, 0, (b^2 - c^2)^{\frac{1}{2}})$.

Implementing step 2 from Figure 2.9, we see that a rotation of $-\Theta_2$ degrees is required, and so from triangle QQQ', we get

$$\text{Sin}\ (-\Theta_2) = -\text{Sin}\ _2 = -\frac{a}{(a^2+b^2+c^2)^{\frac{1}{2}}} \text{ and}$$

$$\text{Cos}\ (-\Theta_2) = \frac{(b^2 + c^2)^{\frac{1}{2}}}{(a^2+b^2+c^2)^{\frac{1}{2}}}$$

$$R_{-\Theta_2}'J = \begin{bmatrix} \frac{(b^2+c^2)^{\frac{1}{2}}}{(a^2+b^2+c^2)^{\frac{1}{2}}} & 0 & \frac{-a}{(a^2+b^2+c^2)^{\frac{1}{2}}} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{a}{(a^2+b^2+c^2)^{\frac{1}{2}}} & 0 & \frac{(b^2+c^2)}{(a^2+b^2+c^2)^{\frac{1}{2}}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since $|V| = (a^2 + b^2 + c^2)^{\frac{1}{2}}$ and introducing the notation $\lambda = (b^2 + c^2)^{\frac{1}{2}}$, we find

$$A_V = R_{-\Theta_2,J} \cdot R_{\Theta_1,I}$$

$$A_V = \begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{\lambda|V|} & \frac{-ac}{|V|} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \ldots(2.9)$$

If both b and c are zero, then V = aI, and So = 0. In this case, only a ± 90° rotation about the y axis is required. So if Θ = 0, it follows that

$$A_V = \begin{bmatrix} 0 & 0 & \frac{-a}{|a|} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{a}{|a|} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \ldots(2.10)$$

In the same manner we calculate the inverse transformation that aligns the vector k with the vector V.

$$A_V^{-1} = (R_{-\Theta_2,\ J}\ R_{\Theta_1,\ I}) = R^{-1}_{\Theta_1,\ I}\ R^{-1}_{-\Theta_2,\ J}$$

$$A_V^{-1} = \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{|V|} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{\lambda|V|} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \ldots(2.12)$$

Rotation of object in Figure 2.5 with = 40° is shown in Figure 2.6(b).

Scaling

Scaling of a point x, y and z by Sx, Sy and Sz respectively are

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \ldots(2.12)$$

Scaling of the object in Figure 2.5 with Sx = 0, Sy = 0.5 and Sz = 0.5 is shown in figure 2.6b.

2.6 SUMMARY

On the whole the choice of a particular graphics system would depend on the job at hand.

CHAPTER - 3

PACKAGE OUTLINE

3.1 INTRODUCTION

This package was developed on a LAN (Local Area Network) environment using a single colour terminal with a built-in CGA board. The network was centered around an MS-DOS Operating System and enhanced using interplacing made possible with the help of a TURBO Editor (TURBO-C Version 2.0). This package which takes up40KB of memory has a built-in user friendly Menu that allows fast and efficient execution of various modules.

3.2 PURPOSE

This package aims at enabling the user to perceive various views of some six sided object based upon his position with respect to some reference origin and in relation to some user specified view plane. Various views are made possible and any one view may be seen at a time. Of the possible views two broad categories are the perspective and parallel projections (Refer Table 3.1).

Among the parallel projections, the most common views are the isometric, the plan, the elevation and the side view.

Hence these have been assigned a distinctive portion (tablet) of the display menu. The perspective projection can be of the 1 vanishing point, 2 vanishing point or 3 vanishing point type, the choice of which is left to the discretion of the user and which will further be based upon his specification of view plane and view point.

Less commonly used views such as Dimetric, Trimetric (axonometric), cavalier, cabinet (oblique) can also be seen but depend to a large extent on the user's knowledge of their categorial placing within the spectrum and parallel projections.

Rotation, Translation and Scaling, in 3-dimensions i.e., before projecting onto a view plane, are also possible and each of these transformations have been allotted a specific tablet on the Menu.

3.3 ATTRIBUTES

a. The elimination of hidden lines

The need for eliminating hidden lines, edges, surfaces or volumes is illustrated in figure 3.2. Figure 3.2 a shows a typical wire frame drawing of a cube. A wire frame drawing represents a three dimensional object as a line drawing of its edges. Figure 3.2a can be interpreted either as a view of the

cube from above and to the left or from below and to the right. The alternate views can be seen by blinking and refocusing the eyes. This ambiguity can be eliminated by removing the lines or surfaces that are invisible from the two alternate view points. The results are shown in Figure 3.2b and 3.2c. The complexity of the hidden line/hidden surface problem has resulted in a large number of diverse solutions. Many of these are for specialized applications. There is no best solution to the hidden line/hidden surface problem.

Hidden line/Hidden surface algorithms can be classified based on the coordinate system or space in which they are implemented. Object space algorithms are implemented in the physical coordinate system in which the objects are described. Very precise results, generally to the precision of the machine are available. These results can be satisfactorily enlarged many times. Object space alogirthms are particularly useful in precise engineering applications. Image space algorithms are implemented in the screen coordinate system in which the objects are viewed. Calculations are performed only to the precision of the screen representation. This is generally very crude, typically 512 x 512 integer points. Scenes calculated in image space and significantly enlarged do not give acceptable results. E.g. the end points of lines may not match. List priority algorithms are partially implemented in both coordinate systems.

Theoretically, the computational work for an object space algorithm that compares every object in a scene grows as the number of objects squared (n^2). Similarly the work for an image space algorithm which compares every object in the scene with every pixel location in screen coordinates theoretically grows as nN. Here, n is the number of objects (volumes, planes or edges) in the scene, and N is the number of pixels. Theoretically, object space algorithms require less work than image space algorithms for n N. Since N is typically $(512)^2$, most algorithms should theoretically be implemented in object space. In practice, this is not the case, image space algorithms are more efficient because it is easier to take advantage of coherence in a raster scan implementation of an image space algorithm.

Since the object in question is a solitary one, an object space hidden line algorithm was implemented in this particular package.

3.3.a. Color

Color is both a psychophysiological phenomenon and a psychophysical phenomenon. The perception of color depends upon the physics of light considered as electromagnetic energy and its interaction with physical materials, and on the interpretation of the resulting phenomena by the human eye-brain visual

system. The human visual system interprets electromagnetic energy with wave lengths between approximately 400 and 700 nanometers as visible light. A nanometer (nm) is 10^{-9} meter or a billionth of a meter. Light is perceived either directly from a source of illumination, eg. a light bulb or indirectly by reflection from the surface of an object or refraction through an object.

When perceived light contains all the visible wave lengths with approximately equal weights, the light source or object is achromatic. An achromatic light source appears white when the reflected or transmitted light from an object is achromatic, it appears white, black, or an intermediate level or shade of gray. Objects that achromatically reflect more than about 80% of the incident light from a white light source appears white. Those that achromatically reflect less than about 3% of the incident light appear black. Intermediate achromatic reflectance levels yield various shades of gray. It is convenient to consider the intensity of the reflected light in a range between 0 and 1, with 0 equated to black and 1 to white. Intermediate values are gray.

3.4 SUMMARY

Although the package developed has a restricted set of attributes, it affords a wide variety of viewing transforms with minimum complexity involved. Hence the overall simplicity allows fast and easy access.

CHAPTER - 4

DESIGN AND REPRESENTATION OF OBJECTS IN 3D

4.1 INTRODUCTION

Before analysing an object using the computer, it should be represented in the form of data understandable by computer. Geometric transformations like translation, scaling are the heart of many graphical systems. Hence, the important geometrical properties of the object should be incorporated in the designing package, such that the object can be viewed from different directions with different transformations. Designing of 3D objects are difficult compared to 2D because of more points and more faces involved, perspective transformation, visible and hidden surface manipulations and more storage requirements in 3D. The coming sections describe the necessary theoretical aspects, which is useful to the design of software.

4.2 THE VARIOUS KINDS OF PROJECTION

We can construct different projections according to the view that is desired. Table 3.1 provides a taxonomy of the families of perspective and parallel projections.

4.2.1. Perspective Projection

a. Basic Principles

The techniques of perspective projection are generalizations of the principles used by artists in preparing perspective drawings of 3-dimensional objects and scenes. The eye of the artist is placed at the centre of projection, and the canvas, or more precisely the plane containing the canvas, becomes the view plane. An image point is located at the intersection of a projector (a ray drawn from an object point to the centre of projection) with the view plane. (Refer Figure 4.1).

Perspective drawings are characterized by perspective foreshortening and vanishing points. Perspective foreshortening is the illusion that objects and lengths appear smaller as their distance from the centre of projection increases. The illusion that certain sets of parallel lines appear to meet at a point is another feature of perspective drawings. These points are called vanishing points. Principal vanishing points are formed by the apparent intersection of lines parallel to one of the three principal x, y or z axes. The number of principal vanishing points is determined by the number of principal axes intersected by the view plane.

b. Mathematical Description of a Perspective Projection

A perspective transformation is determined by prescribing a centre of projection and a view plane. The view plane is determined by its view reference point R_o and view plane Normal N. The object point P is located in world coordinates at (x, y, z). The problem is to determine the image point ccordinates P(x', y', z'). Refer Figure 4.1.

Example

The standard perspective projection is shown in Figure 4.2. Here the view plane is the xy plane, and the centre of projection is taken as the point C(0, o, -d) on the negative z axis.

Using similar triangles ABC and A'OC, we find

$$x' = \frac{d.x}{z+d} \qquad y' = \frac{d.y}{z+d} \qquad z' = 0$$

The perspective transformation between object and image point is non linear and so cannot be represented as 3 x 3 matrix transformation. However if we use homogeneous coordinates, the perspective transformation can be represented as a 4 x 4 matrix.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} d.x \\ d.y \\ 0 \\ z+d \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

4.2.2. Perspective Anomalies

The process of constructing a perspective view introduces certain anomalies which enhance realism in terms of depth axes but also distort actual sizes and shapes.

1. Perspective foreshortening

The farther an object is from the centre of projection, the smaller it appears (ie. its projected size becomes smaller). Refer Figure 4.3.

2. Vanishing points

Projections of lines that are not parallel to the view plane (i.e. lines that are not perpendicular to the view plane normal) appear to meet at some point on the view plane. A common manifestation of this anomaly is the illusion that rail road tracks meet at a point on the horizon.

3. View Confusion

Objects behind the centre of projection are projected upside down and backward onto the view plane. Refer Figure 4.5.

4. Topological Distortion

Consider the plane that passes through the centre of projection and is parallel to the view plane. The points of this plane are projected to infinity by the perspective transformation. In particular, a finite line segment joining a point which lies in front of the viewer to a point in back of the viewer is actually projected to a broken line of infinite extent. Refer Figure 4.6.

Example

For the standard projection, the projections L_1' and L_2' of parallel lines L_1 and L_2 having the direction of the vector K appear to meet at the origin. Refer Figure 4.4.

4.2.3. Parallel Projection

Basic Principles

Parallel projection methods are used by drafters and engineers to create working drawings of an object which preserve

its scale and shape. The complete representation of these details often requires two or more view (projections) of the object onto different view planes.

In parallel projection, image points are found as the intersection of the view plane with a projector drawn from the object point and having a fixed direction. Refer Figure 4.7. The direction of projection is the prescribed direction for all projectors. Orthographic projections are characterized by the fact that the direction of projection is perpendicular to the view plane. When the direction of projection is parallel to any one the principal axes, this produces the front, top and side views of mechanical drawings (also referred to as multiview drawings). Axonometric projections are orthographic projections in which the direction of projection is not parallel to any of the three principal axes. Nonorthographic parallel projections are called oblique parallel projections. Refer Figure 4.8.

4.2.4. Mathematical Description of a Parallel Projection

A parallel projective transformation is determined by prescribing a direction of projection vector V and a view plane. The view plane is specified by its view plane reference point Ro, a view plane normal N. The object point P is located at (x, y, z) in world coordinates. The problem is to determine

the image point coordinates P_1 (x', y', z'). Refer Figure 4.7.

If the projection vector V has the direction of the view plane normal N, the projection is said to be Orthographic. Otherwise it is called oblique. Refer Figure 4.8.

4.2.5. Some Common Subcategories of Orthographic Projections are

1. Isometric - the direction of projection makes equal angles with all of the three principal axes.

2. Dimetric - the direction of projection makes equal angles with exactly two of the principal axes.

3. Trimetric - the direction of projection makes unequal angles with the three principal axes.

Some Common Subcategories of Oblique Projections are

1. Cavalier - the direction of projection is chosen so that there is no foreshortening of lines perpendicular to the xy plane.

2. Cabinet - the direction of projection is chosen so that lines perpendicular to the xy planes are foreshortened by half their lengths.

4.3 GENERALIZED PROCEDURE FOR PERSPECTIVE TRANSFORMATION

The perspective projection onto the view plane z = d where the center of projection is the origin $(o_1 o_1 o_1)$. The projection Matrix is

$$Per_{K,Ro} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$ (Refer Figure 4.9)

We can conclude that the vectors $\overline{PC}$ and $\overline{P'C}$ (refer Fig. 4.10) satisfy

$\overline{P'C} = \overline{PC}$ then

$x' = (x - a) + a$

$y' = (x - b) + b$

$z' = (z - c) + c$

Also, we find (by using the equation of the view plane) that

$$\underline{\alpha} = \frac{\alpha}{n_1 (x - a) + n_2 (y - b) + n_3 (z - c)}$$

[ie., p' (x', y', z') is on the view plane and thus satisfier the view plane equation $n_1(x'-x_o) + n_2(y'-y_o) + n_3 (z'-z_o) = 0$]

Here, $d = (n_1x_o + n_2y_o + n_3z_o) - (n_1a + n_2b + n_3c)$

d is proportional to the distance D from the view plane to the centre of projection, that is $d = \pm|N|D$

TO FIND THE HOMOGENEOUS COORDINATE MATRIX REPRESENTATION IT IS EASIEST TO PROCEED AS FOLLOWS :

1. Translate so that the centre of projection C lines at the origin. Now $R'_o = (x_0-a, y_o-b, z_o-c)$ becomes the reference point of the translated plane (the normal vector is unchanged by translation).

2. Project onto the translated plane using the origin as the center of projection by constructing the transformation Per_{N,R'_o}

3. Translate back.

Introducing the intermediate quantities

$$d = n_1 x_o + n_2 y_o + n_3 z_o$$

and $$d_1 = n_1 a + n_2 b + n_3 c$$

We obtain $d = d_o - d_1$, and so $Per_{N,R_o,C} = T_C \cdot Per_{N,R_o} \cdot T_{-C}$

Then with R'_o used as the reference point in constructing the projection $P_{N'}$ R'_o

$$Per_{N,R_o,C} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ n_1 & n_2 & n_3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} d+an_1 & an_2 & an_3 & -ad_o \\ bn_1 & d+bn_2 & bn_3 & -bd_o \\ cn_1 & cn_2 & d+cn_3 & -cd_o \\ n_1 & n_2 & n_3 & -d_1 \end{bmatrix}$$

4.4 GENERALIZED PROCEDURE FOR PARALLEL PROJECTION

The intersection point of a line segment determined by two points. $P_1(x_1,y_1,z_1)$ and $P_2(x_2,y_2,z_2)$ and a given plane is given.

