मङ्गलायतन विश्वविद्यालय

॥ विश्वं ज्ञाने प्रतिष्ठितम् ॥

MANGALAYATAN
UNIVERSITY

Learn Today to Lead Tomorrow

Computer Graphics

MAL-6115

Edited By

Dr. Hibah Islahi

DIRECTORATE OF DISTANCE AND ONLINE EDUCATION

MANGALAYATAN
UNIVERSITY

UNIT I

INTRODUCTION TO COMPUTER GRAPHICS AND GRAPHIC SYSTEMS

NOTES

STRUCTURE

1.0 LEARNING OBJECTIVES

After going through this unit, you should be able to:

- explain development of computer graphics
- describe basic graphic systems
- discuss about display processors and character generators.

NOTES

1.1 OVERVIEW OF COMPUTER GRAPHICS

Introduction

Computer technology has progressed rapidly over the years. Starting from machine languages, we are into the world of high level languages like C, C++ and Java. Keeping pace with these developments is the field of Computer Graphics.

Computer Graphics is the field of Computer Science in which we are interested in generating objects and images using pixels (picture elements). It started with elementary programs like point, line and circle generation. Earlier Graphics systems used primitives to generate simple graphics. But with the development of interactive input and output devices and the technology, it is possible virtually to design anything on the computer system. Animation and cartoons have reached high quality with the use of Advanced Graphics. The scene showing Titanic sinking in the sea was possible through Graphics designing with Maya. The movies like Spiderman, Superman, Hanuman etc. have generated special effects using Computer Graphics.

Multimedia, CAD and CAM have all been possible because of Computer Graphics.

According to computer terminology, computer program is the collection of algorithm and data structure. In the same convention Computer Graphics has been defined as the collection of data structure, graphic algorithms and any higher level language. We can use any of the higher level language like C, C++, Java or any of the Graphics packages like Maya, 3-D Max or Flash to generate graphics of our choice.

Simplest application of graphics is to plot a point (pixel) on the screen of any resolution (VGA or SVGA) or to plot a line or plot a circle. Algorithm are already defined and given by the scientists/researchers and students are just required to implement those algorithms in the programming language of their choice.

Computer Graphics need a logical mind and mastery over a programming language to succeed. This field is becoming popular day by day and there are a larger number of opportunities to be explored. There are different types of Computer Graphics like Passive and Interactive Graphics.

Computer Graphic is the discipline of producing picture or images using a computer including modeling – creation, manipulation, and storage of geometric objects and rendering – converting a scene to an image, or the process of transformations, rasterization, shading, illumination, and animation of the image.

Computer Graphics has been widely used, such as graphics presentation, paint systems, computer-aided design (CAD), image processing, simulation and virtual reality, and entertainment. From the earliest text character images of a non-graphic mainframe computers to the latest photographic quality images of a high resolution personal computers, from vector displays to raster displays, from 2D input, to 3D input and beyond, Computer Graphics has gone through its short, rapid changing history.

1.2 DEVELOPMENT OF COMPUTER GRAPHICS

PreHistory

The foundations of Computer Graphics can be traced to artistic and Mathematical "inventions," for example,

- Euclid (circa 300 – 250 BC) who's formulation of geometry provides a basis for Graphics concepts.

NOTES

- Filippo Brunelleschi (1377 – 1446) architect, goldsmith, and sculptor who is noted for his use of perspective.
- Rene Descartés (1596 – 1650) who developed analytic geometry, in particular coordinate systems which provide a foundation for describing the location and shape of objects in space.
- Gottfried Wilhelm Leibniz (1646 – 1716) and Issac Newton (1642 – 1727) who co-invented calculus that allow us to describe dynamical systems.
- James Joseph Sylvester (1814 – 1897) who invented matrix notation. A lot of graphics can be done with matrices.
- Schoenberg who discovered splines, a fundamental type of curve.
- J. Presper Mauchly (1919 – 1995) and John William Mauchly (1907 – 1980) who build the ENIAC computer.

Early History

I would like to date the history of Computer Graphics from the Whirlwind Project and the SAGE computer system, which were designed to support military preparedness. The Whirlwind Project started as an effort to build a flight simulator and SAGE was to provide an air defense system in the United States to guard against the threat of a nuclear attack. The SAGE workstation had a vector display and light pens that operators would use pinpoint planes flying over regions of the United States. You can see a SAGE workstation at the Boston Computer Museum. The display is a large radar screen with a wire frame outline of the region being scanned. The light pens are like old large metal drills. A SAGE computer is on display on the corner of "Hollywood and Vine" at IBM in Kingston New York.

Besides the being the age of the first vacuum tube computers, the 1940's was when the transistor was invented at Bell Labs (1947). In 1956, the first transistorized computer was built at MIT.

The Age of Sutherland

In the early 1960's IBM, Sperry-Rand, Burroughs and a few other computer companies existed. The computers of the day had a few kilobytes of memory, no operating systems to speak of and no graphical display monitors. The peripherals were Hollerith punch cards, line printers, and roll-paper plotters. The only programming languages supported were assembler, FORTRAN, and Algol. Function graphs and "Snoopy" calendars were about the only graphics done.

In 1963 Ivan Sutherland presented his paper ***Sketchpad*** at the Summer Joint Computer Conference. Sketchpad allowed interactive design on a vector graphics display monitor with a light pen input device. Most people mark this event as the origins of Computer Graphics.

The Middle to Late '60's

Software and Algorithms

Jack Bresenham taught us how to draw lines on a raster device. He later extended this to circles. Anti-aliased lines and curve drawing is a major topic in Computer Graphics. Larry Roberts pointed out the usefulness of homogeneous coordinates, 4×4 matrices and hidden line detection algorithms. Steve Coons introduced parametric surfaces and developed early computer aided geometric design concepts. The earlier work of Pierre Bézier on parametric curves and surfaces also became public. Author Appel at IBM developed hidden surface and shadow algorithms that were pre-cursors to ray tracing.

NOTES

The fast Fourier transform was discovered by Cooley and Tukey. This algorithm allows us to better understand signals and is fundamental for developing antialiasing techniques. It is also a precursor to wavelets.

Hardware and Technology

Doug Englebart invented the mouse at Xerox PARC. The Evans and Sutherland Corporation and General Electric started building flight simulators with real-time raster graphics. The floppy disk was invented at IBM and the microprocessor was invented at Intel. The concept of a research network, the ARPANET, was developed.

The Early '70's

The state of the art in computing was an IBM 360 computer with about 64 KB of memory, a Tektronix 4014 storage tube, or a vector display with a light pen (but these were very expensive).

Software and Algorithms

Rendering (shading) were discovered by Gouraud and Phong at the University of Utah. Phong also introduced a reflection model that included specular highlights. Keyframe based animation for 3-D graphics was demonstrated. Xerox PARC developed a "paint" program. Ed Catmull introduced parametric patch rendering, the z-buffer algorithm, and texture mapping. BASIC, C, and Unix were developed at Dartmouth and Bell Labs.

Hardware and Technology

An Evans and Sutherland Picture System was the high-end graphics computer. It was a vector display with hardware support for clipping and perspective. Xerox PARC introduced the Altos personal computer, and an 8 bit computer was invented at Intel.

The Middle to Late '70's

Software and Algorithms

Turned Whitted developed recursive ray tracing and it became the standard for photorealism, living in a pristine world. Pascal was the programming language everyone learned.

Hardware and Technology

The Apple I and II computers became the first commercial successes for personal computing. The DEC VAX computer was the mainframe (mini) computer of choice. Arcade games such as Pong and Pac Mac became popular. Laser printers were invented at Xerox PARC.

The Early '80's

Hardware and Technology

The IBM PC was marketed in 1981 The Apple Macintosh started production in 1984, and microprocessors began to take off, with the Intel x86 chipset, but these were still toys. Computers with a mouse, bitmapped (raster) display, and Ethernet became the standard in academic and Science and Engineering settings.

The Middle to Late '80's

Software and Algorithms

Jim Blinn introduces blobby models and texture mapping concepts. Binary Space Partitioning (BSP) trees were introduced as a data structure, but not many realized how useful they would become. Loren Carpenter started exploring fractals in computer graphics. Postscript was developed by John Warnock and Adobe was formed. Steve Cook introduced stochastic sampling to ray tracing. Paul Heckbert taught us to ray trace. Character animation became the goal for animators. Radiosity was introduced by the Greenberg and folks at Cornell. Photoshop was marketed by Adobe. Video arcade games took off, many people/organizations started publishing on the desktop. UNIX and X windows were the platforms of choice with programming in C and C++, but MS-DOS was starting to rise.

NOTES

Hardware and Technology

Sun workstations, with the Motorola 680x0 chipset became popular as advanced workstation a in the mid 80's. The Video Graphics Array (VGA) card was invented at IBM. Silicon Graphics (SGI) workstations that supported real-time raster line drawing and later polygons became the computer graphicists desired. The data glove, a precursor to virtual reality, was invented at NASA. VLSI for special purpose graphics processors and parallel processing became hot research areas.

The Early '90's

The Computer to have now was an SGI workstation with at least 16 MB of memory, at 24-bit raster display with hardware support for Gouraud shading and z-buffering for hidden surface removal. Laser printers and single frame video recorders were standard. UNIX, X and Silicon Graphics GL were the operating systems, window system and Application Programming Interface (API) that Graphicist used. Shaded raster graphics were starting to be introduced in motion pictures. PCs started to get decent, but still they could not support 3-D graphics, so most programmer's wrote software for scan conversion (rasterization) used the painter's algorithm for hidden surface removal, and developed "tricks" for real-time animation.

Software and Algorithms

Mosaic, the first graphical Internet browser was written by xxx at the University of Illinois, National Center for Scientific Applications (NCSA). MPEG standards for compressed video began to be promulgated. Dynamical systems (physically based modeling) that allowed animation with collisions, gravity, friction, and cause and effects were introduced. In 1992 OpenGL became the standard for graphics APIs In 1993, the World Wide Web took off. Surface subdivision algorithms were rediscovered. Wavelets begin to be used in Computer Graphics.

Hardware and Technology

Hand-held computers were invented at Hewlett-Packard about 1991. Zip drives were invented at Iomega. The Intel 486 chipset allowed PC to get reasonable floating point performance. In 1994, Silicon Graphics produced the Reality Engine: It had hardware for real-time texture mapping. The Ninetendo 64 game console hit the market providing Reality Engine-like graphics for the masses of games players. Scanners were introduced.

The Middle to Late '90's

The PC market erupts and supercomputers begin to wane. Microsoft grows, Apple collapses, but begins to come back, SGI collapses, and lots of new startups enter the graphics field.

Software and Algorithms

NOTES

Image based rendering became the area for research in photo-realistic graphics. Linux and open source software become popular.

Hardware and Technology

PC graphics cards, for example 3dfx and NVIDIA, were introduced. Laptops were introduced to the market. The Pentium chipset makes PCs almost as powerful as workstations. Motion capture, begun with the data glove, becomes a primary method for generating animation sequences. 3-D video games become very popular: DOOM (which uses BSP trees), Quake, Mario Brothers etc. Graphics effects in movies become pervasive: Terminator 2, Jurassic Park, Toy Story, Titanic, Star Wars I. Virtual reality and the Virtual Reality Meta (Markup) Language (VRML) become hot areas for research. PDA's, the Palm Pilot, and flat panel displays hit the market.

The '00's

Today most graphicists want an Intel PC with at least 256 MB of memory and a 10 GB hard drive. Their display should have graphics board that supports real-time texture mapping. A flatbed scanner, color laser printer, digital video camera, DVD, and MPEG encoder/decoder are the peripherals one wants. The environment for program development is most likely Windows and Linux, with Direct 3D and OpenGL, but Java 3D might become more important. Programs would typically be written in C++ or Java.

What will happen in the near future — difficult to say, but high definition TV (HDTV) is poised to take off (after years of hype). Ubiquitous, untethered, wireless computing should become widespread, and audio and gestural input devices should replace some of the functionality of the keyboard and mouse.

You should expect 3-D modeling and video editing for the masses, computer vision for robotic devices and capture facial expressions, and realistic rendering of difficult things like a human face, hair, and water. With any luck C++ will fall out of favor.

Ethical Issues

Graphics has had a tremendous affect on society. Things that affect society often lead to ethical and legal issues. For example, graphics are used in battles and their simulation, medical diagnosis, crime re-enactment, cartoons and films. The ethical role played by a Computer Graphicist in the use of graphics programs that may be used for these and other purposes is discussed and analyzed in the notes on Ethics.

More History of Computer Graphics

In the 1950's, output were via teletypes, line printer, and Cathode Ray Tube (CRT). Using dark and light characters, a picture could be reproduced.

1950: Ben Laposky created the first graphic images, an Oscilloscope, generated by an electronic (analog) machine. The image was produced by manipulating electronic beams and recording them onto high-speed film.

1951: UNIVAC-I: the first general purpose commercial computer, crude hardcopy devices, and line printer pictures.

1951: MIT – Whirlwind computer, the first to display real time video, and capable of displaying real time text and graphic on a large oscilloscope screen.

In the **1960's**, beginnings of modern interactive graphics, output are vector graphics and interactive graphics. One of the worst problems was the cost and inaccessibility of machines.

1960: William Fetter coins the Computer Graphics to describe new design methods.

1961: Steve Russel — Spacewars, first video/computer game

NOTES

1963: Douglas Englebart – first mouse

Ivan Sutherland – Sketchpad, interactive CG system, a man-machine graphical communication system, it features:

- pop-up menus
- constraint-based drawing
- hierarchical modeling
- utilized light pen for interaction

He formulated the ideas of using primitives, lines polygons, arcs, etc. and constraints on them; He developed the dragging, rubber banding and transforming algorithms; He introduced data structures for storing. He is considered the **founder of the Computer Graphics.**

1964: William Fetter — first computer model of a human figure.

1965: Jack Bresenham – line-drawing algorithm

1968: Tektronix – a special CRT, the direct-view storage tube, with keyboard and mouse, a simple computer interface for $15, 000, which made graphics affordable

Ivan Sutherland – first head-mounted display

1969: John Warnock – area subdivision algorithm, hidden-surface algorithms

Bell Labs – first frame buffer containing 3 bits per pixel

In the early **1970's**, output start using raster displays, graphics capability was still fairly chunky.