Let $N = n_1I + n_2J + n_3K$ be the normal vector to the plane and $P_o(x_0, y_o, z_o)$ a point on the plane. The equation of the plane is then

$$n_1(x-x_o) + n_2(y-y_o) + n_3(z-z_o) = 0$$

The equation of the line determined by P_1 and P_2 is

$$x = x_1 + (x_2 - x_1)t$$

$$y = y_1 + (y_2 - y_1)t$$

$$z = z_1 + (z_2 - z_1)t$$

Substituting there equations into the equation of the plane, we have

$$n_1[x_2 + (x_2 - x_1)t - x_o] + n_2[y_1 + (y_2 - y_1)t - y_o]$$

$$+ n_3[z_1 + (z_2 - z_1)t - z_o] = 0$$

Solving for t, we find

$$t = \frac{-n_1 (x_1 - x_o) + n_2(y_1 - y_o) + n_3(z_1 - z_o)}{n_1(x_2 - x_1) + n_2(y_2 - y_1) + n_3(z_2 - z_1)}$$

Calling this number t_I, we find that the intersection point

$P_I(x_I, y_1, z_1)$ is

$$x_I = x_1 + (x_2 - x_1)t_I$$

$$y_I = y_1 + (y_2 - y_1)t_I$$

$$z_I = z_1 + (z_2 - z_1)t_I$$

If $0 \leq t_I \leq 1$ the intersection point lies on the line segment between P_1 and P_2. Otherwise, the intersection is on the extended line.

Note that the number t_I can be written in vector form as

$$t_I = - \frac{N.\overline{P_o P_1}}{N.\overline{P_1 P_2}}$$

4.5 VIEW PLANE TRANSFORMATION

4.5.1. Three-Dimensional Viewing

Three-dimensional viewing of an object on picture requires the specification of a projection plane (called the view plane) and a center of projection (or view point).

4.5.2. Specifying the Projection Plane

The projection plane is specified by prescribing

1. a reference point $R_o(x_o, y_o, z_o)$ in world coordinates and
2. a Unit normal vector $N = n_1I + n_2J + n_3K$, $|N| = 1$, to the projection plane

Refer Figure 4.11

From this information, we can construct the projections used in presenting the required view with respect to the given view point.

4.5.3. View Plane Coordinates

The view plane coordinate system Can be specified as follows:

1. Let the reference point R_o (x_o, y_o, z_o) be the origin of the coordinate system and

2. Determine the coordinate axis.

To do this, we first choose a reference vector $\vec{U}$ called the up vecture $\vec{J}_q$ can then be determined by the projection of the Vector $\vec{U}$ onto the view plane. We let the vector $\vec{J}_q$ define the direction of the positive q axis for the view plane coordinate system. To calculate $\vec{J}_q$ we proceed as follows : with N being the view plane unit normal vector.

let $\vec{U}_q = \vec{U} - (N.U)\,\vec{N}$. Then

$$J_q = \frac{\vec{U}_q}{|\vec{U}_q|}$$

is the unit vector that defines the direction of the positive q axis.

Refer Figure 4.12

Finally, the direction vector $\vec{I}_p$ of the positive P axis is chosen so that it is perpendicular to $\vec{J}_q$, and by convention, so that the trial $\vec{I}_p$, $\vec{J}_q$ and $\vec{N}$ forms a left-handed coordinate system. That is

$$\vec{I}_p = \frac{\vec{N} \times \vec{J}_q}{|\vec{N} \times \vec{J}_q|}$$

The coordinate system is called the view plane coordinate system. This is shown in Figure 4.13.

Right-handed world coordinates (x, y, z) can be changed to left-handed view-plane coordinates (x', y', z') by performing the transformation : (Refer Figure 4.13b).

$$T_{RL} : \quad x' = x \qquad y' = y \qquad z' = z$$

In matrix form, for homogeneous coordinates:

$$T_{RL} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The general transformation for changing from world to view plate coordinates is developed in the following section and it is shown in Figure 4.13.

The world coordinate axis are determined by the right handed trial of unit vectors $[\vec{I}, \vec{J}, \vec{K}]$.

The view plane coordinate axis are determined by the left-handed trial of vector $(\vec{I}_p, \vec{J}_q, \vec{N})$ and the view reference point R_o (x_o, y_o, z_o).

Referring to Figure 4.13 we construct the transformation two through the concatenation of the matrix determined by the following steps :

1. Translate the view plant reference point R_o (x_o, y_o, z_o) to the world coordinate origin via the translation matrix T_v. Here V is the vector with components $-x_o\vec{I}-y_o\vec{J}-z_o\vec{K}$.

2. Align the view plane normal N with the vector -K (the direction of the negative z axis) using the transformation A_N-K.

$$A_{N,\,-\,k} = A_{-k}^{-1} \cdot A_N$$

Let I_p' be the new position of the vector I_p after performing the alignment transformation i.e.

$$\vec{I}_p' = A_n - K \cdot \vec{I}_p$$

3. Rotate $\vec{I}_p'$ about the z axis so that it aligns with $\vec{I}$, the direction of the x axis. With being the angle between $\vec{I}_p'$ and $\vec{I}$, the rotation is $R\,\Theta_{,k}$.

4. Change from the right-handed coordinates to left-handed coordinates by applying the transformation T_{RL}. Then T_{wv} = T_{RL}, R, k, A_{N-K}, T_v. If (x_w, y_w, z_w) are world coordinates (x_v, y_v, z_v) of P can be found by applying the transformation T_{wv}.

4.6 HIDDEN LINE REMOVAL

4.6.1 Introduction

The Roberts Algorithm represents the first known solution to the hidden line problem. It is a mathematically Elegent solution which operates in object space. The algorithm eliminates the edges or planes from the volume that are hidden by the volume itself.

4.6.2. Mathematics behind the Roberts Algorithm

The Roberts algorithm requires that all volumes or objects in a picture be convex. Concave volumes must be subdivided into component convex volumes. The algorithm considers a convex planar polygonal volume to be represented by a collection of intersecting planes. The equation of a plane in 3 space is D.

$$ax + by + cz + d = 0$$

In matrix notation the result is

$$[x \quad y \quad z \quad 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0$$

$$\text{or } [x \quad y \quad z \quad 1] \quad [p]^T = 0$$

Where $[P]^T = [a \ b \ c \ d]$ represents the plane. A convex solid can thus be represented by a volume matrix of plane equation coefficients e.g.

$$[V] = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \\ b_1 & b_2 & \cdots & b_n \\ c_1 & c_2 & \cdots & c_n \\ d_1 & d_2 & \cdots & d_n \end{bmatrix}$$

where each column represents the coefficients of single plane.

If [S] (a general point in the space) is one the plane then [S] . [P] = 0. If [S] is not on the plane, the sign of the dot product indicates which side it is on. The Roberts Algorithm uses the convention that points on the side of a plane corresponding to the inside of a volume yield positive dot products.

A technique due to Martin Newell gives both an exact solution for the plane equation for planar polygons and a "best" approximation for almost planar polygons If a, b, c, d are the coefficients of the plane equation, then

$$a = \sum_{i=1}^{n} (y_i - y_j)(z_i + z_j)$$

$$b = \sum_{i=1}^{n} (z_i - z_j)(x_i + x_j)$$

$$c = \sum_{i=1}^{n} (x_i - x_j)(y_i + y_j)$$

where

if i = n then j = 1 else j = i + 1 and

d is obtained using any point in the plane.

Selecting the Test Point

If the view or eye point is at infinity on the positive z axis looking towards the origin, then the view direction is towards negative infinity on the z axis. In homogeneous coordinates this vector is represented by

$$[E] = [0 \quad 0 \quad -1 \quad 0]$$

[E] also represents the point at infinity on the negative z axis. In fact [E] represents any point on the plane at $z = \infty$ ie., any point $(x, y, -\infty)$. Thus, if the dot product of (E) and the plane in the volume matrix is negative, then (E) is outside these planes. Consequently, these planes are hidden with respect to a view point any where on the plane at $z = \infty$, and the test

point at Z = is hidden by the volume itself. This is shown in figure 4.14.

These hidden planes are cllaed self-hidden planes or back faces. Hence,

$$[E] \cdot [V] \quad 0$$

identifies self-hidden planes or backfaces. For axonometic projections (eye point at infinity) that is equivalent looking for positive values in the 3rd row of the volume matrix.

An efficient implementation of the Roberts Algorithm is given below :

An algorithm assumes that a volume consists of polygonal planar faces, the faces consist of edges, and the edges consist of individual vertics. All vertices, edges, and faces are associated with a specific volume.

4.6.3. Algorithm

To eliminate the self-hidden planes.

For each volume in the scene.
Calculate the plane equation for each face polygon of the volume.

Check the sign of the plane equation.

Calculate a point inside the volume as the average of the vetrices.

Calculate the dot product of the plane equation and the point inside the volume.

If the dot product is 0 change the sign of theplane equation.

Form the volume matrix.

Identify the self-hidden planes.

Take the dot product of the test point at infinity and the transformed volume matrix.

If the dot product is 0, then the plane is hidden. Eliminate the entire polygon forming the plane. This eliminates the necessity for separately identifying hidden lines as the intersection of two hidden planes.

4.7 STORAGE STRUCTURE

A two dimensional array, M(4 x 8) has been chosen as the storage structure for the object's vertex co-ordinates where each column contains the x, y and z vlaues of the vertices. The columns of the matrix M are ordered as shown in Figure 4.21.

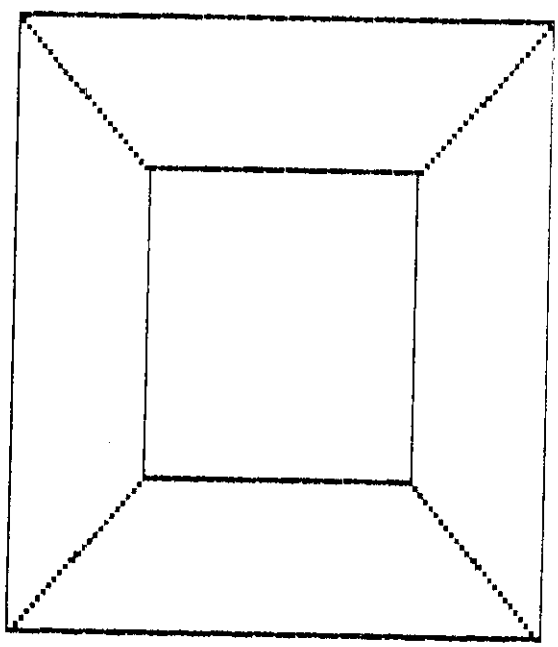
Surface allocation was carried out by assigning appropriate column of the matrix M to a matrix Sul (6 x 12) based on a predefined allocation procedure.

TRANSLATE
ROTATE
SCALE
PARALLEL
PLAN
ELEVATION
SIDEVIEW
PERSPECTIVE
ISOMETRIC

VARIOUS VIEWS OF A TRUNCATED, SQUARE BASED PYRAMID

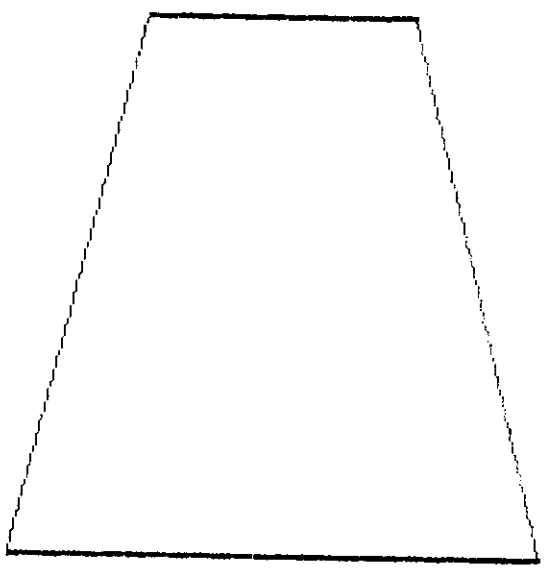
```
Enter the value of x1 y1 z1
3 3 11
Enter the value of x2 y2 z2
5 10 9
Enter the value of x3 y3 z3
9 10 9
Enter the value of x4 y4 z4
11 3 11
Enter the value of x5 y5 z5
3 3 3
Enter the value of x6 y6 z6
5 10 5
Enter the value of x7 y7 z7
9 10 5
Enter the value of x8 y8 z8
11 3 3
```

```
Specify your  up vector
0 0 -1
```



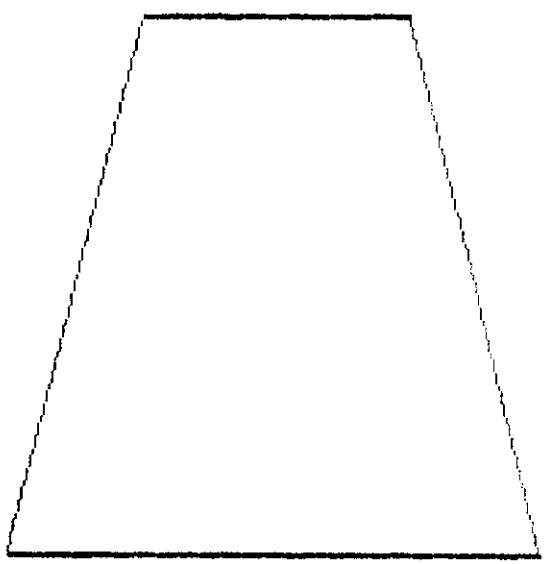
PLAN

```
Specify your up vector
0 1 0
```



ELEVATION

```
Specify your up vector
0 1 0
```

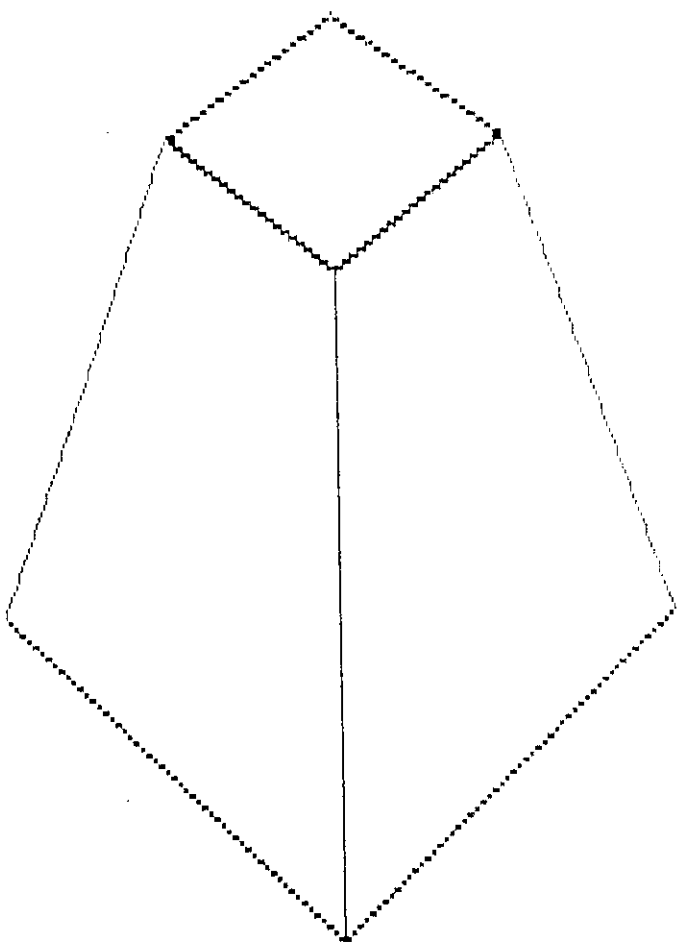


SIDE VIEW

```
Do you want to change the co-ordinates of the vi plane('Y') or ('N')
0.000000   0.000000   20.000000   20.000000   20.000000   0.000000   20.00000
 0.000000   0.000000
Enter the values of view points x1,y1,z1
50 50 50
```