1972: Nolan Kay Bushnell – Pong, video arcade game

1973: John Whitney, Jr. and Gary Demos – "Westworld", first film with computer graphics

1974: Edwin Catmuff –texture mapping and Z-buffer hidden-surface algorithm

James Blinn – curved surfaces, refinement of texture mapping

Phone Bui-Toung – specular highlighting

1975: Martin Newell – famous CG teapot, using Bezier patches

Benoit Mandelbrot – fractal/fractional dimension

1976: James Blinn – environment mapping and bump mapping

1977: Steve Wozniak — Apple II, color graphics personal computer

1979: Roy Trubshaw and Richard Bartle – MUD, a multi-user dungeon/Zork

In the **1980's** output are built-in raster graphics, bitmap image and pixel. Personal computers costs decrease drastically; trackball and mouse become the standard interactive devices

1982: Steven Lisberger – "Tron", first Disney movie which makes extensive use of 3-D Computer Graphics

NOTES

Tom Brighman – "Morphing", first film sequence plays a female character which deforms and transforms herself into the shape of a lynx.

John Walkner and Dan Drake – AutoCAD

1983: Jaron Lanier – "DataGlove", a virtual reality film features a glove installed with switches and sensors to detect hand motion

1984: Wavefron tech. – Polhemus, first 3D graphics software

1985: Pixar Animation Studios – "Luxo Jr.", 1989, " Tin toy"

NES – Nintendo home game system

1987: IBM – VGA, Video Graphics Array introduced

1989: Video Electronics Standards Association (VESA) – SVGA, Super VGA formed

In the **1990's**, since the introduction of VGA and SVGA, personal computer could easily display photo-realistic images and movies. 3D image renderings are become the main advances and it stimulated cinematic graphics applications.

1990: Hanrahan and Lawson – Renderman

1991: Disney and Pixar – "Beauty and the Beast", CGI was widely used, Renderman systems provides fast, accurate and high quality digital computer effects

1992: Silicon Graphics – OpenGL specification

1993: University of Illinois — Mosaic, first graphic Web browser

Steven Spielberg – "Jurassic Park" a successful CG fiction film

1995: Buena Vista Pictures – "Toy Story", first full-length, computer-generated, feature film

NVIDIA Corporation – GeForce 256, GeForce3(2001)

2003: ID Software – Doom graphics engine

1.3 CLASSIFICATION OF COMPUTER GRAPHICS

This is also called types of Computer Graphics. Computer Graphics has been classified into two categories according to the application domain and requirements. They are Passive and Interactive Computer Graphics.

1. **Passive Computer Graphics:** This is also called Off-line graphics. Once the graphics program is developed, the user has no control over the display even if he wishes to change the display. Development takes place independently in Off-line mode. Example of Passive Graphics can be compared to static web page (Using HTML) where user has no control over the contents on the monitor. The screen will be static with no marquees flying on the screen or no advertisements changing or no color change on mouse movements etc.

   The displays are generally calligraphic in nature and are printed by line printers, plotters etc.

2. **Interactive Computer Graphics:** This is also called On-line graphics. User can dynamically control the display on the monitor. Displays are controlled by mouse, trackball, Joystick etc. This is termed as Interactive Computer Graphics (ICG) because the user can interact with the machine as per his requirements. ICG is the case in which user can dynamically control the picture contents, formats, sizes or the color on the display surface by means of interaction devices such as keyboards, light pen, mouse etc.

Cartoons, special effects in movies (remember Dinosaurs of Jurassic Park), video games and dynamic web sites(Using dynamic HTML) are all making use of ICG. ICG affects our lives in a number of indirect ways. For example

(*i*) Flight simulators helps us to train the pilots of our airplanes. These pilots spend a lot of time for their training not in a real aircraft but on the ground at the controls of a flight simulator. Flight simulator has many advantages over real aircraft for training purposes including safety, fuel saving and ability to familiarize the trainee with different aspects and technology of aircraft.

(*ii*) Architects can explore alternative solutions to design problem with Interactive Graphics. Design programs like CAD help designers, engineers, draftsmen etc. to complete the tasks associated with hand drafting of designs easily and more efficiently.

(*iii*) Microprocessor based video-game processors attached to home TV's.

(*iv*) Cartoons, Animation and special effects in movies and TV serials are all possible with the use of computer graphics.

1.4 BASIC GRAPHIC SYSTEMS

Interactive Computer Graphics consists of three components namely digital memory buffer, TV monitor and display controller. Using these components, we are able to see the output on the screen in form of pixels(picture elements). Following is the explanation of these components:

1. **Digital Memory Buffer:** This is the place where images or pictures are stored as an array (matrix of 0 and 1, 0 represents darkness and 1 represents image or picture). This is also called Frame Buffer. In today's terms, frame buffer is called V-RAM (Video RAM) and it helps to store the image in bit form. It helps to increase the speed of graphics (Sometimes we watch movies on our Computer system and the movie run slowly. System engineer is then called for. He comes and fits in V-RAM(in Mega bytes) into our system and the movie runs perfectly).
2. **TV Monitor:** Monitor helps us to view the display and they make use of CRT technology (Cathode Ray Tube) will be discussed in Output devices. Raster scan and random scan monitors are also discussed in graphics devices section.
3. **Display Controller:** It is an interface between Digital Memory Buffer and TV Monitor. Its job is to pass the contents of Frame buffer to the monitor. This passing has to be fast for steady display on the monitor (depending on the material of the system). The image must be passed repeatedly to the monitor to maintain a steady picture on the screen. The display controller reads each successive byte of data from the FB memory and converts 0's and 1's into corresponding video signal. This signal is then fed to the TV monitor to produce a black and white picture on the screen. In today's terms, display controller is recognized as display card and one of our choices can be VGA card with a resolution of 640 × 480. (Display controllers are also capable of displaying image in colors and will be discussed in colored display section).

Several graphics systems have been designed this way.

Fig. 1.1 Frame buffer (bit Pattern of 0 and 1)

NOTES

Throughout the book certain questions will again and again be asked or encountered and they are very important to be discussed right now. The questions are like:

1. How to display straight lines and curved lines?
2. Why speed is important for graphics displays?
3. How pictures are made to grow or shrink on the screen?
4. What happens if the picture is too large to fit in the screen?
5. Can we directly draw on screen?
6. How is analog form of data converted into digital form so that the computer system can understand it?

My Professor would ask these questions regularly in class and once these questions were clear to us, graphics became an easy subject for us.

Ans 1. Straight lines can be drawn using algorithms that have to be implemented in any programming language. There are two methods of drawing lines (as we will study later) DDA(Digital Differential Analyzer) and Bresenham's line algorithms. These algorithms have been used to generate lines.

Curves can be drawn by breaking them into several straight lines. Then the algorithm of straight line drawing can be applied to these straight lines. Curves can also be generated if function of the curve is known or curve fitting techniques are also used to generate curves.

Ans 2. Speed is important because we are talking of Interactive graphics and in this type of graphics, speed of transfer of entire picture contents from FB to monitor must very fast Also we need fast responses in ICG so speed is important. A slow graphics system will not be acceptable even to a small child who would want his cartoons to be at the best possible speed. A powerful processor can do the trick aided with additional V-RAM in the system.

Ans 3. Pictures can be made to grow and shrink according to our needs using Mathematical techniques like

- Coordinate Geometry
- Matrix method
- Trigonometric functions

Using these techniques, various 2-D or 3-D transformations are possible. They are:

1. Changing the scale (scaling to magnify or shrink)
2. Shifting the position (translating to move to new position on the screen)
3. Rotation (rotating the object by some angle)

These transformations are achieved through equations implemented in algorithms and hardware. Transformation like shearing and reflection are also important in Computer Graphics.

Programs for scaling, translation and rotation are given in this book.

Ans 4. It sometimes happens that image is too large for the screen. We will study the concepts of Clipping and Windowing in this regard. Clipping is the process in which the big whole picture is chopped into small portions and only a portion inside the window is displayed. Clipping divide the picture into visible and invisible parts and rejects invisible parts and displays them.

We will study Cohen Sutherland and Sutherland Hodgeman algorithms for Clipping of lines and polygons respectively.

Ans 5. Yes, we can directly draw on screen. Haven't we seen cricket live coverage and seen Geoffrey Boycott or Sunil Gavasker or Ravi Shastri drawing directly on the screen when they mark the fielder or the trajectory of the ball. Devices like light pen, mouse, tablets etc. helps us to draw on the screen directly.

Light pen, mouse or tablets will be covered later in Interactive devices.

Ans 6. Analog to digital conversion has to be performed since the computer is a discrete digital device, while most data in the world outside the computer is continuous. The data must be therefore converted to enable the computer to use it. The conversion of continuous analog data to discrete data is called analog to digital conversion or simply A/D conversion.

The computer graphics programmer must have some knowledge of A/D conversion because many graphics input devices are analog devices and A/D conversion can affect the accuracy of data collected even while exhibiting a high degree of degree of precision. Typically, A/D conversion is performed either by hardware devices incorporated in the computer's input circuitry or by means of firmware that directly interfaces with computer's input circuitry.

1.5 GRAPHICS PRIMITIVES

Regardless of the differences in display devices, most graphic systems offer a similar set of graphic primitive commands (although the form and syntax may differ from system to system). The first primitive command is that for drawing a line segment. These primitives are widely used in GKS and CORE graphics packages. While a segment may be specified by its two endpoints, it is often the case that the segments drawn will be connected end to end. Primitive commands were very important when Random scan was being in graphics systems but now in Raster scan devices, primitives are no longer used.

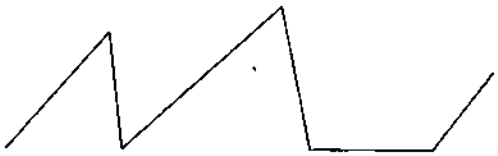


Fig. 1.2

LINEABS (X, Y) is called an absolute line command because the actual coordinates of the final positions are passed. Most of the graphics packages offer this line primitive. Even in C language, definition of this primitive has been picked up. **Lineto(*x*, *y*)** in C language refers to LINEABS(*x*, *y*) only. This command is used to pass the final coordinates of the point.

LINEREL (DX, DY) is called relative line command because in this command we only indicate how far to move from the current position.

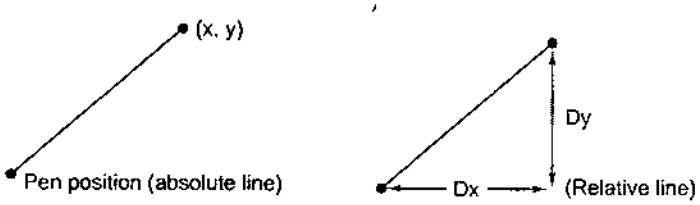


Fig. 1.3

We have two more commands MOVEABS (X, Y) and MOVEREL (DX, DY) and we can construct a line drawing by series of these commands. If these commands are located in a function then each time a function is called, an image of the object is produced. If

NOTES

absolute commands are used then the house will always be located at the same position on the screen.

As discussed above. C language supports the following functions **lineto ()**

For example **lineto(1, 1)** draws a line from current pen position to (1, 1).

In C language, there are built –in functions like :

linerel (Int *x*, Int *y*)

moverel (Int *x*, Int *y*)

setlinestyle ()

And their definitions have been taken from Graphics primitives.

Graphics primitive commands are stored in **Display file**. Display file is a file storing all primitive commands and when this file is interpreted, visual image is displayed on the screen. We need to study normalized coordinates, display file and interpreter for a better understanding of the system.

1.6 NORMALISED DEVICE COORDINATES

Different display devices may have different screen sizes as measured in pixels. If we wish our program to be device independent, we should specify coordinates in some units other than pixel and use the interpreter to convert the coordinates to the appropriate pixel for the particular display we are using.

Device independent units are called normalized device coordinates.

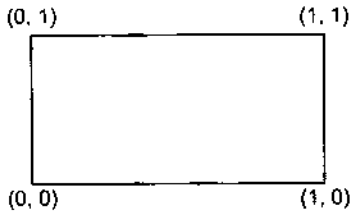


Fig. 1.4

Screen measure 1 unit wide, 1 unit height

1.7 DISPLAY FILE AND ITS STRUCTURE

Each display file command contains two constituents:

1. Operation code (OPCODE) which indicates what kind of command it is (Example Line, move)
2. Operands which are the coordinates of a point (*x*, *y*).

Display is made up of a series of these instructions.

One possible method for storing these instructions is to use three separate arrays : one for operation code (DF – OP), one for *x* coordinate (DF – X) and one for y coordinate (DF – Y).

MOVE and LINE are two possible instructions we have already used. Let us define an OPCODE of 1 to mean a move command and an OPCODE of 2 to mean LINE command. Then a command to move to X = 0.2 and X = 0.8 would look like (1, 0.2, 0.8). The statements

DF – OP[2] = 1;
DF – X [2] = 0.2;

DF – Y[2] = 0.8;

will store this instruction in second display file position.

DF – OP	DF – X	DF – Y
1	0.2	0.8

1.8 THE DISPLAY FILE INTERPRETER

NOTES

Line and Move commands store their information in display file. We then use information in the file to create the image. The display file will contain the necessary information to construct the picture. Saving instructions such as these usually takes less storage than saving the picture itself.

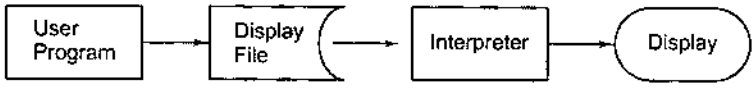


Fig. 1.5

Graphic file interpreter serves an interface between our graphic program and the display device. Interpreter may be thought of as a machine which executes these instructions. The result is a visual image.

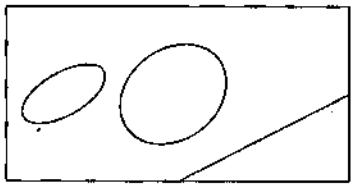


Fig. 1.6

There are many more Graphics primitive commands other than Line and Move. They are Text, Linestyle and polygon primitives.