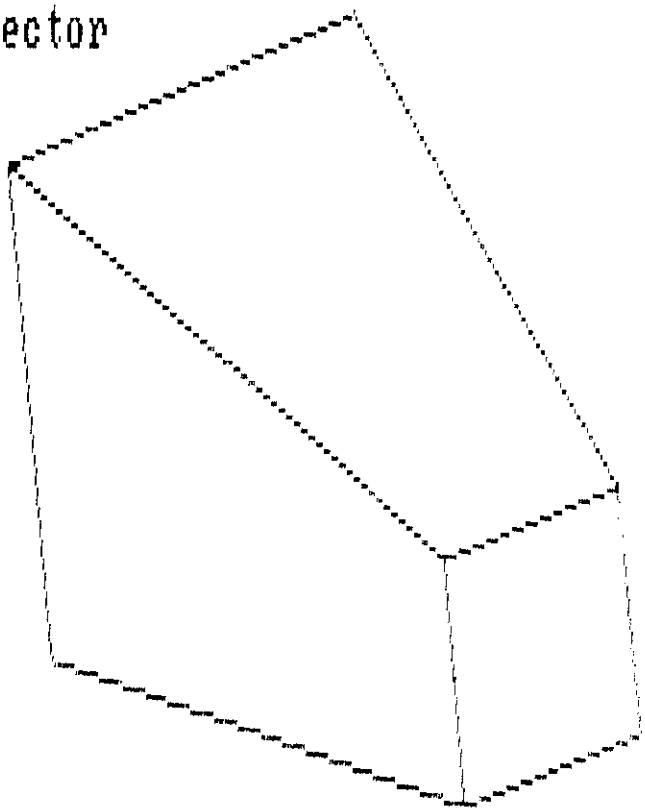
SPECIFY YOUR UP VECTOR

0 1 0



PERSPECTIVE VIEW

```
Specify your up vector
2 .8 1
```

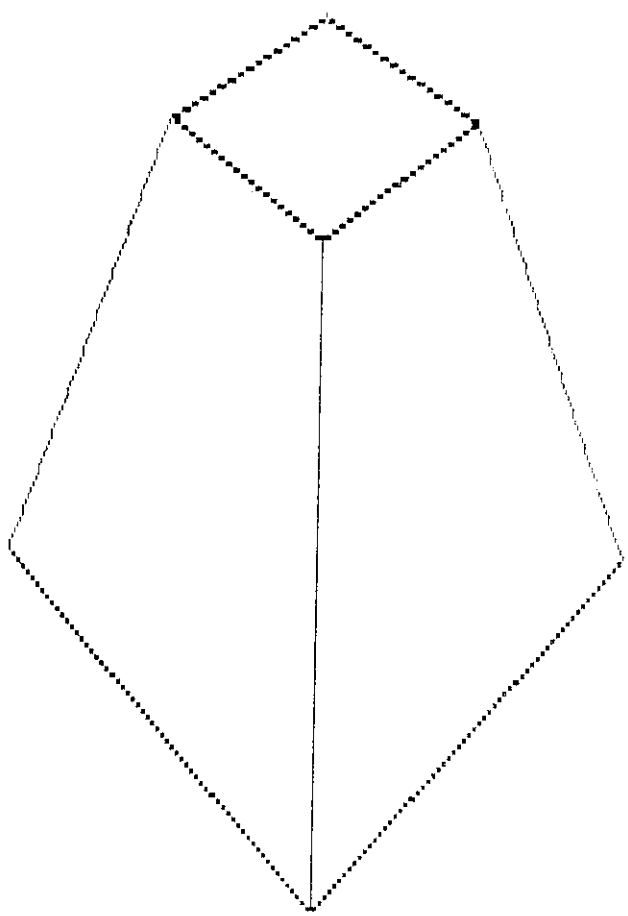


ISOMETRIC VIEW

```
Specify your translation factors
5 10 5


Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000    0.000000    20.000000    20.000000    20.000000    0.000000    20.00000
 0.000000    0.000000
Enter the values of view points x1,y1,z1
50 50 50
```

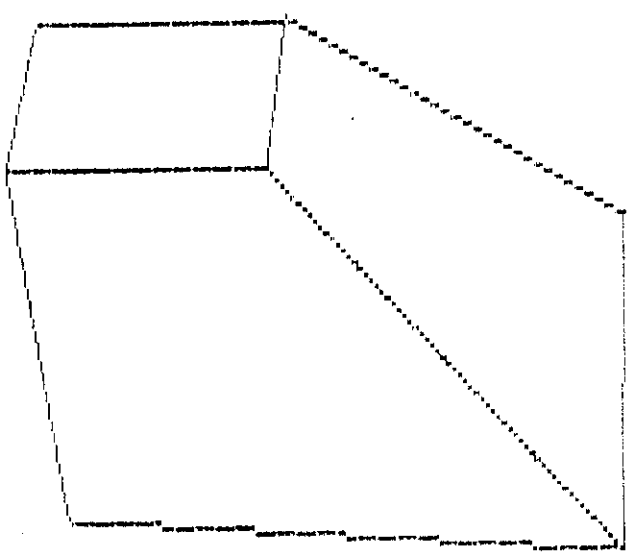
SPECIFY YOUR UP VECTOR

0 1 0



PERSPECTIVE VIEW OF A TRANSLATED OBJECT

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000   0.000000   20.000000   20.000000   20.000000   0.000000   20.00000
 0.000000   0.000000
Enter rotation vectors 1st point
0 0 0

Enter rotation vectors 2nd point
1 1 1
Enter the angle of rotation
40

Enter the values of view points x1,y1,z1
50 50 50

SPECIFY YOUR UP VECTOR
        0 1 0
```

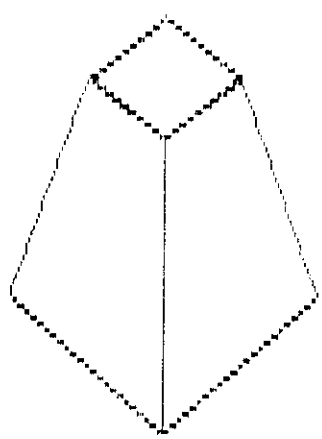


PERSPECTIVE VIEW OF A ROTATED OBJECT

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000    0.000000    20.000000    20.000000    20.000000    0.000000    20.000000
 0.000000    0.000000
Specify your scaling factors
.5 .5 .5


Enter the values of view points x1,y1,z1
50 50 50
```

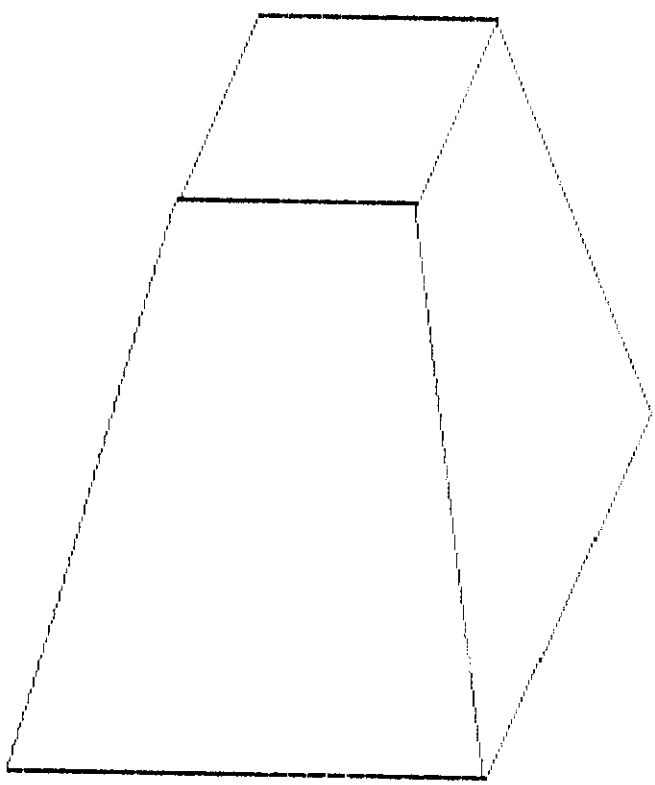
SPECIFY YOUR UP VECTOR

0 1 0



PERSPECTIVE VIEW OF A SCALED OBJECT

```
Do you want to change the co-ordinates of the vi plane('Y') or ('N')
2.000000   0.000000   12.000000   2.000000   10.000000   12.000000   10.000000
 10.000000   12.000000
Specify the first point of the projection vector
0 0 0
Specify the second point of the projection vector
1 2 3

SPECIFY YOUR UP VECTOR

          0 1 0
```

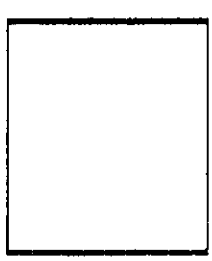


OBLIQUE VIEW

VARIOUS VIEWS OF A CUBE

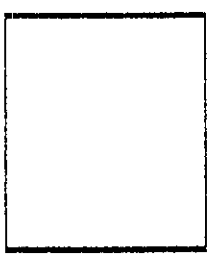
```
Enter the value of x1 y1 z1
3 3 4
Enter the value of x2 y2 z2
3 6 4
Enter the value of x3 y3 z3
6 6 4
Enter the value of x4 y4 z4
6 3 4
Enter the value of x5 y5 z5
3 3 1
Enter the value of x6 y6 z6
3 6 1
Enter the value of x7 y7 z7
6 6 1
Enter the value of x8 y8 z8
6 3 1
```

```
Specify your up vector
0 0 -1
```



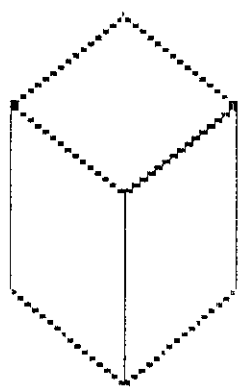
PLAN

```
Specify your up vector
0 1 0
```



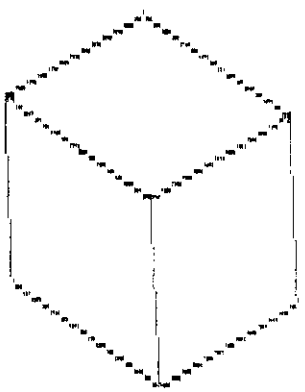
ELEVATION & SIDE VIEW

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000   0.000000   20.000000   20.000000   20.000000   0.000000   20.000000
 0.000000   0.000000
Enter the values of view points x1,y1,z1
50 50 50

SPECIFY YOUR UP VECTOR

          0 1 0
```



PERSPECTIVE VIEW FIG. 2.5

```
Specify your up vector
2 1 1
```

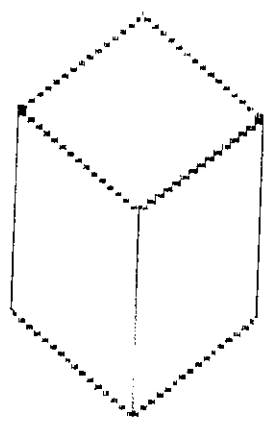


ISOMETRIC VIEW

```
Specify your translation factors
5 10 5

Do you want to change the co-ordinates of the vi.plane( 'Y' ) or ( 'N' )
0.000000    0.000000    20.000000    20.000000    20.000000    0.000000    20.00000
 0.000000    0.000000
Enter the values of view points x1,y1,z1
50 50 50
```

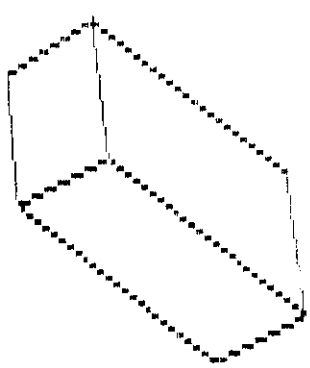
SPECIFY YOUR UP VECTOR

0 1 0



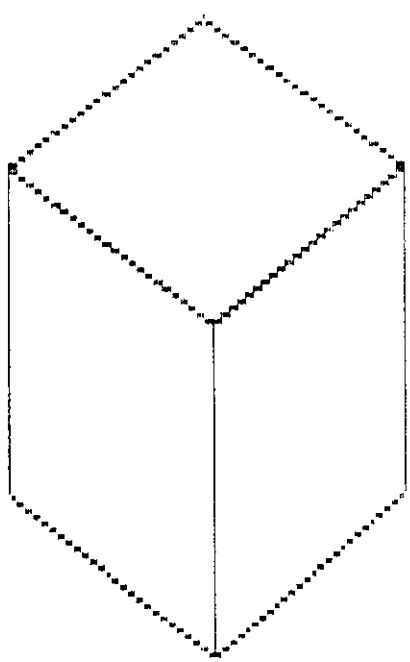
PERSPECTIVE VIEW OF A TRANSLATED OBJECT (FIG. 2.5a)

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000   0.000000   20.000000   20.000000   20.000000   0.000000   20.000000
 0.000000   0.000000
Enter rotation vectors 1st point
0 0 0

Enter rotation vectors 2nd point
1 1 1
Enter the angle of rotation
180

Enter the values of view points x1,y1,z1
70 70 70

 SPECIFY YOUR UP VECTOR

          0 1 0
```



PERSPECTIVE VIEW OF A ROTATED OBJECT (Fig. 2.6b)

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000   0.000000   20.000000   20.000000   20.000000   0.000000   20.000000
 0.000000   0.000000
Specify your scaling factors
1.5 1.5 1.5

SPECIFY YOUR UP VECTOR

          0 1 0
```

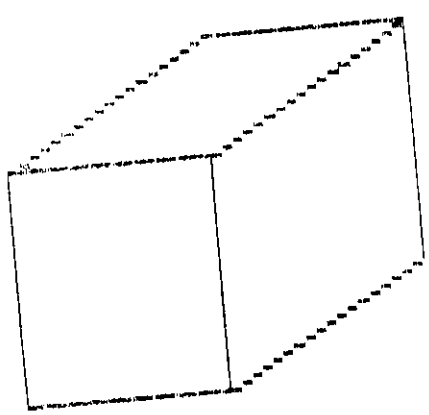


PERSPECTIVE VIEW OF A SCALED OBJECT (Fig. 2.6c)

```
Do you want to change the co-ordinates of the vi plane('Y') or ('N')
2.000000   0.000000   12.000000   2.000000   10.000000   12.000000   10.000000
 10.000000   12.000000
Specify the first point of the projection vector
0 0 0
Specify the second point of the projection vector
2 1 2
```

SPECIFY YOUR UP VECTOR

0 1 0

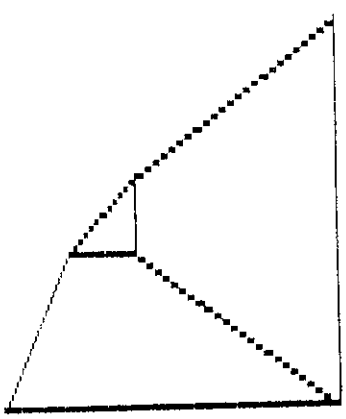


OBLIQUE VIEW

VARIOUS VIEWS OF A TRUNCATED, TRIANGULAR PYRAMID

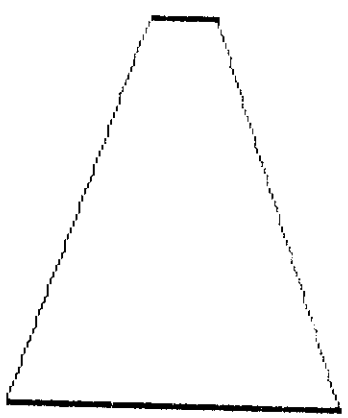
```
Enter the value of x1 y1 z1
1 2 7
Enter the value of x2 y2 z2
2 7 5
Enter the value of x3 y3 z3
3 7 5
Enter the value of x4 y4 z4
6 2 7
Enter the value of x5 y5 z5
2.5 2 4.5
Enter the value of x6 y6 z6
2.5 7 4.5
Enter the value of x7 y7 z7
3 7 4
Enter the value of x8 y8 z8
6 2 2
```