1.9 TEXT PRIMITIVE

Another primitive operation is that of text output. Most graphic displays involve textual as well as graphical output data, labels (giving scale values etc.), instructions, commands, values, messages and so on may be presented with the image.

The primitive command involved here is the output of a character or a string of characters. The character themselves may be drawn by either the dot matrix or the stroke method. Their patterns are copied from memory into frame buffer or created by special character generation hardware.

Ex. Text ('My first graphic program') would output this on screen.

1.10 LINE STYLE PRIMITIVE

Many display devices offer a selection of line style. Lines may be continuous or they may be dashed or dotted. One may be able to select the color of the line or its intensity or thickness. It is desirable to be able to change the line style in the middle of the display process. We therefore need a display file command for changing line style. When the interpreter encounters such a command, the line style is changed and all subsequent lines are drawn in this new style.

For example, SETLINESTYLE (2) would give us broken line

SETLINESTYLE (0) would give us solid line.

Other options are

0. Solid line
1. Dotted line
2. Broken line
3. Dashed line

NOTES

1.11 POLYGON PRIMITIVE

So far discussion has dealt with only the lines. The world might seem rather dull if it were made out of straight lines. How interesting is the world of patterns, colors and shading. Unfortunately, much of the early graphics dealt with line drawings only. This was because the available devices (DVST, Plotters etc.) were line drawing devices. Raster display can display solid patterns and objects with no greater effort than that involved in showing their outlines. Coloring and shading is possible with raster technology.

We introduce new graphic primitive, the Polygon. We shall discuss what polygons are and how to represent them?

We wish to be able to represent a surface. One basic surface primitive is a polygon, a many sided figure. A polygon may be represented as a number of line segments connected end to end to form a closed figure. The line segments that make up the polygon boundary are called sides or edges. The endpoints of the sides are called the polygon's vertices. The simplest polygon is the triangle, having three sides and three vertex points.

Polygon can be divided into two classes: Concave and Convex. A Convex polygon is a polygon such that for any two points inside the polygon, all points on the line segment connecting them are also inside the Polygon. A concave polygon is the one which is not convex. A triangle is always convex and so are the shapes shown below.

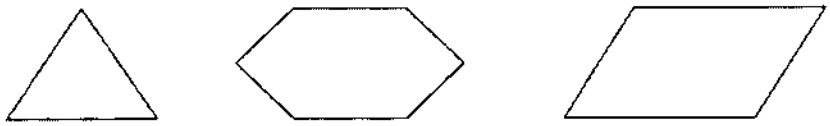


Fig. 1.7

Some concave polygon shapes are shown below:

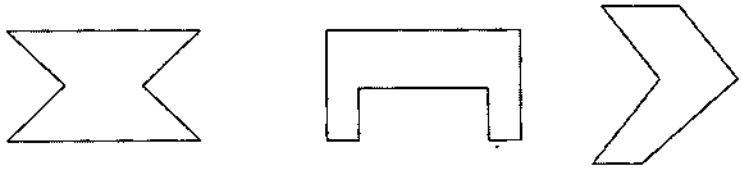


Fig. 1.8

1.12 POLYGON REPRESENTATION

If we want to add polygons to our graphics system, we must first decide how to represent them.

Some graphics, devices have polygon drawing primitive to directly draw polygon shape. There is trapezoid primitive also and a polygon can be drawn as a series of trapezoids.

Now the question is what should a polygon look like in a display file?

We introduce a new command into the display file. This new command will tell us how many sides are in the polygon so that we'll know how many of the line commands are part of the polygon. Upon interpretation this new command will act like a move to correctly position the pen for drawing the first side.

Let's talk about the Operation code 3 or greater. We can use these codes to indicate polygons. The value of the operation code will indicate the number of sides in a polygon. The X and Y variables of the polygon will be the coordinates of the point where the first side to be drawn begins. Polygons are closed figures so it will also be the final endpoint of the last side to be drawn.

Let's work out an example of a polygon given below and look at its display file

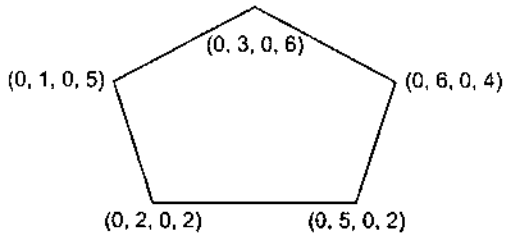


Fig. 1.9

The display file for the given polygon is

DF-OP	DF-X	DF-Y
5	0.2	0.2
2	0.5	0.2
2	0.6	0.4
2	0.3	0.6
2	0.1	0.5
2	0.2	0.2

1.13 ENTERING POLYGON IN DISPLAY FILE

When we say entering polygons we mean algorithms for entering polygons into display file. Information required to be specified in polygon case is number of sides and the coordinates of vertex points of a polygon to be drawn. Arrays are used to pass the coordinates of all vertices to the routine.

Following is the algorithm for entering the polygon into the display file using absolute commands.

1.14 ALGORITHM

Algorithm PolyAbs for entry of an absolute polygon into the display file Variables AB, AC are arrays containing vertices.

Global variables penX, penY the current pen position.
Local variable I for stepping through the polygon sides.
Start
{
if N < 3 then give error 'Polygon size error'

NOTES

```
//*Enter polygon instructions
penX = AB[N] ;
penY = AC[N] ;
// *Enter information in display file
Display-file-enter(N);
//*Enter the instructions for the sides
For I = 1 to N do Lineabs(AB[I],AC[I]) ;
Return,
End;
}
```

Similarly programs can be developed using relative line commands to represent polygons. C language has got functions to support the use of absolute and relative line commands.

CORE and GKS Graphics Systems are some of the market prevalent products and support these primitives operations.

Graphics Kernel

The Graphical Kernel System (GKS) was the first ISO standard for low level computer graphics, introduced in 1977. GKS provides a set of low-level drawing features for two-dimensional line and vector graphics. The calls are designed to be portable across different programming languages, graphics devices and uses, so that applications written to use GKS will be readily portable to many platforms and devices.

A main developer and promoter of the GKS was professor José Luis Encarnação, formerly director of the Fraunhofer Institute for Computer Graphics (IGD) in Darmstadt, Germany.

1.15 GRAPHICAL INPUT/OUTPUT DEVICE

Graphical Interactive Input Devices

Data has to be passed on to the computer system so that it can be processed and presented to the user in some format. Same is the case with Computer graphics. Data has to be entered into the system via some input device (Keyboard, mouse, light pen etc.) so that it can be manipulated by the programs and images presented to the user on some output device (Monitor or printer).

Any device that allows information from outside the computer to be communicated to the computer is considered an input device. Some common graphical input devices are:

ALPHANUMERIC KEYBOARDS: This is an input device. A standard known as ASCII has been developed to allow computer to encode keyboard characters. Whatever we key in is translated by the keyboard translation program in a machine readable form so that it can be understood by the system.

TRACK BALLS: A track ball uses a hard sphere to control cursor movement. The ball can be rotated by hand in any direction. The trackball translates the sphere's direction and speed of rotation into a digital signal used to control the cursor.

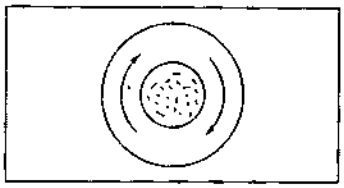


Fig. 1.10

NOTES

JOYSTICK: It is the most important input device used to play video games. A joystick uses a lever to control the position and speed with which the joystick is moved. This gets translated into digital signals that are sent to the computer to control the cursor movement.

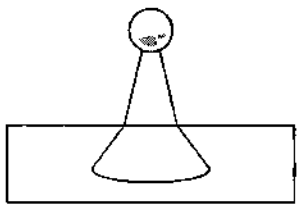


Fig. 1.11

MOUSE: DOUG ENGELBART developed mouse as an alternative to keyboard in 1960's. Mouse has three buttons. When user presses one of the buttons, the mouse marks the place on the screen. It has a ball that is pressed against he mouse pad.

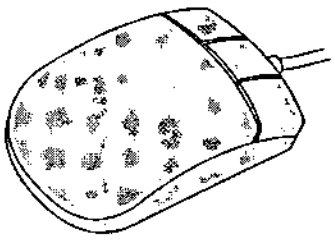


Fig. 1.12

The joystick, trackball, mouse etc. are transducers that convert a graphic system user's movements into changes in voltage. **Transducer** is a device that converts energy from one form to other.

LIGHT PEN: It is a pointing device that can be used to choose a displayed menu option. The pen consists of a photocell placed in a small tube. Light pen is very useful in graphics. The user at a CAD terminal can draw directly on the screen with the light pen.

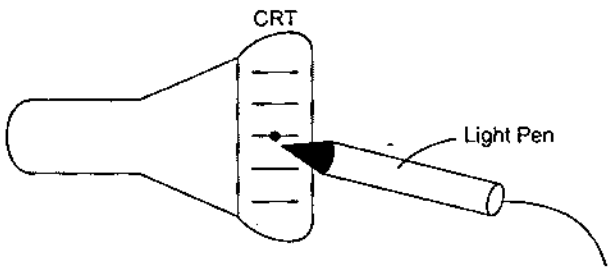


Fig. 1.13

1.16 GRAPHICS TABLET (DIGITISING TABLET, DIGITISER)

The graphics tablet is a pressure sensitive, flat electronic board which takes input from a pen or puck. The board or tablet is usually fairly large (35cm × 35cm +) and is very high

NOTES

resolution. It is much like drawing on a piece of paper except that the drawing appears on the computer screen rather than on the tablet. The position of the pen or puck is detected by the computer and the *x-y* coordinates transmitted to the screen. Some models are capable of making wider or narrower lines depending on whether greater or lesser force is applied to the pen. More advanced models also often have macro buttons and other buttons that duplicate menu bar commands.

This device is useful for people who don't know how to type or for artists etc. who want to draw as they would on paper. Drawings can be placed on the tablet and then traced onto the screen. Digitizers, are based on the principle of an electric or magnetic field that is generated through one component (the pen) and detected through the other component (the tablet). There are several types:

1. Touch-sensitive — consists of a pen and a touch sensitive surface using a matrix of wires in the tablet
2. Sonic digitizers — sound impulses detected by two microphone sensors on the tablet
3. Light-detector digitizer — detects interruptions in a matrix of light beams
4. Magnetic-field sensors — detects changes in the magnetic field.

Digitizer or digitizing tablets are flat boards that have a grid of interconnecting wires just below the surface. They also have either an electronic 'pen' or 'puck' that can be moved round the surface. The pen stylus is more natural, but the puck offer the advantage of having upto four or more buttons that can be programmed to perform certain tasks. The electronic circuitry of the digitizing tablet can tell exactly where you are holding the pen in relation to the rest of the board by transmitting pulses between the pen and the wired grid.

The computer then translates the location of the pen on the board into numerical coordinates that represents *x* and *y* values on the screen. The *x* value represents the horizontal position and the *y* value represents the vertical position. The resolution tablets range from 1000 to 10000 lines per inch, giving very accurate position to the computer. Both the puck or pen stylus has buttons that can be pressed to indicate a particular command to the computer.

What makes tablets popular as pointing devices is the absolute positioning of the cursor. With a tablet, there is a direct relationship to the movement on screen, so we can place the puck or pen on the lower right corner of the tablet, the cursor appears on the lower right corner of the screen. The only disadvantage to digitizing tablets is the desktop space they require. Also they cannot be used to tracing natural pictures into the computers.

TOUCH SCREEN/PANELS: These are devices which allow the user to directly activate data by touching the screen. Touch is a more intuitive interface for users who are not familiar with computers. In public applications, users have little training in the technology and are simply instructed to 'touch your selection.' This has wide user appeal. They have many applications in museums, shops, restaurants, street maps etc. as touch systems are best used for selection of pre-programmed information.

It is also possible to buy touch screen monitors and touch-screen add-on kits for PCs - useful for young children and the physically handicapped people. Touch sensitive screen can work in several ways:

1. The finger can interrupt a light beam
2. The pad can detect changes in capacitance.

A different GUI is needed - bigger and perhaps needs to be made more iconic. The drawbacks are that moving the arm may become tiring if they are used for a long time, the screen may become dirty and when the hand is being used to select part of the screen

other parts may be obscured. The advantages are an intuitive interface, flexibility in the information provided, an ability to include graphics, signs and symbols and usually a rugged and reliable interface.

As the name implies, touch panels allow displayed objects or screen positions to be selected with the touch of a finger. A typical application of touch panels is for the selection of processing options that are implemented with graphical icons. Some systems, such as the plasma panels are designed with touch screens. Other systems can be adapted for touch input by fitting a transparent device with a touch sensing mechanism over the video monitor screen. Touch input can be recorded using optical, electrical, or acoustical methods. Optical touch panels employ a line of infrared Light Emitting Diodes (LEDs) along one vertical edge and along one horizontal edge of the frame. The opposite vertical and horizontal edges contain light detectors: These detectors are used to record which beams are interrupted when the panel is touched. The two crossing beams that are interrupted identify the horizontal and vertical coordinates of the screen position selected.

NOTES

Positions can be selected with an accuracy of about 1/4 inch. With closely spaced LEDs, it is possible to draw two horizontal or two vertical beams simultaneously. In this case, an average position between the two interrupted beams is recorded. The LEDs operate at infrared frequencies, so that the light is not visible to a user.

An electrical touch panel is constructed with two transparent plates separated by a small distance. One of the plates is coated with a conducting material, and the other plate is coated with a resistive material. When the outer plate is touched, it is fed into contact with the inner plate. This contact cracks a voltage drop across the massive plate that is converted to the coordinate values of the selected *screen* position. In acoustical touch panels, high-frequency sound waves are generated in the horizontal and vertical directions across a glass plate. Touching the screen causes part of each wave to be reflected from the finger to the emitters. The screen position at the point of contact is calculated from a measurement of the time interval between the transmission of each wave and its reflection to the emitter.

1.17 IMAGE SCANNER

The most common device used to input images into a computer is a scanner. Scanners are similar to copy machines, except they store the image electronically instead of transferring it into another piece of paper. There are many different type of scanners from hand held units that look like a mouse to drum scanners that can fill a entire room.

Most scanners expose the image to a bright light. Then electronic receptors called Charged Coupled Device (CCD) sensors pick up that light and convert it to electronic pulses. Those pulses get translated to the numbers. There are black and white as well as color scanners. The resolution of scanners ranges from 100 dpi to 2000 dpi and beyond.

Voice Systems

Speech recognizers are used in some graphics workstations as input devices to accept voice commands. The voice system input can be used to initiate graphics operations or to enter data. These systems operate by matching an input against a predefined dictionary of words and phrase called the vocabulary of the system.

A dictionary (vocabulary) is set up for a particular operator by having the operator speak the command words to be used into the system. Each word is spoken several times and the system analyzes the word and establishes a frequency pattern for that word in the dictionary along with the corresponding function to be performed.

NOTES

Later, when a voice command is given, the system searches the dictionary for a frequency-pattern match. Voice input is typically spoken into a microphone mounted on a headset. The microphone is designed to minimize input of other background sounds. If a different operator is to use the system, the dictionary must be reestablished with that operator's voice patterns. Voice systems have some advantage over other input devices, since the attention of the operator does not have to be switched from one device to another to enter a command.

The biggest disadvantage is that voice systems are user-dependent while the need is for systems that are not user-dependent. For example, if one of the employee working on the voice system leaves the company then that system will have to be either bought again or programmed again which is a big bottleneck in the way.

1.18 CONTINUAL REFRESH GRAPHIC OUTPUT/DISPLAY DEVICES

Cathode Ray Tube (CRT)

The quality of displayed image is very important in most applications of computer graphics. CRT was the only available device which was capable of converting computer's electrical signals into variable images at high speeds.

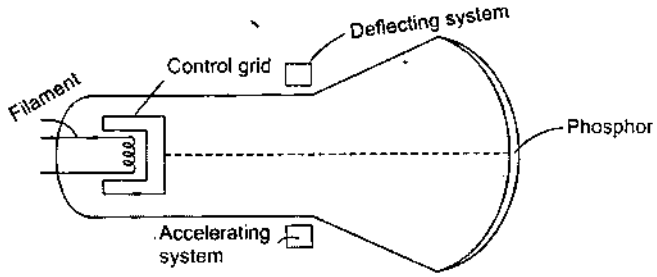


Fig. 1.14

The face of the CRT is more or less flat and is coated on the inside with phosphor which glows when electron beam strikes it.

A beam of electrons *i.e.*, cathode rays is emitted by electron gun and passes through focusing and deflection system that directs the beam on a specified point on the phosphor coated screen.

CRT is best suited for interactive Computer Graphics.

1.19 WHY PHOSPHORS ARE USED

The phosphors used in a graphic display are normally chosen for color characteristics and persistence. Generally, persistence measured as the time for the brightness to drop to 1/10 of it initial values, should last about 100 msec or less, allowing refresh at 30 hertz rates.

The phosphors should also possess a no. of other attributes: Small grain size for added resolution, high efficiency in terms of electric energy converted to light and resistance to burning under prolonged excitation. There are many different phosphors which have been produced to improve performance using various compounds of Zinc, Cadmium and calcium.

1.20 DIRECT VIEW STORAGE TUBE

Some devices were developed where refreshing is not important. DVST is CRT with some modifications, *i.e.*, storage capability. Whatever picture or image displayed is stored on the tube.

Bad points:
Slow and difficult
Quality deteriorates with time

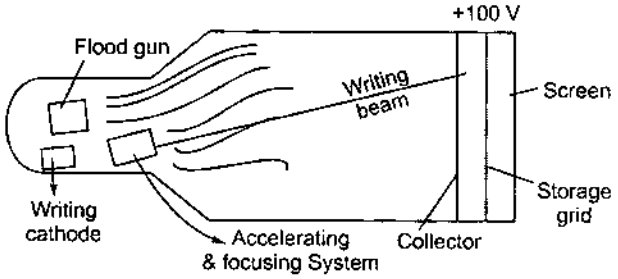


Fig. 1.15

DVST behaves like CRT with extremely long persistence. A line written on the screen will remain visible for up to an hour before it fades away from sight.

It has a similar focusing and accelerating system but difference is that beam doesn't white directly on the screen but on the storage grid as positive charged pattern. Storage grid is wire mesh grid coated with dielectric material (thin Al coating).

Function of collector is for smoothing purposes (unwanted electrons are removed).

1.21 RASTER AND RANDOM DISPLAY (MONITORS)

Raster Monitors/Displays

Depending on different types of display devices, there are different displays possible like Raster scan and Random scan. Earlier display devices use to support random scan as it took less memory because we were not interested in all the pixels of the image. But with cost of systems and memory going down, raster scan is used now-a-days. We will be referring to Raster scan in our book.

Computer Graphics images are composed of a finite number of pixels. A display with good resolution might have 1000 divisions in X and Y directions. The screen would then have 1000 X 1000 or 1 million pixels. Each pixel requires at least one bit of intensity information, light or dark and further bits are need if shades of gray or different colors are desired. Thus if we actually store the information for each pixel in computer memory, a lot of memory would be required. This is in fact, what is done in some RASTER GRAPHIC DISPLAYS. The portion of memory used to hold pixels is called frame buffer. The memory is usually scanned and displayed by DMA, *i.e.*, special hardware independent of central processor (leaving the processor free for generation of images).

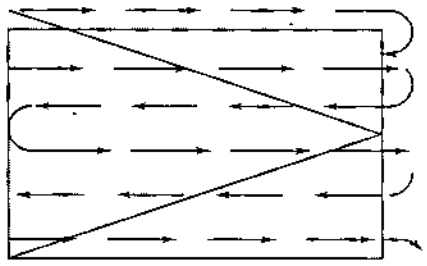


Fig. 1.16

The term RASTER is a synonym for "matrix" therefore a raster scan CRT scans a matrix with electron beam. The rate at which electron beam scans the surface of CRT is often directly related to the frequency of local line voltage.

NOTES

A raster CRT device can be considered as a matrix of discrete cells each of which can be made bright thus it is a point plotting technique.

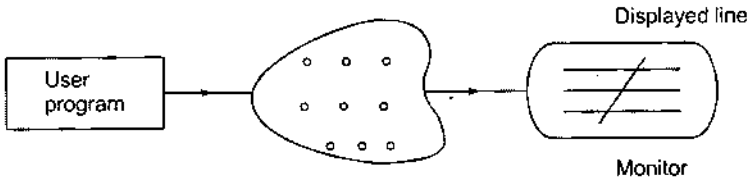


Fig. 1.17

NOTES

In the raster display, Frame buffer may be examined to determine what is currently being displayed. Surfaces as well as lines may be displayed on raster scan display devices. The raster display can also display color images.

The most common type of graphics monitor employing a CRT is the raster scan display, based on television technology. In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. Picture definition is stored in a memory area called the refresh buffer or frame buffer.

This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row (scan line) at a time. Each screen point is referred to as a pixel (shortened form of picture element). The capability of a raster scan system to store intensity information for each screen point makes it well suited for the realistic display of scenes containing subtle shading and color patterns. Home television sets and printers are examples of other systems using raster scan methods. Intensity range for pixel positions depends on the capability of the raster system. In a simple black and white system, each screen point is either on or off, so only one bit per pixel is needed to control the intensity of screen positions. For a bi-level system, a bit value of 1 indicates that the electron beam is to be turn on at that position, and a value of 0 indicates that the beam intensity is to be off.

Additional bits are needed when color and intensity variations can be displayed. Upto 24 bits per pixel are included in high-quality systems, which can require several megabytes of storage for the frame buffer, depending on the resolution of the system. A system with 24 bits per pixel and a screen resolution of 1024 × 1024 requires 3 megabytes of storage for the frame buffer. On a black-and-white system with one bit per pixel, the frame buffer is commonly called a bitmap. For Systems with multiple bits per pixel, the frame buffer is often referred to as a pixmap.

Refreshing on raster-scan displays is carried out at the rate of 60 to 80 frames per second, although some systems are designed for higher refresh rates. Sometimes, refresh rates are described in units of cycles per second, or Hertz (Hz), where a cycle corresponds to one frame. Using these units, we would describe a refresh rate of 60 frames per second as simply 60 Hz. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line. The return to the left of the screen, after refreshing each scan line, is called the horizontal retrace of the electron beam. And at the end of each frame (displayed in 1/80th to 1/60th of a second), the electron beam returns (vertical retrace) to the top left comer of the screen to begin the next frame.

On some raster-scan systems (and in TV sets), each frame is displayed in two passes using an interlaced refresh procedure. In the first pass, the beam sweeps across every other scan line from top to bottom. Then after the vertical retrace, the beam sweeps out the remaining scan lines. Interlacing of the scan lines in this way allows us to see the entire scene displayed in one-half the time it would have taken to sweep across all the lines at once from top to bottom.

Interlacing is primarily used with slower refreshing rates. On an older, 30 frame per-second, non interlaced display, for instance, some flicker is noticeable. But with interlacing, each of the two passes can be accomplished in 1/60th of a second, which brings the refresh rate nearer to 60 frames per second. This is an effective technique for avoiding flicker, providing that adjacent scan lines contain similar display information.

Random Monitors/Displays

NOTES

When operated as a random-scan display unit, a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn. Random scan monitors draw a picture one line at a time and for this reason are also referred to as vector displays (or stroke-writing or calligraphic displays). The component lines of a picture can be drawn and refreshed by a random scan system in any specified order.

A pen plotter operates in a similar way and is an example of a random scan, hard-copy device. Refresh rate on a random scan system depends on the number of lines to be displayed. Picture definition is now stored as a set of line drawing commands in an area of memory referred to as the refresh display file. Sometimes the refresh display file is called the display list, display program, or simply the refresh buffer.

To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all line drawing commands have been processed, the system cycles back to the first line command in the list. Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second. High quality vector systems are capable of handling approximately 100,000 "short" lines at this refresh rate. When a small set of lines is to be displayed, each refresh cycle is delayed to avoid refresh rates greater than 60 frames per second. Otherwise, faster refreshing could burn out the phosphor.

Random scan systems are designed for line drawing applications and cannot display realistic shaded scenes. Devices such as DVST, plasma panel etc. support only line drawing. They don't support for solid areas which can be constructed on raster displays. Such line drawing devices are calligraphic displays.

Since picture definition is stored as a set of line drawing instructions and not as a set of intensity values for all screen points, vector (random) displays generally have higher resolution than raster systems. Also, vector displays produce smooth line drawings because the CRT beam directly follows the line path. A raster system, in contrast, produces jagged lines that are plotted as the point sets.

Difference between Raster and Random Displays: The difference between raster and random display is that raster display gives a realistic image and is not confined to just line or calligraphic drawings. Any kind of graphics can be designed using raster displays. Vector (random) displays generally have higher resolution than raster systems as we can pass primitive commands directly. For example if we have to draw a line, raster would print the pixels of the line row by row on each scan line but in random display line is directly drawn from the current pen position to the final coordinates passed. Also, vector displays produce smooth line drawings because the CRT beam directly follows the line path. A raster system, in contrast, produces jagged lines that are plotted as the point sets.

NOTES

Random	Raster
Restricted to engineering, line drawing applications	Stores intensity information for each screen point so well suited for real life applications which include shading and coloring
Doesn't use interlacing Higher resolution	Uses interlacing Lower resolution
More expensive Uses monochrome or beam penetration type	Less expensive Uses monochrome or shadow mask type
Image is displayed by steering the beam along the vectors	Image is displayed by scanning the whole display area
Editing is easy Refresh rate depends directly on picture complexity	Editing is difficult Refresh rate independent of picture complexity

1.22 LCD AND LED

Liquid Crystal Displays are commonly used in small systems, such as calculators and portable, laptop computers. These non emissive devices produce a picture by passing polarized light from the surroundings or from an internal light through a liquid crystal material that can be aligned to either block or transmit the light. The term **liquid crystal** refers to the fact that these compounds have a crystalline arrangement of molecules, yet they flow like a liquid. Flat-panel displays commonly use nematic (threadlike) liquid-crystal compounds that tend to keep the long axes of the rod-shaped molecules aligned. A flat-panel display can then be constructed with a nematic liquid crystal. Two glass plates, each containing a light polarizer at right angles to the other plate, sandwich the liquid crystal material. Rows of horizontal transparent conductors are built into one glass plate, and columns of vertical conductors are put into the other plate. The intersection of two conductors defines a pixel position. Polarized light passing through the material is twisted so that it will pass through the opposite polarizer. The light is then reflected back to the viewer. To turn off the pixel, we apply a voltage to the two intersecting conductors to align the molecules so that the light is not twisted. This type of flat-panel device is referred to as a passive-matrix LCD.

Picture definitions are stored in a refresh buffer, and screen is refreshed at the rate of 60 frames per second, as in the emissive devices. A hand calculator with Back lighting is also commonly applied using solid-state electronic devices. Colors can be displayed by using different materials or dyes and by placing a triad of color pixels at each screen location.

Another method is to place a transistor at each pixel location, using thin-film transistor technology. The transistors are used to control the voltage at pixel locations and to prevent charge from gradually leaking out of the liquid-crystal cells. These devices are called active-matrix displays.

LED

A type of emissive device is the Light Emitting Diode (LED). A matrix of diodes is arranged to form the pixel positions in the display, and picture definition is stored in a refresh buffer. As in scan-line refreshing of a CRT, information is read from the refresh

buffer and converted to voltage levels that are applied to diodes to produce the light patterns in the display

1.23 VGA AND SVGA MONITORS

VGA Monitors

NOTES

Abbreviation of *video graphics array*, a graphics display system for PCs developed by IBM. VGA has become one of the de facto standards for PCs. In text mode, VGA systems provide a resolution of 720 by 400 pixels. In graphics mode, the resolution is either 640 by 480 (with 16 colors) or 320 by 200 (with 256 colors). The total palette of colors is 262,144.

Unlike earlier graphics standards for PCs — MDA, CGA, and EGA — VGA uses analog signals rather than digital signals. Consequently, a monitor designed for one of the older standards will not be able to use VGA.