```
Specify your up vector
0 0 -1
```



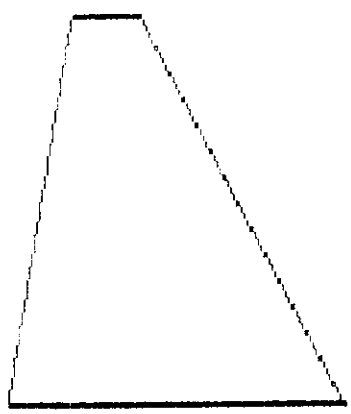
PLAN

```
Specify your up vector
0 1 0
```



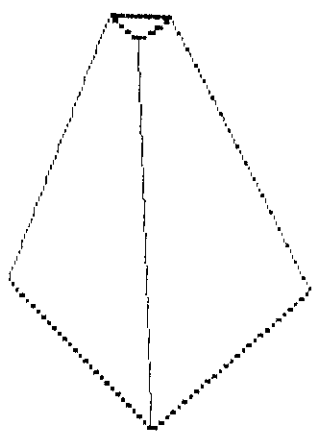
ELEVATION

```
Specify your up vector
0 1 0
```



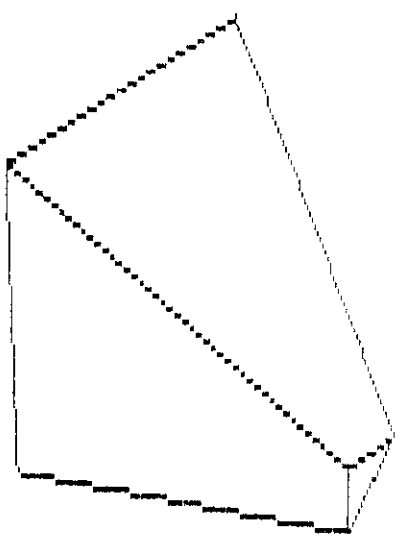
SIDE VIEW

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000    0.000000    20.000000    20.000000    20.000000    0.000000    20.000000
 0.000000    0.000000
Enter the values of view points x1,y1,z1
30 30 30

SPECIFY YOUR UP VECTOR

          0 1 0
```



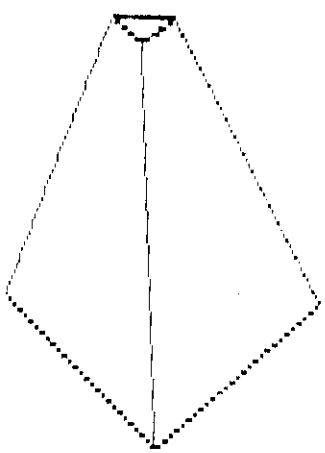
PERSPECTIVE VIEW

```
Specify your up vector
2 1 1
```



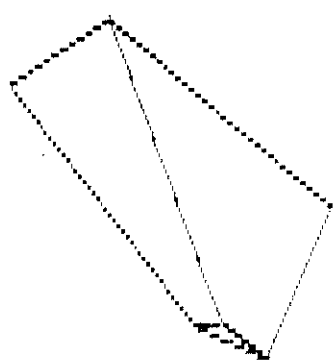
ISOMETRIC VIEW

```
Specify your translation factors
5 10 5

Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000    0.000000    20.000000    20.000000    20.000000    0.000000    20.000
 0.000000    0.000000
Enter the values of view points x1,y1,z1
30 30 30

SPECIFY YOUR UP VECTOR

          0 1 0
```



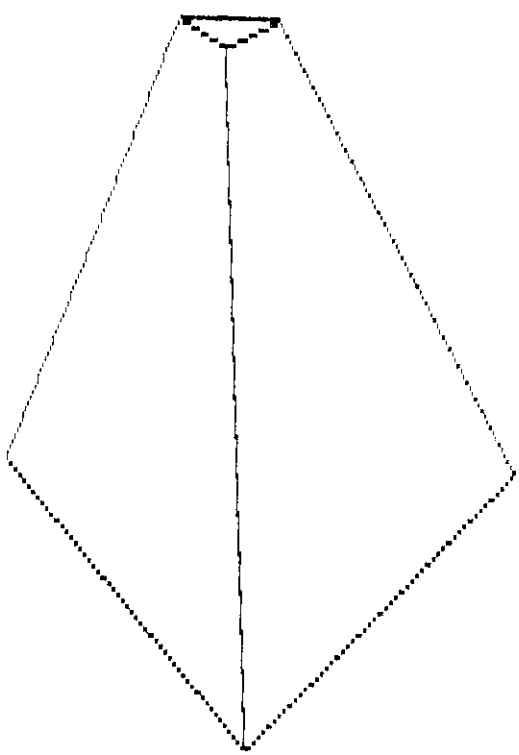
PERSPECTIVE VIEW OF A TRANSLATED OBJECT

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000   0.000000   20.000000   20.000000   20.000000   0.000000   20.0000
 0.000000   0.000000
Enter rotation vectors 1st point
0 0 0

Enter rotation vectors 2nd point
1 1 1
Enter the angle of rotation
180

Enter the values of view points x1,y1,z1
30 30 30

SPECIFY YOUR UP VECTOR

        0 1 0
```



PERSPECTIVE VIEW OF A ROTATED OBJECT

```
Do you want to change the co-ordinates of the vi_plane('Y') or ('N')
0.000000   0.000000   20.000000   20.000000   20.000000   0.000000   20.000
 0.000000   0.000000
Specify your scaling factors
2 1.5 2

Enter the values of view points x1,y1,z1
30 30 30

SPECIFY YOUR UP VECTOR

        0 1 0
```



PERSPECTIVE VIEW OF A SCALED OBJECT

CHAPTER - 5

COMMERCIAL APPLICATIONS

5.1 INTRODUCTION

There are many benefits that result out of the use of this typical package. Some of the benefits are intangible reflected in improved work quality, more pertinent and usuable information, an improved control, all of which are difficult to quality. Other benefits are tangible, but the savings from them show up for down stream us real time application, so that it is difficult to assign a Rupee figure to them in the designed phase.

5.2 BENEFITS

5.2.1 Productivity Improvement in Design

Increased productivity translates into a more competative position for a firm because it will reduce staff requirements on a given project. This leads to lower costs in addition to improving respones time on projects with tight schedules.

Productivity improvement in design as compared to the traditional design process is dependent on such factors as:

Complexity of the Engineering drawing

Level of detail required in the drawing

Degree of Repetitiveness in the designed parts

Degree of symmetry in the parts

Extensiveness of library of commonly used entities

5.2.2. Shorter Lead Times

Interactive design of objects on a computer is inherently faster than the traditional design process. Accordingly, it is possible to produce a finished set of component drawings and the associated reports in a relatively short time. The enhanced productivity of designers working with such systems will tend to reduce the prominence of design, engineering analysis and drafting as critical time elements in overall lead time.

5.2.3. Design Analysis

The design analysis routines available help to consolidate the design process into a more logical work pattern. Rather than having a back-and-forth exchange between design and analysis groups, the same person can perform the analysis while remaining at a graphics work station. This helps to improve the concentration of designers, since they interacting with their designs in a real-time sense. Because of this analysis capability, designs

can be created which are closer to optimum. There is a time saving to be derived from the computerised analysis routines, both in designer time and in elapsed time.

Since alternations in preliminary designs are generally easier to make and analyse with a graphics system more design alternatives can be explored and compared in the available development time. Consequently, it is reasonable to believe that a better design will result from the computerised design procedure.

5.2.4. Fewer Design Errors

Interactive computerised graphics systems provide an intrinsic capability for avoiding design drafting and documentation errors. Errors can be avoided mainly because such a system, with its interactive capabilities can be programmed to question input that may be erroneous.

5.2.5. Greater Accuracy in Design Calculations

There is also a high level of dimensional control, far beyond the levels of accuracy attainable manually. Mathematical accuracy is often to 14 significant decimal places.

5.3 SUMMARY

As can be seen wide ranging applications are possible with slight modifications made depending on the requirements of the user.

CHAPTER - 6

CONCLUSION

As has been stated over and over again the simplicity of this particular graphics package is a plus point in that, it may cater to a wide variety of was with modifications made as and when appropriate.

For example, the object in question which happens to bne a six sided planar figure, may be considered to be a primitive in the dimension 3 sense. This implies that more and more complicated structures may be built out of various transformations of the entity in question.

With suitable further alterations to the structural representation of the data (object vertices, edges, faces),the incorporation of a clipping procedure would result in the creation of an n-sided planer but convex volume.

This package would thus serve as a model on which more and more complicated packages may be based. A suitable set of 3 dimensional primitive would include a cube, a cylinder, a sphere, and a cone. From this given set most of the objects we observe in day to day life may be constructed using simple

concatenation techniques. In itself this package would also prove very useful to engineers and designers by providing a forehand knowledge of the aesthetic qualities of a design yet to be implemented.

BIBLIOGRAPHY

1. Roy A. Plastock C, Gardon Kalley, "Theory and Problems of Computer Graphics, Schaum's Outline Series in Computers (1987).

2. David, F.Rogers, "Procedural Elements For Computer Graphics", McGraw-Hill International Editions.(1988).

3. Stand Kelly - Bootle, "Mastering TURBO C", BPB Publications (1988).

4. William M. Newman, Robert F. Sproull, "Principles of Interactive Computer Graphics", McGraw-Hill International Editions (1989).

5. Steven Harrington, "Computer Graphics" - A Programming Approach, McGraw-Hill International Editions (1987).

6. Donald Hearn, M.Pauline Baker, "Computer Graphics", Prentice Hall International Editions (1986).

7. Ray Duncan, "Advanced MS DOS", Microsoft Corporation.

8. A.L.Stevens, "TURBO-C : Memory - Resident utilities, Screen I/O and Programming Techniques", BPB Publications (1989).

```
#include <graphics.h>
#include  <stdio.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdarg.h>
#include <stdlib.h>
#define ESC 0x1b
char c,c1;
int col,l=0,k,a;
struct viewporttype vp;
int maxx,maxy;
main()
{
  int GraphDriver = CGACO,GraphMode;

  initgraph(&GraphDriver,&GraphMode,"");
  maxx = getmaxx();
  maxy = getmaxy();

  a=1;
  efn(2);

  setviewport(0,0,maxx,maxy,1);
  getviewsettings(&vp);
  rectangle(0,0,vp.right-vp.left,vp.bottom-vp.top-50);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
  outtextxy(90,70,"  WELCOME TO OUR GRAPH LAND");

  a=1;
  c=getch();
  efn(2);

  rectangle(0,0,vp.right-vp.left,vp.bottom-vp.top-25);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
  outtextxy(90,20,"VIEWING TRANSFOMATIONS");
  outtextxy(90,40,"OF A SIX-SIDED OBJECT ");
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  outtextxy(200,70,"DONE BY");
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  outtextxy(200,85,"SRIDHARAN.S");
  outtextxy(200,100,"ELVIN MATHEW VARGHESE");
  outtextxy(200,115,"ARUL.S");
  outtextxy(200,130,"GUIDED BY");
  outtextxy(200,145,"MISS.G.ANDAL JAYALAKSHMI");


  a=0;
  c =getch();
  efn(2);

  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  outtextxy(140,70,"DO YOU REQUIRE HELP('Y')/('N')");
  c1 = getch();
```

```
    if(c1 == 'Y'|| c1 == 'y') help();
    system("sri15.exe");

    }

    efn(col)
    {
    if(ESC == c)
    {
     closegraph();
     exit(1);
     }
     else
     {
       cleardevice();
       setcolor(1);
       if(col == 1)
       setbkcolor(BROWN);
       else if(col == 2)
       setbkcolor(GREEN);
       else setbkcolor(CYAN);
       if(a!=1)
       rectangle(0,0,vp.right-vp.left,vp.bottom-vp.top-50);
       settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
       outtextxy(maxx/2,maxy-8,"Esc Aborts, Press any Key to Cont....");

    }
}

help()
{
  clearviewport();
  setviewport(0,0,maxx,maxy,1);
  getviewsettings(&vp);
  rectangle(0,0,vp.right-vp.left,vp.bottom-vp.top);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  outtextxy(60,30,"  1. SPECIFY VIEW PLANE CO_ORDINATES IN
                      CLOCKWISE ORDER ");
  outtextxy(60,50,"  2. THE VIEW PLANE CO_ORDINATES SHOULD
                      NOT BE COLLINEAR");
  outtextxy(60,70,"  3. THE VIEW PLANE SHOULD LIE  BETWEEN  THE
                      VIEW-POINT AND THE OBJECT");
  outtextxy(60,90,"  4. DON'T SPECIFY AN UPVECTOR PERPENDICULAR TO
                      THE VIEW PLANE");
  outtextxy(60,110," 5. TO SKIP TO THE VERTICAL COLUMN OF
                      THE MENU ,PRESS THE KEY F10");
  outtextxy(60,130," 6. TO COME BACK TO THE MAIN PROGRAM ,PRESS ESC");
  outtextxy(60,150, "7. MAKE SURE THAT THE OBJECT ALWAYS LIES WITHIN
                      THE SCREEN");
  getch();
}
```

```
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
#include "myhead.h"

#if __STDC__
#define _Cdecl
#else
#define _Cdecl  cdecl
#endif

/*To draw a line using DDA algorithm*/

void dda(x1,y1,x2,y2)
float x1,y1,x2,y2;
{
 x11 = 25 * x1;
 y11 = 12 * y1;
 x22 = 25 * x2;
 y22 = 12 * y2;
 dx1 = x22 - x11;
 dy1 = y22 - y11;
 if (dx1 < 0) dx = -(dx1); else dx = dx1;
 if (dy1 < 0) dy = -(dy1); else dy = dy1;

 if (dx > dy) len = dx;
         else len = dy;
            xin = dx1 / len;
            yin = dy1 / len;
            x_1 = x11;
            y_1 = y11;

  for(i=0;i<len;i++)
  {
   x = x_1; y = y_1;
   putpixel(x,192-y,1);
   x_1 = x_1 + xin;
   y_1 = y_1 + yin;
  }
}
```

```
/* To draw a rectangle by passing 4 points */

void rect(x1,y1,x2,y2,x3,y3,x4,y4)
float x1,y1,x2,y2,x3,y3,x4,y4;
{
  dda(x1,y1,x2,y2);
  dda(x2,y2,x3,y3);
  dda(x3,y3,x4,y4);
  dda(x4,y4,x1,y1);
}

/*To read the co_ordinates of the cube*/
/* and to find the maximum of x, y,z values */

void in_mat()
{
  int col;

  if(scl !=1)
  printf("\nDo you want to change the co-ordinates of
                          the cube('Y') or ('N')\n");
  ch = getch();
  if(ch == 'Y' || ch == 'y') system("prog2.exe");

  f1 = fopen("inmat.dat","r");
  for(col = 0; col < 8; col++)
   {
     fscanf(f1,"%f %f %f %f",&M[0][col],&M[1][col],
                             &M[2][col],&M[3][col]);
     if (M[0][col] > x_max) x_max = M[0][col];
     if (M[0][col] > x_min) x_min = M[0][col];
     if (M[1][col] > y_max) y_max = M[1][col];
     if (M[1][col] > y_min) y_min = M[1][col];
     if (M[2][col] > z_max) z_max = M[2][col];
     if (M[2][col] > z_min) z_min = M[2][col];
   }
  fclose(f1);
  getch();
 }
```