Since its introduction in 1987, several other standards have been developed that offer greater resolution and more colors (SVGA, 8514/A graphics standard, and XGA), but VGA remains the lowest common denominator. All PCs made today support VGA, and possibly some other more advanced standard.

It has been technologically outdated in the PC market for some time. VGA was the most recent graphical standard that the majority of manufacturers conformed to, making it the lowest common denominator that all PC graphics hardware supports before a device-specific driver is loaded into the computer. For example, the Microsoft Windows splash screen appears while the machine is still operating in VGA mode, which is the reason that this screen always appears in reduced resolution and color depth.

VGA was officially superseded by IBM's XGA standard, but in reality it was superseded by numerous extensions to VGA made by clone manufacturers that came to be known as "Super VGA".

SVGA Monitors

Super Video Graphics Array, almost always abbreviated to Super VGA or just SVGA is a broad term that covers a wide range of computer display standards.

Originally, it was an extension to the VGA standard first released by IBM in 1987. Unlike VGA—a purely IBM-defined standard—Super VGA was defined by the Video Electronics Standards Association (VESA), an open consortium set up to promote interoperability and define standards. When used as a resolution specification, in contrast to VGA or XGA for example, the term *SVGA* normally refers to a resolution of 800 × 600 pixels.

Super VGA was first defined in 1989. In that first version, it called for a resolution of 800 × 600 4-bit pixels. Each pixel could therefore be any of 16 different colors. It was quickly extended to 1024 × 768 8-bit pixels, and well beyond that in the following years.

Although the number of colors was defined in the original specification, this soon became irrelevant as (in contrast to the old CGA and EGA standards) the interface between the video card and the VGA or Super VGA monitor uses simple analog voltages to indicate the desired color depth. In consequence, so far as the monitor is concerned, there is no theoretical limit to the number of different colors that can be displayed. Note that this applies to *any* VGA or Super VGA monitor.

While the output of a VGA or Super VGA video card is analog, the internal calculations the card performs in order to arrive at these output voltages are entirely digital. To increase the number of colors a Super VGA display system can reproduce, no change at

NOTES

all is needed for the monitor, but the video card needs to handle much larger numbers and may well need to be redesigned from scratch. Even so, the leading graphics chip vendors were producing parts for high-color video cards within just a few months of Super VGA's introduction.

On paper, the original Super VGA was to be succeeded by Super XGA, but in practice the industry soon abandoned the attempt to provide a unique name for each higher display standard, and almost all display systems made between the late 1990s and the early 2000s are classed as Super VGA.

Monitor manufacturers sometimes advertise their products as *XGA* or *Super XGA*. In practice this means little, since *all* Super VGA monitors manufactured since the later 1990s have been capable of at least XGA and usually considerably higher performance.

1.24 DISPLAY PROCESSOR

Display processor referred to as a graphics controller or a display coprocessor also. It makes the CPU free from the graphics chores. There is a separate memory for the display processor.

Display processor basically used for digitizing a picture definition given in a program into a set of pixel-intensity values for storage in the frame buffer. This process is called scan conversion. Display processors also used for generating various line styles, displaying color areas and performing various operations such as magnify or reduce the image rotate it or shift it or reflect it. It can interface with input devices. The process is illustrated in Fig. 1.18.

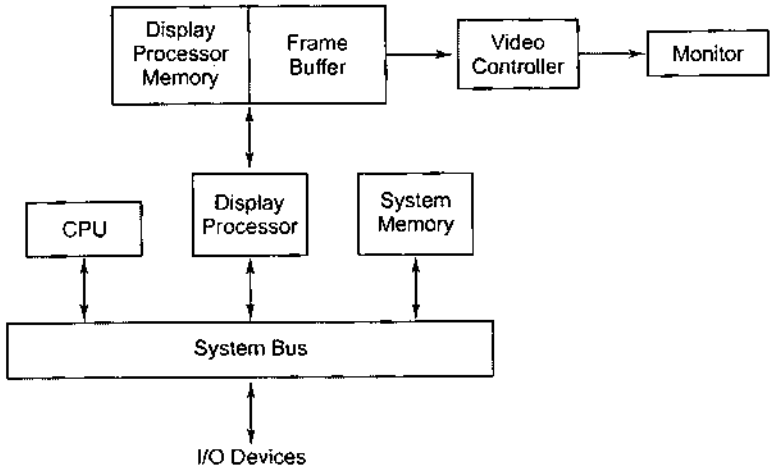


Fig. 1.18 Raster-graphics with a display processor

Run-length encoding is used for saving in storage space. The disadvantages of this approach are that changes in the intensity are difficult to make and run length decreases but storage requirements actually increases. When short runs are involved in this process the display processor gets difficulty for processing.

1.25 CHARACTER GENERATION

Characters are almost always built into the graphics display device, usually as hardware but sometimes through software. There are three primary methods for character generation.

Stroke Method

This method creates characters out of a series of line segments, like strokes of a pen as shown in Fig. 1.19. We can generate our own stroke method character generator by VECGEN algorithm. We can change the scale by doubling length of each segment.

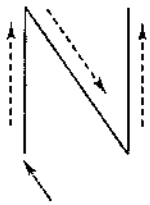


Fig. 1.19 Stroke method

Dot Matrix or Bit Map Method

In this characters are represented by an array of dots as shown in Fig. 1.20. An array of 5 dots wide and 7 dots high is often used. But 7×9 and 9×13 arrays are also found. This array is like a small frame buffer, just big enough to hold a character. The dots are the pixels for this small array. Placing the character on the screen then becomes a matter of copying pixel values from the small character array into some portion of the screen's frame buffer. The size of a dot is fixed.

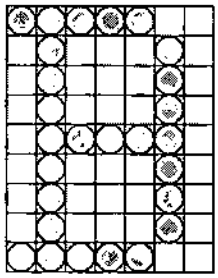


Fig. 1.20 Character by dot matrix method [B]

Starbust Method

In this method we use pattern of 24 strokes to generate a character. Due to its characteristics appearances it is called starbust. We can generate a character by highlighting some of the strokes and keeping others blank. We can show the composition of 24 strokes as shown in Fig. 1.21.

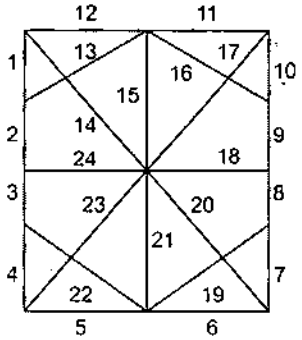


Fig. 1.21 Starbust method

Now we can generate any character using above strokes. Suppose we want to generate "F", we simply highlight the strokes numbered—1, 2, 3, 4, 11, 12, 24.

Using this method we can generate different characters, numbers and different special characters. For each character, we require 24 bit code word. Depending upon the character, we can highlight a particular stroke.

Disadvantages

- This method is very slow.
- Requires code conversion feature.
- Poor character quality.

NOTES

Antialiasing techniques can be applied to characters. This can improve the appearance of the character, particularly for very small fonts and characters.

SUMMARY

- Computer Graphics has been defined as the collection of data structure, graphic algorithms and any higher level language. Passive Computer Graphics is called Off-line graphics while Interactive Computer Graphics is called On-line graphics. Application areas include Flight simulators, CAD, video games and many more. The survey of computer graphics takes us into the start of computer graphics (black era) till the present day graphics (Colored era).
- Components of Computer Graphics include Frame buffer, monitor and display controller. Input devices include keyboards, mouse, light pen, joystick, digitizer, touch panel, image scanners, voice systems etc while output devices include monitors (CRT) and printers/plotters.
- VGA is a graphics display system for PCs developed by IBM. VGA has become one of the de facto standards for PCs. In text mode, VGA systems provide a resolution of 720 by 400 pixels. In graphics mode, the resolution is either 640 by 480 (with 16 colors) or 320 by 200 (with 256 colors). The total palette of colors is 262,144.
- Super Video Graphics Array, almost always abbreviated to Super VGA or just SVGA is a broad term that covers a wide range of computer display standards. Originally, it was an extension to the VGA standard first released by IBM in 1987. Super VGA was first defined in 1989. In that first version, it called for a resolution of 800 × 600 4-bit pixels. Each pixel could therefore be any of 16 different colors. It was quickly extended to 1024 × 768 8-bit pixels, and well beyond that in the following years.
- Analog to digital conversion and vice versa is done by A/D converter. Raster scan scans the screen line by line while random scan plots the image directly through the use of primitive commands. Raster scan is being used now-a-days in graphics applications. Even the home TV uses raster scan technology for rendering real images.
- Graphics primitives are primitive commands used by earlier graphics systems. Random scan uses graphics primitives to plot the line or surfaces. GKS and CORE are some standards associated with graphics primitives.

REVIEW QUESTIONS

1. What do you understand by Computer Graphics? Give application areas of Interactive Computer Graphics.
2. What is display memory called in a system which has portion of memory reserved for graphics use only?
3. Good graphic programmers avoid the use of floating point operation whenever possible. Why?
4. Explain the following terms:
   (*i*) Persistence (*ii*) Phosphorence
   (*iii*) Raster scan (*iv*) Pixels and frame buffer.
5. What do you understand by V-RAM and how much V-RAM do you recommend for your system?

6. What do you understand by Input/ Output devices? Explain the working of CRT.
7. What is the difference between Raster and Random scan? Highlight the advantages of using Raster scan.
8. **Fill up the blanks:**
   (*a*) CAD stands for .....
   (*b*) CAM stands for .....
   (*c*) DMA stands for .....
   (*d*) Pixel stands for .....
   (*e*) CRT is coated with .....a..... to avoid burning under prolonged excitation.

NOTES

ANSWERS

8. (a) Computer Aided Designing (b) Computer Aided Manufacturing
   (c) Direct Memory Access (d) Picture Element
   (e) Phosphor.

FURTHER READINGS

- **Computer Graphic:** V.K. Pachghare, Laxmi Publications, 2007. Second edition.
- **Computer Graphics:** Prabhakar Gupta, Vineet Agarwal and Manish Varshney, Laxmi Publications, 2011.
- **Computer Graphics:** Rajiv Chopra, S. Chand Publisher, 2011.
- **Computer Graphics:** C.S. Verma, Ane Books, 2011.
- **Computer Graphics:** Pradeep K. Bhatia, I.K. International, 2009, pbk, Second Edition.
- **Computer Graphics:** Ruchi Mishra, Global Vision Publisher, 2010.

UNIT II

NOTES

OUTPUT PRIMITIVES (LINE AND CIRCLE DRAWING ALGORITHMS)

STRUCTURE

2.0 LEARNING OBJECTIVES

After going through this unit, you should be able to:

- describe color display techniques
- explain the term frame buffer
- discuss about scan conversion
- describe the line drawing algorithms
- explain ellipse and circle generation.

2.1 COLOR DISPLAY TECHNIQUES

NOTES

The world would look so dull if there were no colors or if the quality of colors is not good. Earlier graphics applications were developed in black and white but with the development in technology and colored monitors becoming popular, graphics applications are now being developed using colored displays.

A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light. By combining the emitted light from the different phosphors, a range of colors can be generated. The two basic techniques for producing color displays with a CRT are the **beam penetration method and the shadow mask method.**

There are two ways of getting colored displays:

1. Beam penetration color display
2. Shadow mask color display

The normal CRT can create image of a single color (black and white) due to limitation of its phosphor.

A colored CRT for line drawing displays has been developed and it uses a multi layer (RBG) phosphor and achieves color display.

1. **BEAM PENETRATION COLORED DISPLAY:** The beam penetration method for displaying color pictures has been used with random-scan monitors. Beam penetration colored display makes use of CRT but with multi layer phosphor. The normal CRT can create images of single color due to limitation of phosphor. A color CRT device for live drawing display has been developed and it uses a multi layer phosphor and achieves color control by modulating a normally constant parameter, namely the **"beam accelerating potential"**.

   The beam penetration CRT is similar to normal CRT's, the only unusual component is the multilayer phosphor in which a layer of red phosphor is deposited behind the initial layer of green phosphor. If the accelerating potential is increased, the velocity of the beam striking the phosphor is greater and as a result the beam penetrates into green phosphor, increasing the green component or the light output. When a fairly low potential electron beam strikes the tube face, it excites only the red phosphor and the therefore produces a red trace.

   Two layers of phosphor, usually red and green, are coated onto the inside of the CRT screen, and the displayed color depends on how far the electron beam penetrates into the phosphor layers. A beam of slow electrons excites only the outer red layer. A beam of very fast electrons penetrates through the red layer and excites the inner green layer. At intermediate beam speeds, combinations of red and green light are emitted to show two additional colors, orange and yellow. The speed of the electrons, and hence the screen color at any point, is controlled by the beam-acceleration voltage. Beam penetration has been an inexpensive way to produce color in random scan monitors, but only four colors are possible, and the quality of pictures is not as good as with other methods. In this way a limited range of colors, *i.e.*, red, orange, yellow and green can be generated.

   Advantages: The biggest advantage is that it is at half cost of shadow mask and its resolution is better.

   Disadvantage: Biggest disadvantage is that change of color takes time which doesn't suit interactive graphics at all.

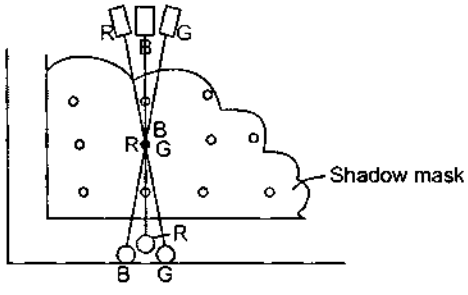


Fig. 2.1

2. SHADOW MASK COLOR DISPLAY : In this display, there are three electron guns, one for each of three primary colors, *i.e.*, red, green and blue. The electron guns are frequently arranged in a triangular pattern called delta corresponding to a similar triangular pattern of red, green and blue phosphor dots on the face of the CRT.

To ensure that individual electron guns excite the correct phosphor dots (e.g., red gun excites only red phosphor dot), a perforated metal grid is placed between the electron guns and the face of the CRT. This is the shadow mask (the perforations in the shadow mask are arranged in the same triangular pattern as phosphor dots).