```
/* To read the co_ordinates of the view_plane*/

void vplane()

{
  int col;


  printf("\nDo you want to change the co-ordinates of
                        the vi_plane('Y') or ('N')\n");

  ch = getch();
  if(ch == 'Y'|| ch == 'y') system("prog3.exe");
  f2 = fopen("plane.dat","r");
  for(col = 0; col < 9; col++)
   {
     fscanf(f2,"%f ",&VP[col]);
   }

  fclose(f2);
    getch();

 }


/*Matrix multiplication procedure*/

void matmul(in_mat1,in_mat2,out_mat,row1,
                  col1,col2,dim1,dim2,dim3)
float in_mat1[],in_mat2[],out_mat[];
int row1,col1,col2,dim1,dim2,dim3;
{
 int row,temp,col;
 for(row = 0; row < row1; row++)
  for(col = 0; col < col2; col++)
  {
   *(out_mat + (row * dim3 + col)) = 0.0;
   for(temp = 0; temp < col1; temp++)
    *(out_mat + (row * dim3 + col)) +=
     *(in_mat1 + (row * dim1 + temp))
      * (*(in_mat2 + (temp * dim2 + col)));
  }
}
```

```
/*To derive the equation of a*/
/* plane using Martin_newell's equation*/

void solve(A)
float A[3][3];
{
  a = 0.0, b = 0.0,c = 0.0;
  for (row = 0; row <  3; row++)
 {
   col = (row + 1) % 3;

   a = a + ((A[1][row] - A[1][col]) * (A[2][row]
                                     + A[2][col]));
   b = b + ((A[2][row] - A[2][col]) * (A[0][row]
                                     + A[0][col]));
   c = c + ((A[0][row] - A[0][col]) * (A[1][row]
                                     + A[1][col]));
 }

}

/*World to view_plane transform*/

twv()
{
 float m1,m10,m11,m12,r1,r2,r3;
 float lda,mod_uq,mod_njq,dnr,costh,sinth;

 for(row = 0; row < 4; row++)
  for( col = 0; col < 4; col++)
  {
   if(row == col)
   {
    TV[row][col] = 1.0;      A_K[row][col] = -1.0;
    R_O_K[row][col] = 1.0; TRL[row][col] = 1.0;
   }
   else{
         TV[row][col] = 0.0;     A_K[row][col] = 0.0;
         R_O_K[row][col] = 0.0; TRL[row][col] = 0.0;
       }
  }


   TV[0][3] = -VP[0];
   TV[1][3] = -VP[1];
   TV[2][3] = -VP[2];
```

```
DN = sqrt((double)(a * a + b * b + c * c));

a = a / DN;
b = b / DN;
c = c / DN;

for(row = 0; row < 4; row++)
 for( col = 0; col < 4; col++)
  AN[row][col] = 0.0;

lda = sqrt((double)(b * b + c * c));

if (!lda)
{
  AN[0][2] = -a / (sqrt((double)(a * a))); AN[1][1] = 1.0;


  AN[2][0] = -AN[0][2]; AN[3][3] = 1.0;
}
else
  {
   AN[0][0] = lda; AN[0][1] = -(a * b) / lda;
   AN[0][2] = -(a * c) / lda;
   AN[1][1]  = c / lda; AN[1][2] = -b / lda;
   AN[2][0] = a; AN[2][1] = b; AN[2][2] = c;
   AN[3][3] = 1.0;
  }

  printf("\nSpecify your up vector\n");
  scanf("%f%f%f",&u1,&u2,&u3);
  cleardevice();

  m1 = a * u1 + b * u2 + c * u3;

  m10 = m1 * a ; m11 = m1 * b; m12 = m1 * c;

  uq[0][0] = u1 - m10; uq[0][1] = u2 - m11;
  uq[0][2] = u3 - m12;

  mod_uq = sqrt((double)(uq[0][0]*uq[0][0] +
            uq[0][1]* uq[0][1] + uq[0][2]* uq[0][2]));


  r1 = uq[0][0] / mod_uq; r2 = uq[0][1] / mod_uq;
  r3 = uq[0][2];
```

```
    NJQ[0][0] = (b * r3 - c * r2);
    NJQ[0][1] = -(a * r3 - c * r1);
    NJQ[0][2] = (a * r2 - b * r1);

    mod_njq = sqrt((double)(NJQ[0][0]* NJQ[0][0] +
          NJQ[0][1]* NJQ[0][1] + NJQ[0][2]*NJQ[0][2]));

    IP[0][0] = NJQ[0][0] / mod_njq;
    IP[1][0] = NJQ[0][1] / mod_njq;
    IP[2][0] = NJQ[0][2] / mod_njq;
    IP[3][0] = 0;

   for(row = 0; row < 4; row++)
    for( col = 0; col < 4; col++)
     A_K[row][col] = 0.0;

    A_K[1][1] = -1.0; A_K[2][2] = -1.0;
    A_K[0][0] = 1.0; A_K[3][3] = 1.0;

    matmul(A_K,AN,AN_K,4,4,4,4,4,4);
    matmul(AN_K,IP,IP1,4,4,1,4,1,1);

    dnr = sqrt((double)(IP1[0][0] * IP1[0][0]
                    + IP1[1][0] * IP1[1][0]));

    costh = IP1[0][0] / dnr;
    sinth = IP1[1][0] / dnr;

    R_O_K[0][0] = costh; R_O_K[0][1] = -sinth;
    R_O_K[1][1] = costh; R_O_K[1][0] = -sinth;

    TRL[2][2] = -1.0;

    matmul(TRL,R_O_K,RE1,4,4,4,4,4,4);
    matmul(RE1,AN_K,RE2,4,4,4,4,4,4);
    matmul(RE2,TV,RE3,4,4,4,4,4,4);

}
```

```
/*To derive the perspective matrix*/
void derive(x1,y1,z1)
float x1,y1,z1;
{
 DN = sqrt((double)(a * a + b * b + c * c));
 a = a / DN;
 b = b / DN;
 c = c / DN;

 d0 = a * VP[0] + b * VP[1] + c * VP[2];
 d1 = a * x1 + b * y1 + c * z1;
 d = d0 - d1;

 per_mat[0][0] = d + x1 * a;
 per_mat[0][1] = x1 * b;
 per_mat[0][2] = x1 * c;
 per_mat[0][3] = -x1 * d0;

 per_mat[1][0] = y1 * a;
 per_mat[1][1] = d + y1 * b;
 per_mat[1][2] = y1 * c;
 per_mat[1][3] = -y1 * d0;

 per_mat[2][0] = z1 * a;
 per_mat[2][1] = z1 * b;
 per_mat[2][2] = d + z1 * c;
 per_mat[2][3] = -z1 * d0;

 per_mat[3][0] = a;
 per_mat[3][1] = b;
 per_mat[3][2] = c;
 per_mat[3][3] = -d1;

 matmul(per_mat,M,out_mat2,4,4,8,4,8,8);
 matmul(RE3,out_mat2,out_mat1,4,4,8,4,8,8);

  for(col = 0; col < 8; col++)
    for(row = 0; row < 3; row++)
    out_mat1[row][col] = out_mat1[row][col] / out_mat1[3][col];
}
```

```
/*Display Routine*/
show()
{
 su[0][0]  = out_mat1[0][0];
 su[0][1]  = out_mat1[1][0];
 su[0][3]  = out_mat1[0][1];
 su[0][4]  = out_mat1[1][1];
 su[0][6]  = out_mat1[0][2];
 su[0][7]  = out_mat1[1][2];
 su[0][9]  = out_mat1[0][3];
 su[0][10] = out_mat1[1][3];

 su[1][0]  = out_mat1[0][3];
 su[1][1]  = out_mat1[1][3];
 su[1][3]  = out_mat1[0][2];
 su[1][4]  = out_mat1[1][2];
 su[1][6]  = out_mat1[0][6];
 su[1][7]  = out_mat1[1][6];
 su[1][8]  = out_mat1[0][7];
 su[1][10] = out_mat1[1][7];

 su[2][0]  = out_mat1[0][2];
 su[2][1]  = out_mat1[1][2];
 su[2][3]  = out_mat1[0][1];
 su[2][4]  = out_mat1[1][1];
 su[2][6]  = out_mat1[0][5];
 su[2][7]  = out_mat1[1][5];
 su[2][9]  = out_mat1[0][6];
 su[2][10] = out_mat1[1][6];

 su[3][0]  = out_mat1[0][5];
 su[3][1]  = out_mat1[1][5];
 su[3][3]  = out_mat1[0][1];
 su[3][4]  = out_mat1[1][1];
 su[3][6]  = out_mat1[0][0];
 su[3][7]  = out_mat1[1][0];
 su[3][8]  = out_mat1[0][4];
 su[3][10] = out_mat1[1][4];

 su[4][0]  = out_mat1[0][0];
 su[4][1]  = out_mat1[1][0];
 su[4][3]  = out_mat1[0][4];
 su[4][4]  = out_mat1[1][4];
 su[4][6]  = out_mat1[0][7];
 su[4][7]  = out_mat1[1][7];
 su[4][9]  = out_mat1[0][3];
 su[4][10] = out_mat1[1][3];
```

```
 su[5][0]  = out_mat1[0][4];
 su[5][1]  = out_mat1[1][4];
 su[5][3]  = out_mat1[0][7];
 su[5][4]  = out_mat1[1][7];
 su[5][6]  = out_mat1[0][6];
 su[5][7]  = out_mat1[1][6];
 su[5][8]  = out_mat1[0][5];
 su[5][10] = out_mat1[1][5];
}


/* Surface construction routine*/

make_surface()
{
 su1[0][0]  = M[0][0];
 su1[0][1]  = M[1][0];
 su1[0][2]  = M[2][0];
 su1[0][3]  = M[0][1];
 su1[0][4]  = M[1][1];
 su1[0][5]  = M[2][1];
 su1[0][6]  = M[0][2];
 su1[0][7]  = M[1][2];
 su1[0][8]  = M[2][2];
 su1[0][9]  = M[0][3];
 su1[0][10] = M[1][3];
 su1[0][11] = M[2][3];

 su1[1][0]  = M[0][3];
 su1[1][1]  = M[1][3];
 su1[1][2]  = M[2][3];
 su1[1][3]  = M[0][2];
 su1[1][4]  = M[1][2];
 su1[1][5]  = M[2][2];
 su1[1][6]  = M[0][6];
 su1[1][7]  = M[1][6];
 su1[1][8]  = M[2][6];
 su1[1][9]  = M[0][7];
 su1[1][10] = M[1][7];
 su1[1][11] = M[2][7];

 su1[2][0]  = M[0][2];
 su1[2][1]  = M[1][2];
 su1[2][2]  = M[2][2];
 su1[2][3]  = M[0][1];
 su1[2][4]  = M[1][1];
 su1[2][5]  = M[2][1];
 su1[2][6]  = M[0][5];
 su1[2][7]  = M[1][5];
```

```
su1[2][8]  = M[2][5];
su1[2][9]  = M[0][6];
su1[2][10] = M[1][6];
su1[2][11] = M[2][6];

su1[3][0]  = M[0][5];
su1[3][1]  = M[1][5];
su1[3][2]  = M[2][5];
su1[3][3]  = M[0][1];
su1[3][4]  = M[1][1];
su1[3][5]  = M[2][1];
su1[3][6]  = M[0][0];
su1[3][7]  = M[1][0];
su1[3][8]  = M[2][0];
su1[3][8]  = M[0][4];
su1[3][10] = M[1][4];
su1[3][11] = M[2][4];

su1[4][0]  = M[0][0];
su1[4][1]  = M[1][0];
su1[4][2]  = M[2][0];
su1[4][3]  = M[0][4];
su1[4][4]  = M[1][4];
su1[4][5]  = M[2][4];
su1[4][6]  = M[0][7];
su1[4][7]  = M[1][7];
su1[4][8]  = M[2][7];
su1[4][9]  = M[0][3];
su1[4][10] = M[1][3];
su1[4][11] = M[2][3];

su1[5][0]  = M[0][4];
su1[5][1]  = M[1][4];
su1[5][2]  = M[2][4];
su1[5][3]  = M[0][7];
su1[5][4]  = M[1][7];
su1[5][5]  = M[2][7];
su1[5][6]  = M[0][6];
su1[5][7]  = M[1][6];
su1[5][8]  = M[2][6];
su1[5][9]  = M[0][5];
su1[5][10] = M[1][5];
su1[5][11] = M[2][5];

}
```

```
/* Hidden line elimination routine*/

void hide()
{
 float A[3][3];
 int i,j,z;
 float mid_x = 0.0,mid_y = 0.0,mid_z = 0.0;
 float smid_x = 0.0,smid_y = 0.0,smid_z = 0.0;
 float x_val = 0.0,y_val = 0.0,z_val = 0.0;
 float dnum;


make_surface();

for(i = 0; i < 6; i++)
  {


    j = 0; z = 0;
    while(j < 9)
     {
      A[0][z] = su1[i][j++];
      A[1][z] = su1[i][j++];
      A[2][z] = su1[i][j++];
      z++;
     }
     solve(A);

     d = -(a * su1[i][0] + b * su1[i][1] + c * su1[i][2]);


    V[0][i] = a; V[1][i] = b; V[2][i] = c; V[3][i] = c;
  }

  for(i = 0; i < 8; i++)
  {
   mid_x = mid_x + M[0][i];
   mid_y = mid_y + M[1][i];
   mid_z = mid_z + M[2][i];
  }

  S[0][0] = mid_x / 8.0;
  S[0][1] = mid_y / 8.0;
  S[0][2] = mid_z / 8.0;
  S[0][3] = 1.0;

  matmul(S,V,view_mat,1,4,6,4,6,6);
```

```
for(i = 0; i < 6; i++)
{
 if (view_mat[0][i] < 0)
 {
   V[0][i] = -1 * V[0][i];
   V[1][i] = -1 * V[1][i];
   V[2][i] = -1 * V[2][i];
   V[3][i] = -1 * V[3][i];
 }
}


for(i = 0; i < 6; i++)
{
 x_val = su1[i][0] + su1[i][3] + su1[i][6] + su1[i][9];
 y_val = su1[i][1] + su1[i][4] + su1[i][7] + su1[i][10];
 z_val = su1[i][2] + su1[i][5] + su1[i][8] + su1[i][11];

 smid_x = x_val / 4.0;
 smid_y = y_val / 4.0;
 smid_z = z_val / 4.0;

 smid_x = smid_x - x1;
 smid_y = smid_y - y1;
 smid_z = smid_z - z1;

 dnum = sqrt((double)(smid_x * smid_x + smid_y *
                      smid_y + smid_z * smid_z));


 smid_x = smid_x / dnum;
 smid_y = smid_y / dnum;
 smid_z = smid_z / dnum;

 F_mat[0][i] = smid_x * V[0][i] + smid_y
                      * V[1][i] + smid_z * V[2][i];

 }
}
```

```
/* Perspective transformation procedure */
void perspective(x1,y1,z1)
float x1,y1,z1;
{
float A[3][3];
int j,z;
  z = 0;
  j = 0;
  while(j < 9)
  {
   A[0][z] = VP[j++];
   A[1][z] = VP[j++];
   A[2][z] = VP[j++];
   z++;
  }
  solve(A);

  twv();
  derive(x1,y1,z1);
  show();
  hide();
  for(row = 0; row < 6; row++)
  {
   if(F_mat[0][row] >= 0)
  rect(su[row][0],su[row][1],su[row][3],su[row][4],
         su[row][6],su[row][7],su[row][9],su[row][10]);
  }
 }