NOTES

Note: Distance between perforations is called pitch.

The color guns are arranged so that the individual beams converge and intersect at the shadow mask. Upon passing through the hole in the shadow mask the red beam, for, e.g., is prevented or masked from intersecting either the green or blue phosphor dot. It can only intersect the red phosphor dot.

By varying the strength of electron beam for each individual primary color, different shades can be combined into a large number of colors for each pixel. Uses much wider range of colors.

It is used in colored TVs and gives us a large number of colors as compared to beam penetration method. Shadow mask methods are commonly used in raster scan systems (including color TV) because they produce a much wider range of colors than the beam penetration method. A shadow mask CRT has three phosphor color dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. This type of CRT has three electron guns, one for each color dot, and a shadow mask grid just behind the phosphor coated screen. Figure illustrates the delta shadow mask method, commonly used in color CRT systems. The three electron beams are deflected and focused as a group onto the shadow mask, which contains a series of holes aligned with the phosphor dot patterns. When the three beams pass through a hole in the shadow mask, they activate a dot triangle, which appears as a small color spot on the screen. The phosphor dots in the triangles are arranged so that each electron beam can activate only its corresponding color dot when it passes through the Shadow mask.

Another configuration for the three electron guns is an in-line arrangement in which the three electron guns, and the corresponding red, green and blue color dots on the screen, are aligned along one scan line instead of in a triangular pattern. This in-line arrangement of electron guns is easier to keep in alignment and is commonly used in high-resolution color CRTs.

We obtain color variations in a shadow mask CRT by varying the intensity levels of the three electron beams. By turning off the red and green guns, we get only the color coming from the blue phosphor. Other combinations of beam intensities produce a small light spot for each pixel position, since our eyes tend to merge the three colors into one composite. The color we see depends on the amount of excitation of the red, green, and blue phosphors.

A white (or gray) area is the result of activating all three dots with equal intensity. Yellow is produced with the green and red dots only, magenta is produced with the blue and red dots, and cyan shows up when blue and green are activated equally. In some low-cost systems, the electron beam can only be set to on or off, limiting displays to eight colors. More sophisticated systems can set intermediate intensity levels for the electron beams, allowing several million different colors to be generated.

Color CRTs in graphics systems are designed as RGB monitors. These monitors use shadow mask methods and take the intensity level for each electron gun (red, green and blue) directly from the computer system without any intermediate processing. High-quality raster-graphics systems have 24 bits per pixel in the frame buffer, allowing 256 voltage settings for each electron gun and nearly 17 million color choices for each pixel. An **RGB color system** with **24 bits** of storage per pixel is generally referred to as a full-color system or a **true-color** system.

NOTES

Plasma Panel

Plasma panels, also called gas-discharge displays, are constructed by filling the region between two glass plates with a mixture of gases that usually uses neon. A series of vertical conducting ribbons is placed on one glass panel, and a set of horizontal ribbons is built into the other glass panel. Firing voltages applied to a pair of horizontal and vertical conductors cause the gas at the intersection of the two conductors to break down into glowing plasma of electrons and ions. Picture definition is stored in a refresh buffer, and the firing voltages are applied to refresh the pixel positions (at the intersections of the conductors) 60 times per second.

It is an alternative to CRT as there was a search for a device which

- Can retain display indefinitely
- No compromise on quality of display
- Small space requirement
- Low voltage power supply.

The term plasma is the name of the device which refers to the neon gas that is sandwiched and sealed between two glass panels. It is a panel of very tiny neon gas bulbs which can selectively be made to glow. By apply firing voltage, glow starts. Once the glow starts, the voltage can be decreased up to which glow continues (sustaining voltage). Below this voltage, glow dies (cut of voltage). Plasma Panel need not be refreshed. Cost is high and resolution low. It is an array of neon gas where on means bright glow and off means no glow.

Alternating methods are used to provide faster application of the firing voltages, and thus brighter displays. Separation between pixels is provided by the electric field of the conductors. One disadvantage of plasma panels has been that they were strictly monochromatic devices, but systems have been developed that are now capable of displaying color and grayscale.

Thin-film electroluminescent displays are similar in construction to a plasma panel. The difference is that the region between the glass plates is filled with a phosphor, such as zinc sulfide doped with manganese, instead of a gas. When a sufficiently high voltage is applied to a pair of crossing electrodes, the phosphor becomes a conductor in the area of the intersection of the two electrodes. Electrical energy is then absorbed by the manganese atoms, which then release the energy as a spot of light similar to the glowing plasma effect in a plasma panel. Electroluminescent displays require more power than plasma panels, and good color and gray scale displays are hard to achieve.

2.2 FRAME BUFFER

Frame buffer is the portion of memory reserved for holding the complete bit-mapped image that is sent to the monitor. Typically the frame buffer is stored in the memory chips on the video adapter. In some instances, however, the video chipset is integrated into the motherboard design, and the frame buffer is stored in general main memory. So we can say in our language that Frame Buffer is the memory area which holds the complete information of the pixel.

NOTES

The **frame buffer** is a video output device that drives a video display from a memory buffer containing a complete frame of data. The information in the buffer typically consists of color values for every pixel (point that can be displayed) on the screen. Color values are commonly stored in 1-bit monochrome, 4-bit palletized, 8-bit palletized, 16-bit high color and 24-bit true color formats. An additional alpha channel is sometimes used to retain information about pixel transparency. The total amount of the memory required to drive the frame buffer is dependent on the resolution of the output signal, as well as the color depth and palette size.

Frame buffers differ significantly from the vector graphics displays that were common prior to the advent of the frame buffer. With a vector display, only the vertices of the graphics primitives are stored. The electron beam of the output display is then commanded to move from vertex to vertex, tracing an analog line across the area between these points. With a frame buffer, the electron beam (if the display technology uses one) is commanded to trace a left-to-right, top-to-bottom path across the entire screen, much in the same way a television renders a broadcast signal. At the same time, the color information for each point on the screen is pulled from the frame buffer, creating a set of discrete picture elements (pixels).

2.3 ELEMENTS OF 2D GEOMETRY FOR GRAPHICS

2D geometry is used extensively in graphics and so we must be familiar with geometry. We must know the equations related to lines, circles, ellipses etc. in order to generate or scan convert them on the system. These equations can be used directly in our programs so derivations are skipped in this book considering them out of scope.

We can use built-in functions of C language to draw lines, circles, ellipses, arcs and polygons but we are interested in drawing them through our own algorithms as they are more flexible.

First let us define what lines are and it can easily be said that any line can be drawn using two endpoints (x_1, y_1) and (x_2, y_2)

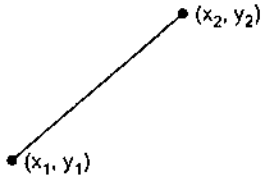


Fig. 2.2

The line equation is y = mx + b

Slope $m = (y_2 - y_1)/(x_2 - x_1)$

where (x_1, y_1) and (x_2, y_2) are end point co-ordinates provided by the user

and $b = y_1 - mx_1$

Slope m is the change in height divided by change in width for two points on the line.

The intercept b is the height at which line crosses the Y axis.

2.4 LINE SEGMENT

Let us consider only those points on a line that lie between two end points A_1 and A_2. This is called a line segment.

Length of a line segment is very important to evaluate because our algorithm will focus on that. So the question is how long is a line segment?

Given two endpoints A_1 and A_2, we can determine its length L using Pythagoras theorem.

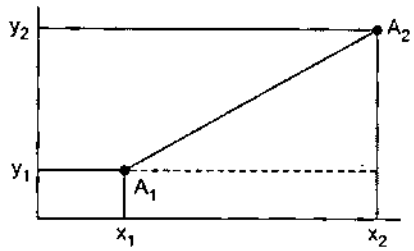


Fig. 2.3 Pythagoras theorem

$$L_2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

or

$$L = ((x_2 - x_1)^2 + (y_2 - y_1)^2)^{1/2}$$

2.5 VECTORS

A vector has a single direction and a length. A vector may be denoted $|D_x, D_y|$ where D_x indicates how far to move along *x*-axis direction and D_y indicates how far to move along the *y*-axis.

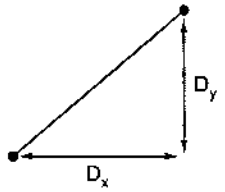


Fig. 2.4

Unlike line segments, vectors have no fixed position in space. They tell us how far and what direction to move but they do not tell us where to start. The idea of a vector is useful because it closely parallels the manner in which a pen draws lines on paper or an electron beam draws lines.

2.6 SCAN CONVERSION

Scan Conversion is the process of representing continuous objects as a collection of discrete pixels. Displaying discrete pixels of the object is called SCAN CONVERSION (Students may be asked to scan convert a point, line, circle or an ellipse in examinations).

The video output circuitry is capable of converting binary values stored in its display memory into pixel on, pixel off information. This information is used by raster output device to display a point. This helps the graphics programmers to display or create models composed of discrete dots.

Any thing in the world can be produced with a dense matrix of dots but we think in terms of graphic objects such as points, lines, circles and ellipses.

Computer graphics has come a long way since the start of graphics. Many algorithms have been developed to scan convert the above mentioned shapes. However regardless

of the method used, computer can produce images on raster devices only by turning appropriate pixels on or off.

Scan conversion algorithm can be implemented in computer hardware or firmware (programs written in ROM). However scan conversion algorithms can be implemented in software as per our requirements.

NOTES

2.7 HOW TO HANDLE SCREEN COORDINATES AND HOW "C" PROGRAM WORKS

The point plotting/scan conversion techniques are based on the use of a Cartesian co-ordinate system. Points are represented by x and y co-ordinates, the value of x increases from L – R and y from top to bottom (Check out the coordinates of your screen by writing simple programs using C language).

Let us start by writing simple programs in C programming language for graphics and see how the screen behaves. When we start drawing any graphics on the screen, we need a header file **graphics.h** and a library file called **graphics.lib**. The header file contains the information and explanations of all functions and constants we need. Both these files are provided as part of TURBO C.

To move from the text mode to the graphics mode that offers the best resolution, the system puts the number corresponding to that mode in variable **gm**. The **gm** number tells us which monitor we are using and its resolution.

The programs given in the book are developed using VGA adapter with the resolution 640 × 480.

To understand **gd**, we've to understand the concept of device drivers. Turbo C offers some graphics drivers. These are the file with **BGI(*.bgi)** extension. Throughout our programs, gd has been assigned the value DETECT thereby asking initgraph() function to figure out which BGI file is needed.

The basic tools or built in functions used for drawing shapes are function like putpixel (), line(), circle(), ellipse(), arc() and drawpoly().

putpixel(x_1, y_1) illuminates or lights the pixel at position (x_1, y_1) on the screen.

line (x_1, y_1, x_2, y_2) draws a line from point (x_1, y_1) to point (x_2, y_2).

2.8 LINE DRAWING ALGORITHMS

Straight line segments are used in almost every graphics application such as generating sceneries, bar charts, block diagram, graph, engineering drawing, architectural plan etc. Since the straight line drawing is very important and so useful it is worth taking care that they are well drawn.

Criteria for Good Computer Generated Lines

1. Lines should appear straight and not zigzag. Point Plotting techniques are admirably suited to the generation of vertical, horizontal or lines at 45 degree to x and y axis
2. Line should terminate accurately
3. Lines should have constant density, *i.e.*, point should be evenly distributed on the line.

Scan converting algorithms compute the points for us and corresponding to these points, signals are generated. Algorithms should be such that they require minimum computations.

There are two lines drawing algorithm which are employed by graphics designers/ developers:

1. DDA (Digital Differential Analyzer) method
2. Bresenham's line drawing algorithm.

NOTES

2.9 DDA LINE DRAWING ALGORITHM

One technique for obtaining a rasterized straight line is to solve the differential equation for a straight line

$$D_Y/D_x = \text{constant} \qquad \text{or} \qquad \Delta_y/\Delta_x = (y_2 - y_1)/(x_2 - x_1)$$

The solution is for the above differential equation is

$$Y_{i+1} = Y_i + D_Y$$

$$Y_{i+1} = Y_i + (y_2 - y_1)/(x_2 - x_1)\,\Delta_x \qquad (1)$$

where (x_1, y_1) and (x_2, y_2) are the end points of the required straight line and Y_i is the initial value for any given step along the line.

In fact, equation 1 represents a recursive relation for successive values of y along the required line and is usually used to rasterize a line. This method is called Digital Differential Analyzer (DDA) method.

Now we can use the above recursive relation in our computer program. Following is a simple DDA algorithm which will work in all quadrants and would rasterize a line. The line end points are (x_1, y_1) and (x_2, y_2) and are not equal. The line end points are provided by the user through input commands of the program.

DDA Algorithm

Integer is integer floor function

Sign returns – 1, 0, 1 as its arguments is < 0, = 0, > 0.

/*Appropriate the line length so that we can move along either of X or Y axis*/

If $abs(x_2 - x_1) >= abs(y_2 - y_1)$ then

$$\text{len} = abs(x_2 - x_1)$$

Else

$$\text{len} = abs(y_2 - y_1)$$

End if

/* Evaluating Δx and Δy to be used in our program*/

$$\Delta x = (x_2 - x_1)/\text{len}$$

$$\Delta y = (y_2 - y_1)/\text{len}$$

/* To round the values we use multiplication by 0.5 and using sign function makes algorithm work in all quadrants*/

$$x = x_1 + 0.5 * \text{sign}(\Delta x)$$

$$y = y_1 + 0.5 * \text{sign}(\Delta y)$$

/*Begin the loop*/

$$I = 1$$

As long as ($I \leq$ len)

Putpixel (integer (x), integer (y))

$$x = x + \Delta x$$

$$y = y + \Delta y$$

NOTES

I = I + 1

End loop

Lets take an example to illustrate the algorithm. We will solve a numerical and plot the line and then scan convert it on the system through a C program.

Example: Consider the line from (0, 0) to (6, 6), use DDA to rasterize the line.