/* Parallel transformation procedure */

parallel()
{
  float A[3][3];
  int j,z;
  float nr,dr,ve1,ve2,ve3,dnum,dn1;
  float t,mid_x=0,mid_y=0,mid_z=0;

  z = 0;
  j = 0;
  while(j < 9)
  {
   A[0][z] = VP[j++];
   A[1][z] = VP[j++];
   A[2][z] = VP[j++];
   z++;
  }
  solve(A);
```

```
dn1 = sqrt((double)(a * a + b * b + c * c));

a = a / dn1;
b = b / dn1;
c = c / dn1;

ve1 = q1 - p1; ve2 = q2 - p2; ve3 = q3 - p3;

dnum = sqrt((double)(ve1 * ve1 + ve2 * ve2 + ve3 * ve3));

px[0][0] = ve1 / dnum; px[0][1] = ve2 / dnum;
px[0][2] = ve3 / dnum;
px[0][3] = 0;

dr = a * px[0][0] + b * px[0][1] + c * px[0][2];

for (row = 0; row < 4; row++)
for (col = 0; col < 8; col++)
{
  M1[row][col] = 1.0;
}

for (row = 0; row < 8; row++)
 {
 nr = a * (M[0][row]- VP[0]) + b * (M[1][row] - VP[1]) +
                               c * (M[2][row] - VP[2]);

 t = - nr / dr;
 M1[0][row] = M[0][row] + px[0][0] * t;
 M1[1][row] = M[1][row] + px[0][1] * t;
 M1[2][row] = M[2][row] + px[0][2] * t;
}

twv();
matmul(RE3,M1,out_mat1,4,4,8,4,8,8);

for(col = 0; col < 8; col++)
  for(row = 0; row < 3; row++)
   out_mat1[row][col] = out_mat1[row][col] /
                        out_mat1[3][col];

  show();

 make_surface();
```

```
  for(i = 0; i < 6; i++)
 {
   j = 0; z = 0;

   while(j < 9)
    {
     A[0][z] = su1[i][j++];
     A[1][z] = su1[i][j++];
     A[2][z] = su1[i][j++];
     z++;
    }
    solve(A);

    d = -(a * su1[i][0] + b * su1[i][1] + c * su1[i][2]);
    V[0][i] = a; V[1][i] = b; V[2][i] = c; V[3][i] = d;
 }

 for(i = 0; i < 8; i++)
 {
  mid_x = mid_x + M[0][i];
  mid_y = mid_y + M[1][i];
  mid_z = mid_z + M[2][i];
 }

 S[0][0] = mid_x / 8.0;
 S[0][1] = mid_y / 8.0;
 S[0][2] = mid_z / 8.0;
 S[0][3] = 1.0;

 px[0][0] = -px[0][0]; px[0][1] = -px[0][1];
 px[0][2] = -px[0][2];

 matmul(S,V,view_mat,1,4,6,4,6,6);

 for(i = 0; i < 6; i++)
 {
  if (view_mat[0][i] < 0)
  {
    V[0][i] = -1 * V[0][i];
    V[1][i] = -1 * V[1][i];
    V[2][i] = -1 * V[2][i];
    V[3][i] = -1 * V[3][i];
  }
}

 matmul(px,V,NE,1,4,6,4,6,6);
```

```
  for(row = 0; row < 6; row++)
  {
   if(NE[0][row] > 0)
  rect(su[row][0],su[row][1],su[row][3],su[row][4],
         su[row][6],su[row][7],su[row][9],su[row][10]);
  }
}

/* Plan routine */

void plan()
{

  VP[0] = 0.0;VP[1] = y_max;VP[2] = z_max;
  VP[3] = 0.0;VP[4] = y_max;VP[5] = 0.0;
  VP[6] = x_max;VP[7] = y_max;VP[8] = 0.0;
  p1 = 0; p2 = 0; p3 = 0;
  q1 = 0 ; q2 = 1; q3 = 0;
  parallel();

}

/* Elevation routine */

void elevation()
{

  VP[0] = 0.0;VP[1] = 0.0;VP[2] = z_max;
  VP[3] = 0.0;VP[4] = y_max;VP[5] = z_max;
  VP[6] = x_max;VP[7] = y_max;VP[8] =z_max;
  p1 = 0; p2 = 0; p3 = 0;


  q1 = 0 ; q2 = 0; q3 = 1;
  parallel();

}

/* End elevation routine */
void sideview()
{

 VP[0] = x_max;VP[1] = 0.0;VP[2] = z_max;
 VP[3] = x_max;VP[4] = y_max;VP[5] = 0.0;
 VP[6] = x_max;;VP[7] = 0.0;VP[8] = 0.0;
 p1 = 0; p2 = 0; p3 = 0;
 q1 = 1 ; q2 = 0; q3 = 0;
 parallel();
}
```

```
/* Isometric routine */

void isometric()
{
  float b_M,B_M;

  b_M = (x_max > y_max) ? x_max : y_max;
  B_M = (b_M > z_max) ? b_M : z_max;
  VP[0] = 0.0;VP[1] = 0.0;VP[2] = 3 * B_M;
  VP[3] = 0.0;VP[4] = 3 * B_M;VP[5] = 0.0;
  VP[6] = 3 * B_M;VP[7] = 0.0;VP[8] = 0.0;
  p1 = 0; p2 = 0; p3 = 0;
  q1 = 1 ; q2 = 1; q3 = 1;
  parallel();

}

/* Rotation procedure */

void rotate()
{
  float ve1,ve2,ve3,dnum ,lda;
  float TEM1[4][4],TEM2[4][4],TEM3[4][4],TEM4[4][4];

  printf("\nEnter rotation vectors 1st point\n");
  scanf("%f%f%f" ,&p1,&p2,&p3);
  printf("\nEnter rotation vectors 2nd point\n");
  scanf("%f%f%f" ,&q1,&q2,&q3);
  printf("Enter the angle of rotation\n");
  scanf("%f", &ang);

  for (row = 0; row < 4; row++)
  for(col = 0; col < 4; col++)
   {
    if(row == col)
    {
     RTP[row][col] = 1.0; RPT[row][col] = 1.0;
     PAN_1[row][col] = 1.0; PAN[row][col] = 1.0;
     RTK[row][col] = 1.0;
    }
    else
    {
     RTP[row][col] = 0.0; RPT[row][col] = 0.0;
     PAN_1[row][col] = 0.0; PAN[row][col] = 0.0;
     RTK[row][col] = 0.0;
    }
  }
```

```
ve1 = q1 - p1; ve2 = q2 - p2; ve3 = q3 - p3;

 dnum = sqrt((double)(ve1 * ve1 + ve2 * ve2 + ve3 * ve3));


RPT[0][3] = q1; RPT[1][3] = q2; RPT[2][3] = q3;
RTP[0][3] = -q1; RTP[1][3] = -q2; RTP[2][3] = -q3;

ve1 = ve1 / dnum; ve2 = ve2 / dnum; ve3 = ve3 / dnum;
lda = sqrt((double)(ve2 + ve3));
if (!lda)
   {
    PAN[0][2] = -ve1 / (sqrt((double)(ve1 * ve1)));
    PAN[2][0] = -PAN[0][2];

    PAN[0][2] = -ve1 / (sqrt((double)(ve1 * ve1)));
    PAN[2][0] = -PAN[0][2];
   }
else
  {

    PAN[0][0] = lda; PAN[0][1] = -(ve1 * ve2) / lda;
    PAN[0][2] = -(ve1* ve3) / lda;
    PAN[1][1] = ve3 / lda; PAN[1][2] = -ve2 / lda;
    PAN[2][0] = ve1; PAN[2][1] = ve2; PAN[2][2] = ve3;

    PAN_1[0][0] = lda; PAN_1[0][2] = ve1;


    PAN_1[1][0] = -(ve1 * ve2) / lda;
    PAN_1[1][1] = ve3 / lda; PAN_1[1][2] = ve2;
    PAN_1[2][0] = -(ve1 * ve3) / lda;
    PAN_1[2][1] = -ve2 / lda; PAN_1[2][2] = ve3;
  }

 ang = (ang * 3.14) / 180;

RTK[0][0] = cos(ang); RTK[0][1] = -sin(ang);
RTK[1][0] = sin(ang); RTK[1][1] = cos(ang);


matmul(RPT,PAN_1,TEM1,4,4,4,4,4,4);
matmul(TEM1,RTK,TEM2,4,4,4,4,4,4);
matmul(TEM2,PAN,TEM3,4,4,4,4,4,4);
matmul(TEM3,RTP,TEM4,4,4,4,4,4,4);
matmul(TEM4,M,OUT,4,4,8,4,8,8);
```

```
 for (row = 0; row < 4; row++)
  for(col = 0; col < 8; col++)
    M[row][col] = OUT[row][col];

 }

/* Scaling routine */

 scale()
 {

  float sx,sy,sz;

 for (row = 0; row < 4; row++)
  for(col = 0; col < 4; col++)
   SC[row][col] = 0.0;

   printf("\nSpecify your scaling factors\n");
   scanf("%f%f%f",&sx,&sy,&sz);
   cleardevice();

   SC[0][0] = sx; SC[1][1] = sy; SC[2][2] = sz;SC[3][3] = 1;

   matmul(SC,M,OUT,4,4,8,4,8,8);

   for (row = 0; row < 4; row++)
    for(col = 0; col < 8; col++)
     M[row][col] = OUT[row][col];

     printf("Enter the values of view points x1,y1,z1\n");

     scanf("%f %f %f",&x1,&y1,&z1);
     cleardevice();
     perspective(x1,y1,z1);
  }

/* Translation routine */

trans(tx,ty,tz)
float tx,ty,tz;
{

 for (row = 0; row < 4; row++)
  for(col = 0; col < 4; col++)
  {
    if(row == col) TR[row][col] = 1.0;
    else TR[row][col] = 0.0;
  }
```

```
    TR[0][3] = tx; TR[1][3] = ty; TR[2][3] = tz;TR[3][3] = 1;


      printf("Enter the values of view points x1,y1,z1\n");
      scanf("%f %f %f",&x1,&y1,&z1);
      cleardevice();
      perspective(x1,y1,z1);
   }


/* Menu creation routine */

menu_build()
{

   cleardevice();

   line(10,168,550,168);
   line(10,200,620,200);
   line(550,10,550,170);
   line(620,10,620,200);

   i = 110;
   while(i < 560)
   {
    line(i,168,i,200);

    i += 110;
   }

   j = 42;

   while(j < 170)
   {
    line(550,j,620,j);
    j += 42;
   }
   settextstyle(0,0,1);
   moveto(10,180);
   outtext("PLAN");
   moveto(125,180);
   outtext("ELEVATION");
   moveto(235,180);
   outtext("SIDEVIEW");
   moveto(350,180);
   outtext("PERSPECTIVE");
   moveto(465,180);
```

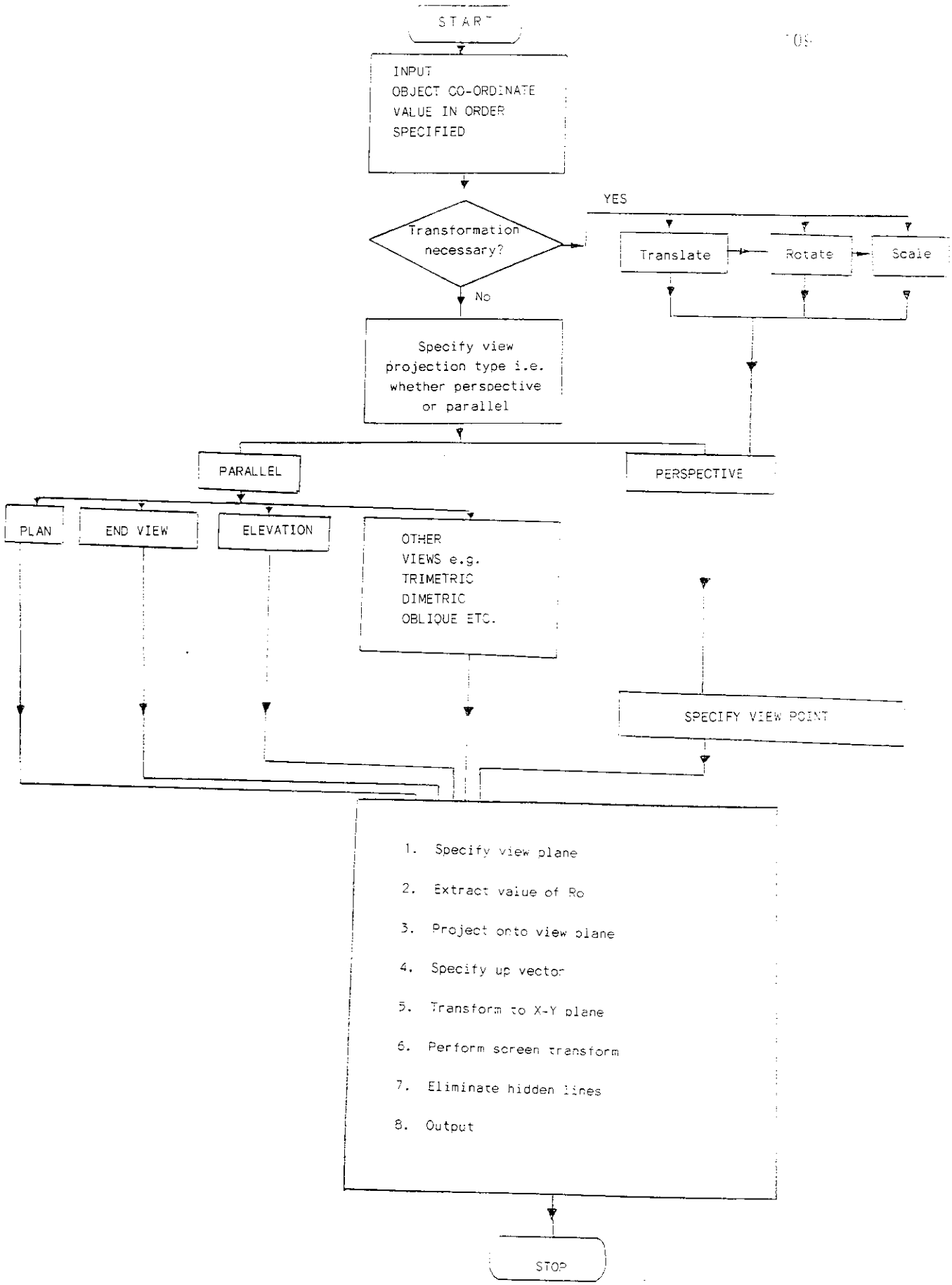
```
  outtext("ISOMETRIC");
  moveto(570,180);

  settextstyle(0,0,1);
  moveto(550,20);
  outtext("TRANSLATE");
  moveto(560,60);
  outtext("ROTATE");
  moveto(560,100);
  outtext("SCALE");
  moveto(550,140);
  outtext("PARALLEL");
  }


/* Menu_processing routine */

menu_process()
 {
 over = 1; hori_menu = 1;
 cur_x = 63,cur_y = 175;

 while(over)
 {
  moveto(cur_x,cur_y);
  outtext("_");
  again = 0;


  inregs.h.ah = 0X08;
  intdos(&inregs,&outregs);
  c1 = outregs.h.al;
  switch(c1)
  {
  case 0 :   outtext(" ");
             inregs.h.ah = 0X08;
             intdos(&inregs,&outregs);
             switch(outregs.h.al)
             {
               case 72 : if(vert_menu)
                         {
                           cur_y = (cur_y - 42) % 160;
                           cur_x = 585;
                         }


                         break;
```

```
        case 80 :  if(vert_menu)
                   {
                     cur_y = (cur_y + 42) % 160;
                     cur_x = 585;
                   }
                   break;

        case 68 : if(hori_menu)
                   {
                     hori_menu = 0;
                     vert_menu = 1;
                     cur_x = 585; cur_y = 25;
                   }
                   else{
                   if(vert_menu)
                   {
                     vert_menu = 0;
                     hori_menu = 1;
                     cur_x = 63; cur_y = 175;
                   }
                   }
                   break;

        case 77 : if(hori_menu)
                   {


                    cur_x = (cur_x + 110) % 530;


                    cur_y = 185;
                   }
                   break;

        case 75 : if(hori_menu)
                   {
                    cur_x = (cur_x - 110) % 530;
                    cur_y = 185;
                   }
                   break;


       }

   break;
```

```
 case 27 :    over = 0; break;

case 13 :{
           cleardevice();

           if(vert_menu)
           {
            switch(cur_y)
            {
             case 25 :printf("\nSpecify your translation
                                          factors\n");
                      scanf("%f%f%f",&tx,&ty,&tz);
                      vplane();
                      cleardevice();
                      trans(tx,ty,tz);
                      getch();
                      break;

             case 67 : vplane();
                       rotate();
                       printf("\nEnter the values of
                             view points x1,y1,z1\n");
                      scanf("%f %f %f",&x1,&y1,&z1);
                      cleardevice();
                      perspective(x1,y1,z1);
                      getch();
                      break;

           case 109 : vplane();
                       if(scl == 1)
                       {


                        printf("Do you want to go back
                             to the original object?");
                        c2 = getch();
                        if(c2 == 'Y' || c2 == 'y')
                      {
                      in_mat();
                      printf("\nEnter the values of
                             view points x1,y1,z1\n");
                      scanf("%f %f %f",&x1,&y1,&z1);
                      cleardevice();
                      perspective(x1,y1,z1);scl = 0;
                      }
                     }
```

```
                else
                 {
                scale(); scl=1 ;
                 }
                getch();
                break;

     case 151 : vplane();
                printf("\nSpecify the first point
                     of the projection vector\n");
                scanf("%f %f %f",&p1,&p2,&p3);
                printf("Specify the second point
                     of the projection vector\n");
                scanf("%f %f %f",&q1,&q2,&q3);
                cleardevice();
                parallel();
                getch();
                break;
    }
   }


else{
if(hori_menu)
{
switch(cur_x)
{
 case 63   : plan();
             getch();
             break;

 case 173 : elevation();
             getch();
             break;

 case 283 : sideview();
             getch(); break;


 case 393 : vplane();
             printf("\nEnter the values of
                     view points x1,y1,z1\n");
             scanf("%f %f %f",&x1,&y1,&z1);
             cleardevice();
             perspective(x1,y1,z1);
             getch();
             break;
```

```
                case 503 : isometric();
                           getch();
                           break;

              } /*for switch*/

              }/*for if*/

     }  /*for else if*/
     again = 1;
  } /*for case 13*/
     break;

   }/*for 1st switch*/
     if(again) menu_build();

  } /*for while*/

  }


  /* MAIN PROGRAM */

  main()
  {

   in_mat();

   initgraph(&graphdriver,&graphmode," ");

   menu_build();
   menu_process();