Solution: Let us perform some initial calculations

Initially $X_1 = 0 \quad Y_1 = 0 \quad X_2 = 6 \quad Y_2 = 6$

So our first step would be to evaluate the length

$$\text{Length} = abs(x_2 - x_1) = 6 - 0 = 6$$
$$\text{Length} = abs(y_2 - y_1) = 6 - 0 = 6$$

So length = 6 (in any X or Y direction)

Had $(x_2 - x_1)$ been larger then X direction would have been chosen otherwise Y direction.

So our next step would be to evaluate Δx and Δy as in our algorithm

$$\Delta x = x_2 - x_1/\text{length} = 6 - 0/6 = 1$$
$$\Delta y = y_2 - y_1/\text{length} = 6 - 0/6 = 1$$
$$x = x_1 + 0.5 * \text{sign}(\Delta x)$$
$$= 0 + 0.5 * \text{sign}(1) \quad \textbf{/* since sign(1) returns 1*/}$$
$$x = 0.5$$

Similarly, $y = 0.5$

Evaluating through main loop gives us

i	*plot*	*x*	*y*
1		0.5	0.5
2	(0, 0)	1.5	1.5
3	(1, 1)	2.5	2.5
4	(2, 2)	3.5	3.5
5	(3, 3)	4.5	4.5
6	(4, 4)	5.5	5.5
7	(5, 5)	6.5	6.5
8	(6, 6)	7.5	7.5

So we can see from the above table how DDA method evaluates and plot new points (0.5 is taken as 0 in above table because we are considering floor value).

Example: Draw a line line from (0, 0) to (– 8, – 4) in third quadrant and develop a C program to draw a line using DDA algorithm.

Solution: Layout of DDA algorithm using "C"

```
Sign (int x)
{
                    if (k > 0) return 1;
                    if (k < 0) return –1;
                    if (k = = 0) return 0;
}
                    dx = x2 – x1;
                    dy = y2 – y1;
   if (abs(dx) > = abs(dy))
   len = abs(dx);
   else
```

```
        len = abs(dy);
        xinc = dx/len;
        yinc = dy/len;
        x = x1 + 0.5 * sign (dx);          /* to round off
        y = y1 + 0.5 * sign (dy);
    while (i < = len)
{
        putpixel(floor x, floor y);
        x+ = xinc;
        y+ = yinc;
        i++;
}
```

NOTES

For the sake of students we have given a program for DDA method in C language. Students can modify it according to their need and requirement. You can make it more interactive and user friendly.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void dda(int,int,int,int);
int sig(int);
int f1=0,p[8],x,y;
main()
{
int gd=DETECT,gm,i,j=0;
int xa[1]={400};
int ya[1]={100};
int xb[1]={800};
int yb[1]={400};
clrscr();
printf("line drawing using dda method");
initgraph(&gd,&gm," ");
while(j<1)
{
dda(xa[j],ya[j],xb[j],yb[j]);
j++;
}
getch();
closegraph();
return(0);
}
void dda(int xa,int ya,int xb,int yb)
{
int dx,dy,l,x1,y1,f,c;
dx=xb-xa;
dy=yb-ya;
if(abs(dx)>=abs(dy))
l=abs(dx);
```

NOTES

```
else
l=abs(dy);
x1=(dx)/l;
y1=(dy)/l;
x=x1+(0.5*sig(dx));
y=y1+(0.5*sig(dy));
while(f1<=l)
{
f=floor(x);
c=floor(y);
putpixel(f,c,WHITE);
x+=x1;
y+=y1;
f1++;
}
}
int sig(int k)
{
if (k>0)
return(1);
if (k<0)
return(-1);
if (k==0)
return(0);
}
```

2.10 BRESENHAM'S LINE ALGORITHM

Bresenham's line algorithm is an efficient method for scan converting straight lines in that it uses only integer addition, subtraction and multiplication by 2 and floating point operations are avoided in Bresenham's algorithm. The computer can perform the operations of integer addition and subtraction very rapidly. The lines drawn are of superior quality as compared to DDA method.

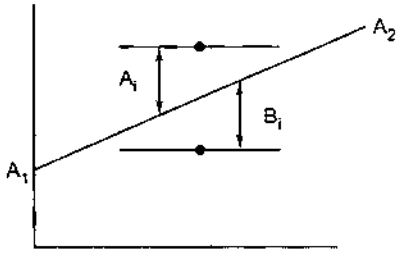


Fig. 2.5

In this algorithm we talk about a true line as shown in the above figure. Call the distance to those pixels lying immediately above the true line A_i and the distance to those directly below it B_i.

We will keep on talking about A_i and B_i throughout the algorithm. Bresenham's algorithm identifies the decision or test variable

$$d_i = A_i - B_i$$

when $d_i < 0$, the closet pixel in the raster will be the pixel below the true line.

Conversely, when $d_i >= 0$ the pixel immediately above the true line is closest.

To implement the algorithm all that remains is to calculate and update the various values of d_i.

Initially set

$$d_1 = 2d_y - d_x$$

where

$$d_x = X_2 - X_1$$

$$d_y = Y_2 - Y_1$$

Thereafter if $d_i >= 0$ the x and y are incremented

$$X_{i+1} = X_{i+1}$$

$$Y_{i+1} = Y_{i+1}$$

and

$$d_{i+1} = d_i + 2(d_y - d_x)$$

If $d_i < 0$ then only x is incremented

$$X_{i+1} = X_{i+1}$$

$$d_{i+1} = d_i + 2d_y$$

Example: Indicate which raster locations would be chosen by Bresenham's algorithm when scan converting a line from screen co-ordinates (1, 1) to (8, 5).

Solution: First the starting values (Initial values) must be found

$$d_x = x_2 - x_1$$
$$= 8 - 1$$
$$= 7$$
$$d_y = d_y - y_1$$
$$= 5 - 1$$
$$= 4$$

Therefore

$$d = 2d_y - d_x$$
$$= 2 * 4 - 7$$
$$= 1 \quad \text{Initial value}$$

Applying the above procedure again and again would get us the required solution.

Ultimately we would get a table of x and y that can be plotted to give a line segment.

x	y	d
1	1	1
2	2	−5
3	2	3
4	3	−3
5	3	5
6	4	−1
7	4	7
And so on	And so on	1

Following is the C implementation of Bresenham's algorithm.

Bresenham's Line Program

```
//TO DRAW A LINE (BRESENHAM'S LINE DRAWING ALGORITHM)...
#include<conio.h>
#include<stdio.h>
```

NOTES

```
#include<graphics.h>
#include<math.h>
#include<dos.h>
void line1 (int x1, int y1, int x2, int y2)
{
int dx, dy, 1, x, y, p, xend;
dx = (x2-x1) ;
dy = (y2-y1) ;
p = 2*dy-dx ;
if (x2 > x1)    //checks which x(x1 or x2) to use as starting point
{ //and which x(x1 or x2) to use as end point
x=x1 ;
y=y1 ;
xend=x2;
}
else
{
x = x2 ;
y = y2 ;
xend = x1 ;
}
while (x<xend)
{
putpixel (x, y, RED);
x++ ;
if (p < 0)
p = p + 2*dy
else
{
y++ ;
p = p + 2* (dy-dx) ;
}
}
}
void main (void)
{
int gd=DETECT,gm;
clrscr() ;
initgraph (& gd, & gm," ");
setbkcolor (GREEN) ;
line1 (200, 0, 300, 300) ;
getch ();
}
```

2.11 BRESENHAM'S CIRCLE GENERATION

NOTES

If a circle is to be scan-converted nicely, the use of trigonometric and power functions must be avoided. It is therefore desirable to perform the calculations necessary to find the scan converted points with only integer addition, subtraction, multiplication by powers of 2. Bresenham's circle algorithm allows these goals to be met with limited number of steps.

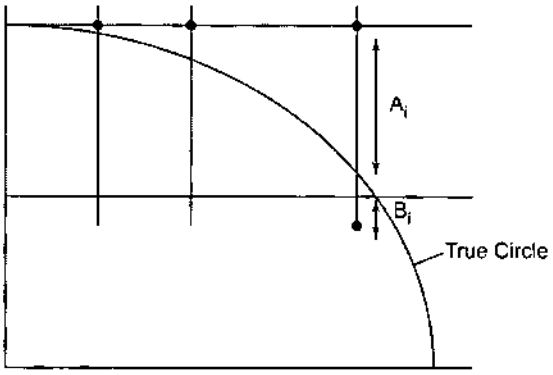


Fig. 2.6

As in the case of straight line, the best approximation of a circle will be defined by those pixels in the raster that fall the least distance from the true circle.

Decision variable d_i value is derived as (derivation is beyond the scope of this book).

$$d_i = 3 - 2r$$

Thereafter if $d_i < 0$ then only x is incremented

$$x_{i+1} = x_i + 1$$

and decision variable changes to

$$d_{i+1} = d_i + 4x_i + 6$$

and if $d_i >= 0$ then x & y are incremented

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i - 1$$

$$d_{i+1} = d_i + 4(x_i - y_i) + 10$$

We are giving Bresenham's Program for circle generation in C:

Bresenham's Circle C Program

```
# include<stdio.h>
# include<conio.h>
# include<math.h>
# include<dos. h>
# include<graphics. h>
void circlemid (int, int, int, int);
void plotpoints (int xcen, int ycen, int r)
{
int p, x, y;
x = 0;
y = r;
p = 3-2*r;
while (x<y)
```

NOTES

```
{
circlemid (xcen, ycen, x, y) ;
x++ ;
if (p<0)
p = p+4*x+6;
else
{
y—;
p = p+4* (x-y)+10;
}
}
}
void circlemid (int xcen, int ycen, int x, int y)
{
static int i = 0;
putpixel (xcen+x, ycen+y, i++) ;
putpixel (xcen-x, ycen+y, i++) ;
putpixel (xcen+x, ycen-y, i++) ;
putpixel (xcen-x, ycen-y, i++) ;
putpixel (xcen+y, ycen+x, i++) ;
putpixel (xcen-y, ycen+x, i++) ;
putpixel (xcen+y, ycen-x, i++) ;
putpixel (xcen-y, ycen-x, i++) ;
delay (10);
}
void main ()
{
int gd==DETECT, gm;
int i ;
clrscr () ;
initgraph(&gd, &gm," ");
for (i=0; i<=15; i++)
{
setbkcolor (i);
plotpoints (300, 230, i+i*3) ;
}
setcolor (4);
outtextxy (282, 227, "Ehtiram") ;
getch () ;
}
```

2.12 MIDPOINT CIRCLE GENERATION

As in the case of straight line, the best approximation of a circle will be defined by those pixels in the raster that fall the least distance from the true circle.

Decision variable d_i value is derived as (derivation is beyond the scope of this book).

$$d_i = 1 - r$$

Thereafter if $d_i < 0$ then only x is incremented

$$x_{i+1} = x_i + 1$$

and decision variable changes to

$$d_{i+1} = d_i + 2x_i + 3$$

and if $d_i >= 0$ then x & y are incremented

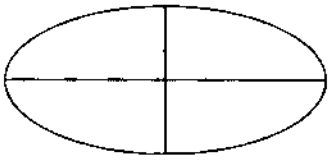
$$x_{i+1} = x_i + 1$$
$$y_{i+1} = y_i - 1$$
$$d_{i+1} = d_i + 2(x_i - y_i) + 5$$

This algorithm is quite similar to Bresenham's algorithm except that initial estimate and decision variable equations are changing by half.

NOTES

2.13 ELLIPSE GENERATION ALGORITHMS

Ellipse has two axes- major and minor axis. It is quite similar to circle.



```
//Ellipse Generating Algorithm
  //Simple Cartesian-Coordinate
  void ellipse(int xc, int yc, int xr, int yr)
  {
  int len;
     if (xr > yr)
     {
     int y12, y22;
     int y11 = yc, y21 = yc;
        for (int x=xc-xr+1; x<=xc+xr; ++x)
        {
           len = yr * sqrt(1 - SQR((x - xc) / double(xr)));
           y12 = yc + len;
           y22 = yc - len;
           line(x-1, y11, x, y12);
           line(x-1, y21, x, y22);
           y11 = y12;
           y21 = y22;
        }
     }
     else
     {
     int x12, x22;
     int x11 = xc, x21 = xc;
        for (int y=yc-yr+1; y<=yc+yr; ++y)
        {
           len = xr * sqrt(1 - SQR((y - yc) / double(yr)));
           x12 = xc + len;
           x22 = xc - len;
           line(x11, y-1, x12, y);
```

NOTES