  getch();
  closegraph();
  }
```


START
INPUT
OBJECT CO-ORDINATE
VALUE IN ORDER
SPECIFIED
Transformation necessary?
YES
Translate
Rotate
Scale
No
Specify view projection type i.e. whether perspective or parallel
PARALLEL
PERSPECTIVE
PLAN
END VIEW
ELEVATION
OTHER
VIEWS e.g.
TRIMETRIC
DIMETRIC
OBLIQUE ETC.
SPECIFY VIEW POINT
1. Specify view plane
2. Extract value of Ro
3. Project onto view plane
4. Specify up vector
5. Transform to X-Y plane
6. Perform screen transform
7. Eliminate hidden lines
8. Output
STOP

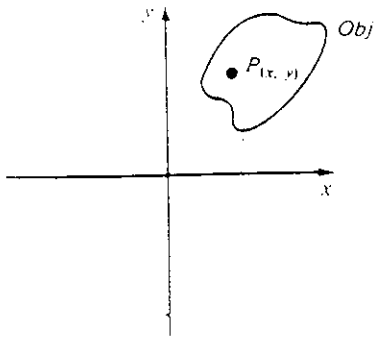


Fig. 2·1

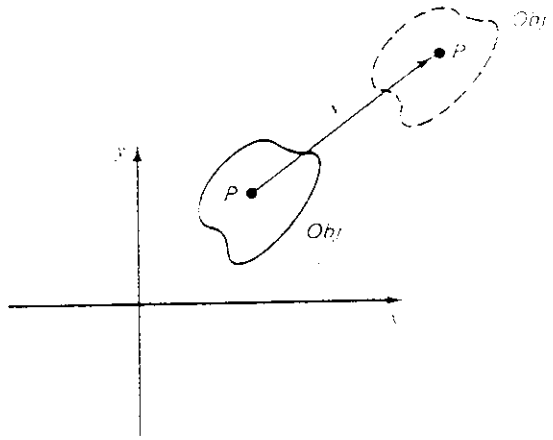


Fig. 2·2

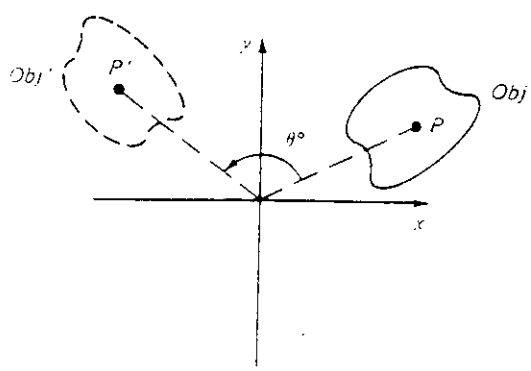
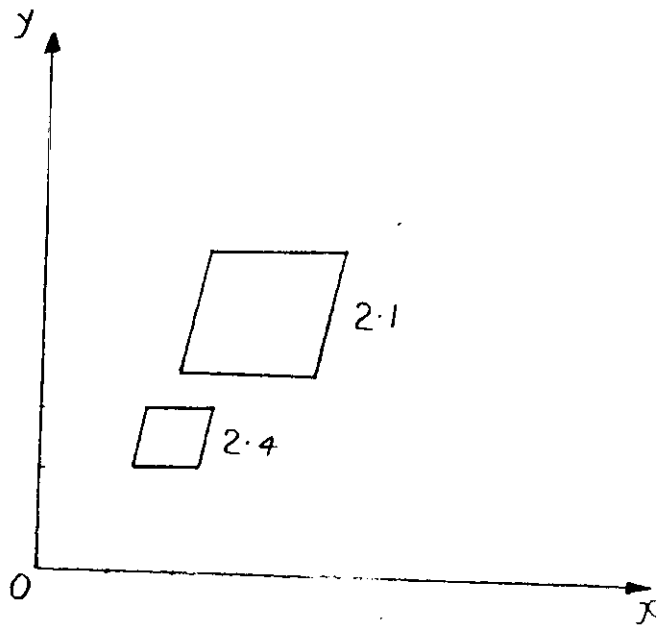


Fig. 2·3

SCALING



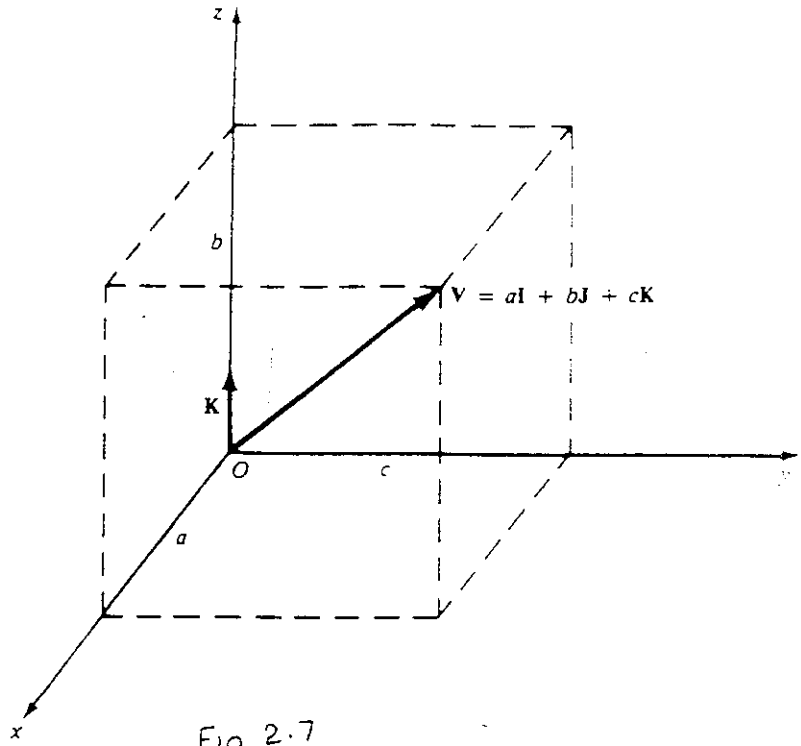


Fig 2.7

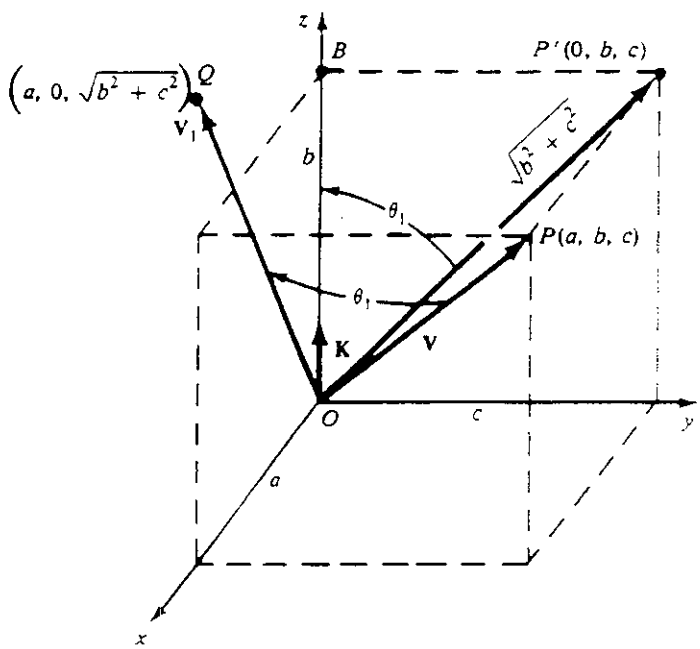


Fig 2.8

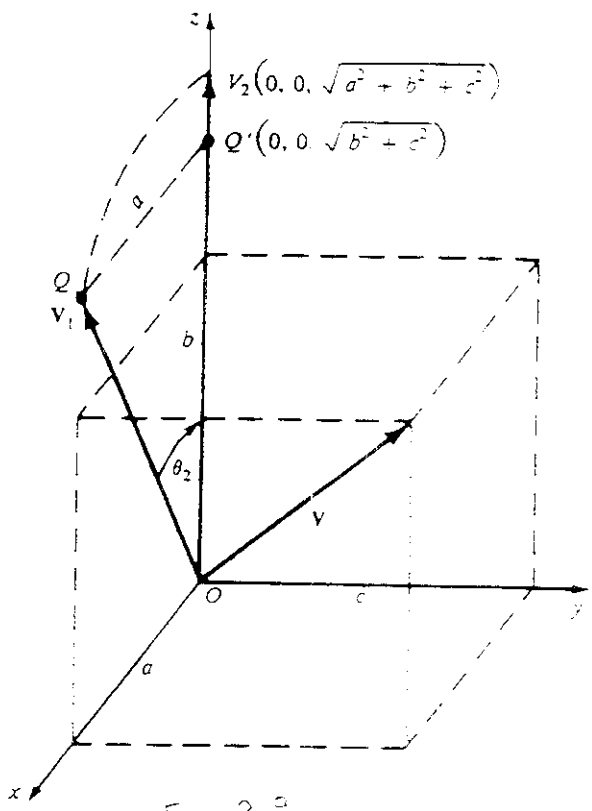
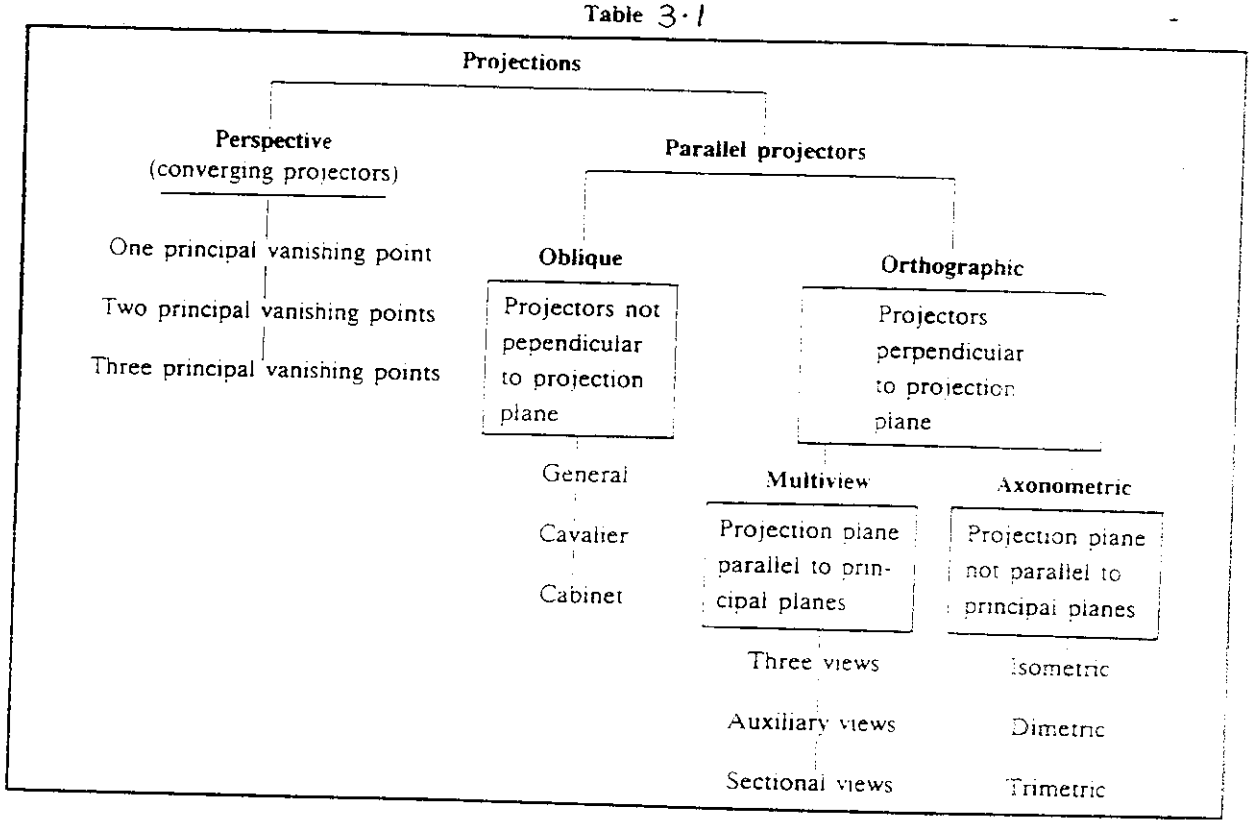
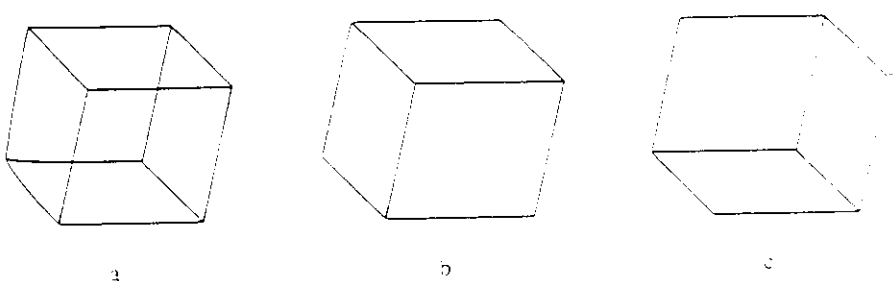


Fig 2.9

Table 3·1





a b c

Figure 3·2 Need for hidden surfaces.

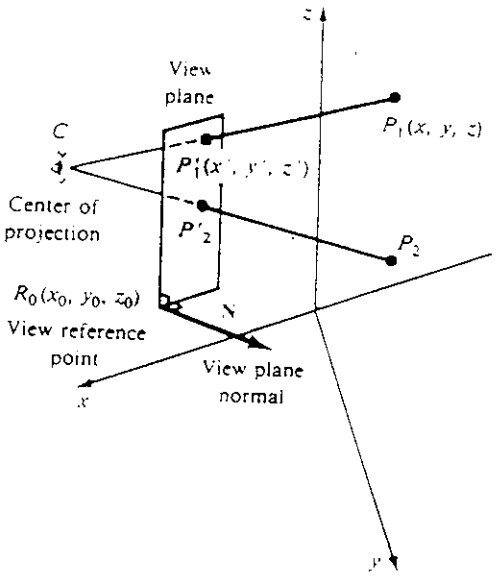


Fig. 4·1

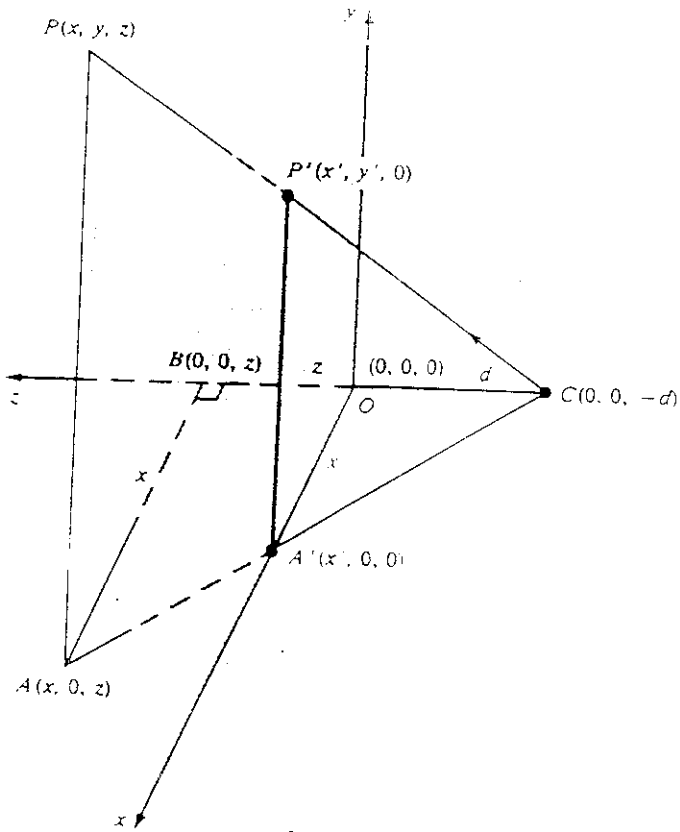
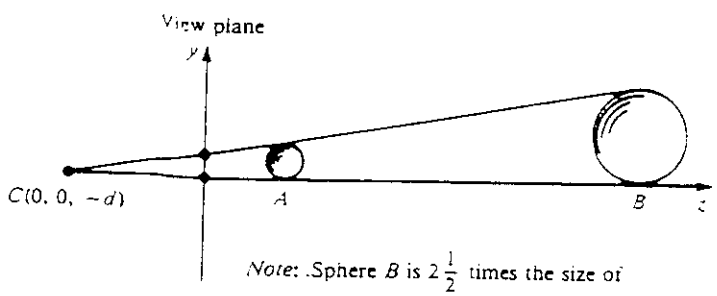


Fig 4·2



Note: Sphere B is $2\frac{1}{2}$ times the size of sphere A; yet both spheres appear to be the same size when projected onto the view plane

Fig. 4·3

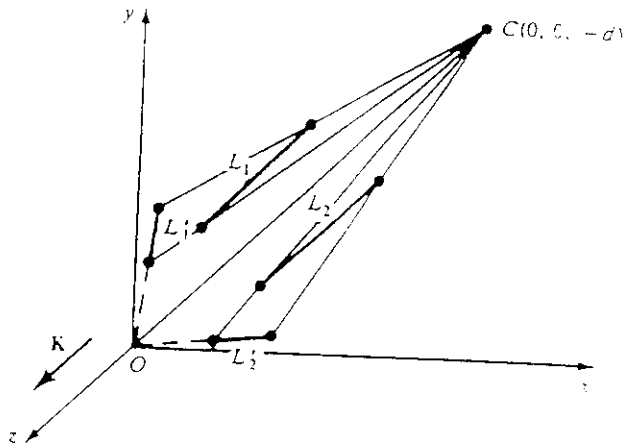


Fig 4·4

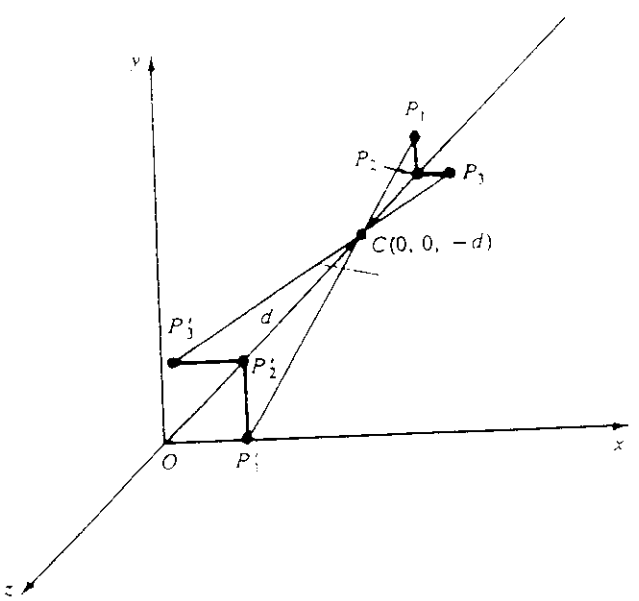
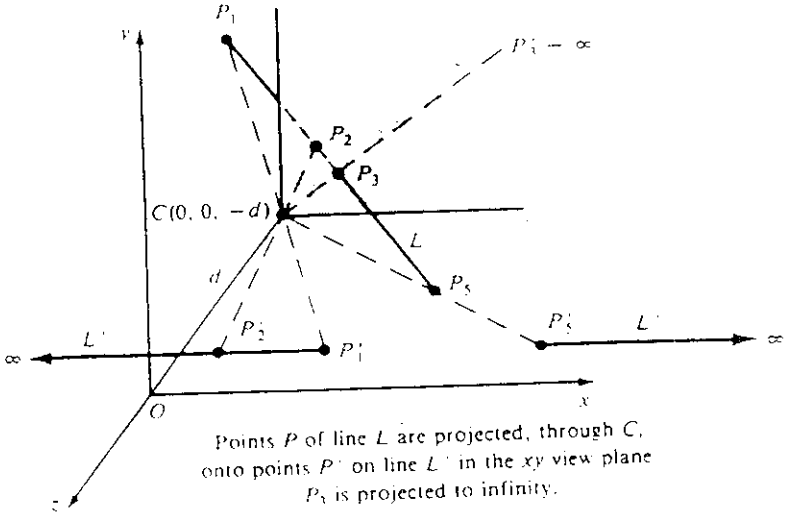


Fig. 4·5



Points P of line L are projected, through C, onto points P' on line L' in the xy view plane. P_5 is projected to infinity.

Fig. 4·6

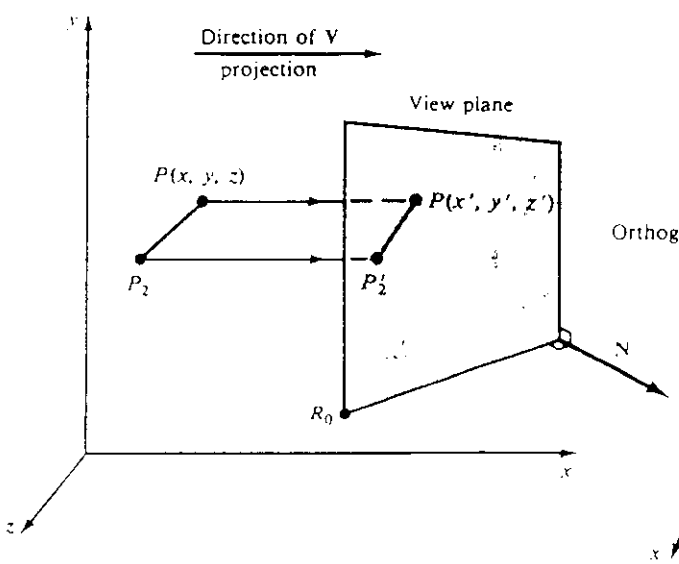


Fig. 4·7

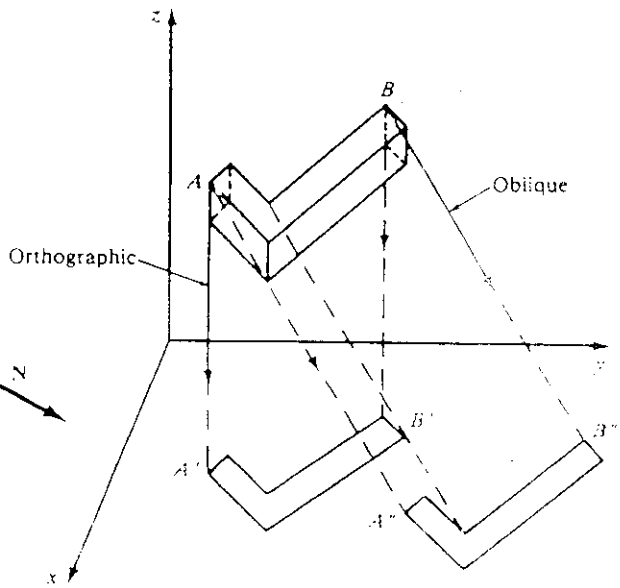


Fig. 4·8

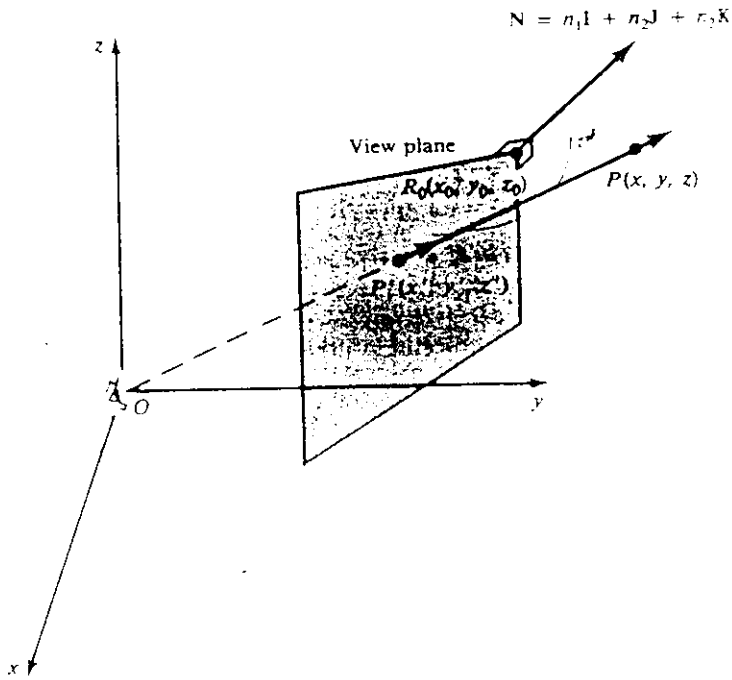


Fig. 4·9

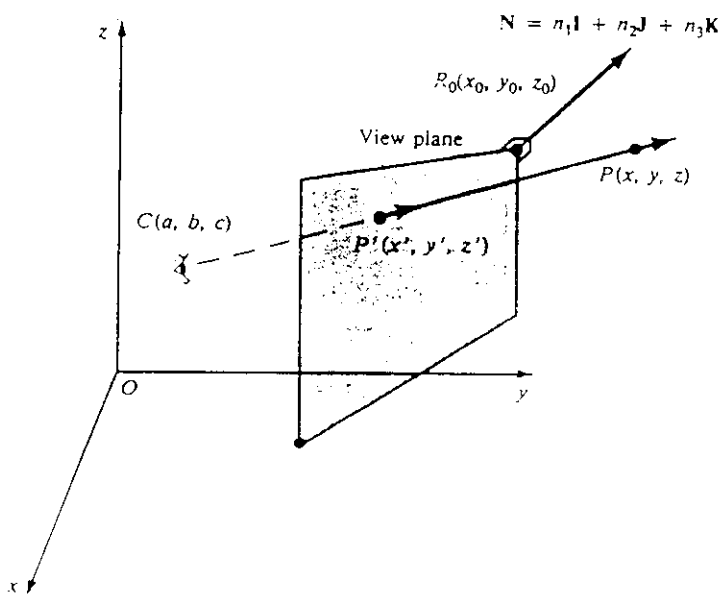


Fig. 4·10

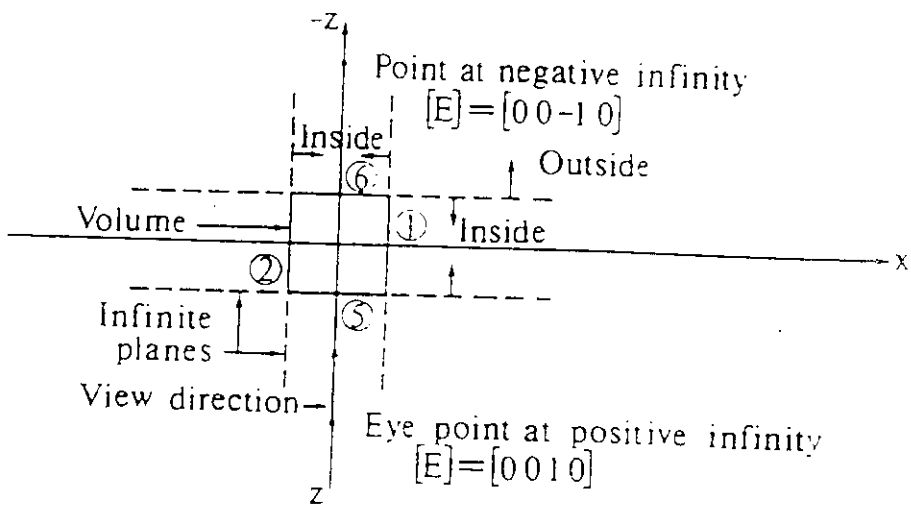


Figure 4·14 Self-hidden planes.

```
FILE *f2;
float x,y,z;
int i= 0;
main()
{
 clrscr();
 f2 = fopen("plane.dat","w");
 while(i < 3)
 {
  i += 1;
  printf("Enter the value of x%d y%d z%d\n",i,i,i);
  scanf("%f%f%f",&x,&y,&z);
  fprintf(f2,"%f %f %f \n",x,y,z);
 }
 fclose(f2);
}
```