```
                line(x21, y-1, x22, y);
                x11 = x12;
                x21 = x22;
            }
        }
    }
    //Ellipse Generating Algorithm
    //Simple Polar-Coordinate
    void ellipse(int xc, int yc, int xr, int yr)
    {
    const int ITERATIONS = 1000;
double theta;
    moveto(xc + xr, yc);
    for (int i=1; i<=ITERATIONS; ++i)
    {
        theta = 2 * PI * i / ITERATIONS;
        lineto(ROUND(xc+xr*cos(theta)), ROUND(yc+yr*sin(theta)));
    }
}
```

2.14 POLYGON FILLING

So far discussion has dealt with only the lines and characters. The world might seem rather dull if it were made out of straight lines. How interesting is the world of patterns, colors and shading. Unfortunately, much of the early graphics dealt with line drawings only. This was because the available devices (DVST, Plotters etc.) were line drawing devices. Raster display can display solid patterns and objects with no greater effort than that involved in showing their outlines. Coloring and shading is possible with raster technology.

We introduced new graphic primitive, the Polygon in the first chapter. We have already discussed what polygons are and how to represent them? We shall now learn how to determine if a point is inside a polygon. Finally, a method for filling in all inside pixels will be developed.

We wish to be able to represent a surface. One basic surface primitive is a polygon, a many sided figure. A polygon may be represented as a number of line segments connected end to end to form a closed figure. The line segments that make up the polygon boundary are called sides or edges. The endpoint of the sides are called the polygon's vertices. The simplest polygon is the triangle, having three sides and three vertex points.

We have already discussed that polygon can be divided into two classes: Concave and Convex. A Convex polygon is a polygon such that for any two points inside the polygon, all points on the line segment connecting them are also inside the Polygon. A concave polygon is the one, which is not convex. A triangle is always convex.

2.15 INSIDE TEST (FOR DETERMINING POINTS INSIDE THE POLYGON)

Having entered commands in the display file, we next might wish to be able to show polygon as solid object by setting the pixels inside the polygon as well as those on boundary.

Let us consider how we can determine whether or not a point is inside the polygon?

EVEN-ODD Method : One way of doing so is to construct a line segment between the point in question and a point known to be outside the polygon. Then we count how many intersections of the line segment with the polygon boundary occur.

If there are odd numbers of intersections then the point is INSIDE, an even number indicates that point is OUTSIDE. This is called the ODD-EVEN test for determining polygon interior points.

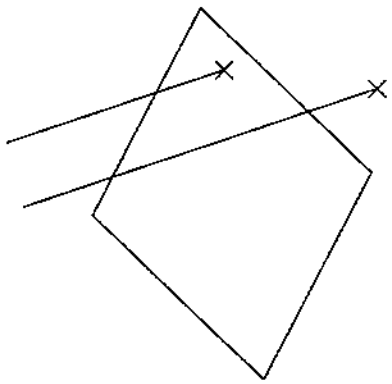


Fig. 2.7

If point of intersection is also the vertex then to handle such a case, we must look at the other endpoints of the two line segments which meet at this vertex. If these points lie on the same side of the constructed line then the point in question counts as even number of intersections. If they lie on the opposite side of the constructed line, the point is counted as single intersection.

There are many methods to fill a polygon. Some of them are :

Flood Fill Method : One way of filling the polygons is to first draw the edges of the polygon in a blank frame buffer. Then starting with some "Seed" point known to be inside the polygon, we set the intensity to the interior style and examine the neighboring pixels. We continue to set the pixel values in an increasing area until we encounter the boundary pixels. This method is called Flood fill because color flows from the "seed" pixel until reaching the polygon boundary.

When flood filling is used, the user will generally provide an initial pixel called a seed, the algorithm will inspect each of the surrounding eight pixels to determine whether the extent has been reached.

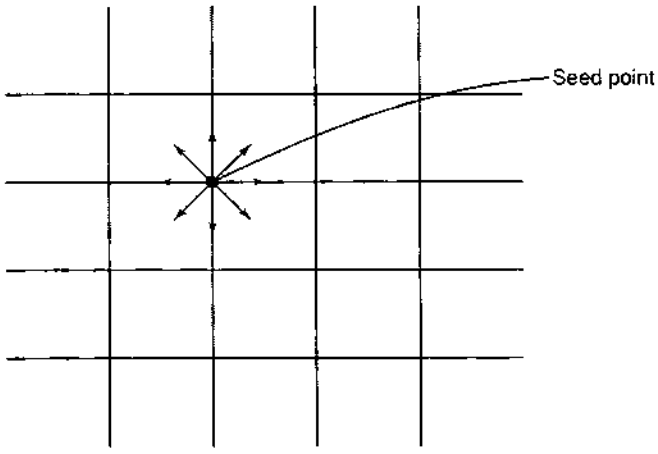


Fig. 2.8

NOTES

Implementing the Flood Fill Algorithm

Flood-Fill

Flood-fill also called *seed-fill*, is an algorithm that determines the area connected to a given node in a multi-dimensional array. It is used in the "bucket" fill tool of paint programs to determine which parts of a bitmap to fill with color and in puzzle games such as Minesweeper, Puyo Puyo, Lumines, and Magical Drop for determining which pieces are cleared.

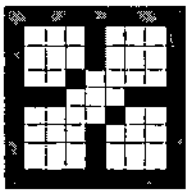


Fig. 2.9

The flood-fill algorithm takes three parameters: a start node, a target color, and a replacement color. The algorithm looks for all nodes in the array which are connected to the start node by a path of the target color, and changes them to the replacement color. There are many ways in which the flood-fill algorithm can be structured, but they all make use of a queue or stack data structure, explicitly or implicitly. One implicitly stack-based (recursive) flood-fill implementation (for a two-dimensional array) goes as follows:

Flood-fill (node, target-color, replacement-color):

1. If the color of *node* is not equal to *target-color*, return.
2. Set the color of *node* to *replacement-color*.
3. Perform **Flood-fill** (one step to the west of *node*, *target-color*, *replacement-color*).

Perform **Flood-fill** (one step to the east of *node*, *target-color*, *replacement-color*).

Perform **Flood-fill** (one step to the north of *node*, *target-color*, *replacement-color*).

Perform **Flood-fill** (one step to the south of *node*, *target-color*, *replacement-color*).

4. Return.

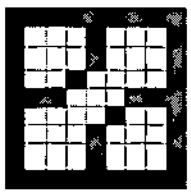


Fig. 2.10

Though easy to understand, this implementation is impractical in languages and environments where stack space is severely constrained (e.g., Java applets).

An explicitly queue-based implementation might resemble the following:

Flood-fill (node, target-color, replacement-color):

1. Set *Q* to the empty queue.
2. If the color of *node* is not equal to *target-color*, return.
3. Add *node* to the end of *Q*.
4. For each element *n* of *Q*:
5. Set the color of *n* to *replacement-color*.
6. If the color of the node to the west of *n* is *target-color*, add that node to the end of *Q*.

   If the color of the node to the east of *n* is *target-color*, add that node to the end of *Q*.

If the color of the node to the north of *n* is *target-color*, add that node to the end of *Q*.

If the color of the node to the south of *n* is *target-color*, add that node to the end of *Q*.

7. Continue looping until *Q* is exhausted.
8. Return.

Most practical implementations use a loop for the west and east directions as an optimization to avoid the overhead of stack or queue management:

NOTES

Flood-fill (node, target-color, replacement-color):

1. Set *Q* to the empty queue.
2. If the color of *node* is not equal to *target-color*, return.
3. Add *node* to *Q*.
4. For each element *n* of *Q*:
5. If the color of *n* is equal to *target-color*:
6. Set *w* and *e* equal to *n*.
7. Move *w* to the west until the color of the node to the west of *w* no longer matches *target-color*.
8. Move *e* to the east until the color of the node to the east of *e* no longer matches *target-color*.
9. Set the color of nodes between *w* and *e* to *replacement-color*.
10. For each node *n* between *w* and *e*.
11. If the color of the node to the north of *n* is *target-color*, add that node to *Q*.
    If the color of the node to the south of *n* is *target-color*, add that node to *Q*.
12. Continue looping until *Q* is exhausted.
13. Return.

Adapting the algorithm to use an additional array to store the shape of the region allows generalization to cover "fuzzy" flood filling, where an element can differ by up to a specified threshold from the source symbol. Using this additional array as an alpha channel allows the edges of the filled region to blend somewhat smoothly with the not-filled region.

Another method to fill is called **SCAN LINE** Method. One need to consider only the pixels which lie inside the polygon.

The algorithm can be sped up by filling lines. Instead of pushing each potential future pixel coordinate into the stack, it inspects the neighbour lines (previous and next) to find adjacent segments that may be filled in a future pass; the coordinates (either the start or the end) of the line segment are pushed on the stack. In most of cases this scanline algorithm is at least an order of magnitude faster than the per-pixel one.

Let us suppose that we start with the largest Y value and work our way down, scanning from left to right as we go in the raster manner. Our constructed test lines will be the horizontal lines at the current Y scanning value. Many problems in computer Graphics can be approached a scan line at a time. Algorithms, which take this approach, are called SCAN LINE algorithms.

We can draw to boundary of the polygon in a blank frame buffer and then examine the pixels in the box around the polygon, scan line by scan line. Moving around the scan line, when we encounter a pixel with the intensity of the boundary, we enter the polygon. Subsequent pixels are given the interior intensity until we encounter a second boundary pixel.

The problem with this scheme is that we must start with a frame buffer free of pixels with the polygon boundary intensity and we must be careful about cases where two polygon edges meet at a single point. We can avoid this problem by determining the polygon boundary values directly from the polygon instruction instead of from the frame buffer. Using the display file instructions, we can determine where the scan line crosses the polygon boundary.

"Consider only the sides which intersect the scan line. Our task becomes setting those pixels on the horizontal scan line which lie inside the polygon."

NOTES

2.16 OUTLINE FOR SCAN LINE FILLING ALGORITHM

The algorithm for filling a polygon should begin by ordering the polygon on the largest Y value. It should begin with the largest Y value and scan down the polygon with decreasing Y. For each Y, it should determine which sides can be intersected and determine the X values for these intersection points. The X values are sorted, paired and passed to the line drawing algorithms.

The algorithm which performs the YX scan and fills in the polygon is called FILL-POLYGON.

It starts by retrieving the polygon information from the display file and sorting it by the largest Y value. This is achieved by means of picking the value through algorithm specially designed for this purpose. The filling, in of the polygon is done by repeating these five steps:

1. First check if any additional polygon sides should be considered for this scan line algorithm.
2. The second step is to sort the X coordinates of the points where the polygon sides crosses the scan line.
3. The third step is to actually turn on or light the pixels between the polygon edges
4. Next the current scan line is decremented.
5. And the process repeats.

These steps are repeated until the scanning process has passed all polygon edges.

2.17 ANTI-ALIASING

In digital signal processing, **anti-aliasing** is the technique of minimizing the distortion artifacts known as aliasing when representing a high-resolution signal at a lower resolution. Anti-aliasing is used in digital photography, computer graphics, digital audio, and many other domains.

In the image domain, aliasing artifacts can appear as wavy lines or bands, or moiré patterns, or popping, strobing, or as unwanted sparkling; in the sound domain, as rough, inharmonic, or spurious tones, or as noise.

Anti-aliasing means removing signal components that have a higher frequency than is able to be properly resolved by the recording (or sampling) device. This removal is done before (re-) sampling at a lower resolution. When sampling is performed without removing this part of the signal, it causes undesirable artifacts such as the black and white noise near the top.

In signal acquisition and audio, anti-aliasing is often done using an analog anti-aliasing filter to remove the out of band component of the input signal prior to sampling with an analog to digital converter. In digital photography, optical anti-aliasing filters are made of birefringement materials, and smooth the signal in the spatial optical

domain. The anti-aliasing filter essentially blurs the image slightly in order to reduce resolution to below the limit of the digital sensor (the larger the pixel pitch, the lower the achievable resolution at the sensor level).

Basically there are two methods of anti-aliasing:

- Post-filtering
- Pre-filtering

NOTES

Post-filtering

In this process sample rate is increased and this is accomplished by increasing the resolution of the raster device. However there is a limit to the ability of CRT raster scan devices to display very fine rasters. This limit suggests that the raster tube calculated at higher resolution and displayed at lower resolution, using some type of averaging to obtain the pixel attributes at the lower resolution. Actually the concept of filtering originates from the field of signal processing. This technique is called post-filtering.

There are two main post-filtering techniques which are as follows:

1. Supersampling
2. Low-pass Filtering

1. **Supersampling: Supersampling** is an antia-liasing technique, the process of eliminating jagged and pixelated edges (aliasing). It is a method of smoothing images rendered in computer games or other programs that generate imagery.

   Aliasing occurs because real-world objects have continuous, smooth curves and lines. Monitors can only display discrete points of light called pixels. Since pixels are square and uniformly colored, lines become jagged.

   Supersampling is one of the ways of solving this problem. Samples are taken at *several* instances inside the pixel (not just at the center as default) and an average color value is calculated. This is achieved by rendering the image at a much higher resolution than the one being displayed, then downsampling (shrinking) it to the desired size, using the extra pixels for calculation. The result is smoother transitions from one line of pixels to another along the edges of objects.

   The number of samples determines the quality of the output. Options available normally range from 2*x* to 16*x*.

   Computational cost and adaptive supersampling: Supersampling is computationally expensive because it requires a lot of video card memory and memory bandwidth, since the amount of buffer used is several times larger. A way around this problem is *adaptive supersampling*. This works by acknowledging that very few pixels will actually be on a boundary, therefore only these need to be supersampled.

   At first only a few samples are made within a pixel. If these values are very similar, only these samples are used for determining color. If not, more are used. The result of this method is that a higher number of samples are calculated only where necessary, thus improving performance.

2. **Low-pass Filtering:** Filtering reduces noise errors in the signal. For most applications a **low-pass filter** is used. This allows through the lower frequency components but attenuates the higher frequencies. The cut-off frequency must be compatible with the frequencies present in the actual signal (as opposed to possible contamination by noise) and the sampling rate used for the Analog to Digital conversion.

   A low-pass filter that's used to prevent higher frequencies, in either the signal or noise, from introducing distortion into the digitized signal is known as an **anti-aliasing filter**. These generally have a sharper cut-off than the normal low-

pass filter used to condition a signal. Anti-aliasing filters are specified according to the sampling rate of the system and there must be one filter per input signal.

Pre-filtering

NOTES

In this method, a pixel is treated as a finite area rather than as a point and this technique, basically works on the true signal in the continuous space to derive proper values for individual pixels. There is one most popular pre-filtering technique, is termed as area sampling which is discussed as follows:

Area Sampling: In this approach, we superimpose a pixel grid pattern onto the continuous object definition. For each pixel area that intersects the object, we calculate the percentage of overlap by the object. This percentage determines the proportion of the overall intensity value of the corresponding pixel that is due to the object's contribution.

SUMMARY

- We looked at geometry involved in drawing lines. We derived equations and these equations can be used by computer programs to generate lines.
- There are two lines drawing algorithm which are employed by graphics designers/ developers:
  1. DDA (Digital Differential Analyzer) method
  2. Bresenham's line drawing algorithm.
- Bresenham's method is a better method since it avoids the use of floating point operations.
- DDA technique for obtaining a rasterized straight line is to solve the differential equation for a straight line

  $$D_Y/D_x = \text{constant} \quad \text{or} \quad \Delta_y/\Delta_x = (y_2 - y_1)/(x_2 - x_1)$$

  The solution is for the above differential equation is

  $$Y_{i-1} = Y_i + D_Y$$

  $$Y_{i-1} = Y_i + (y_2 - y_1)/(x_2 - x_1)\,\Delta_x$$

  where (x_1, y_1) and (x_2, y_2) are the end points of the required straight line and Yi is the initial value for any given step along the line.
- Bresenham's line algorithm is an efficient method for scan converting straight lines in that it uses only integer addition, subtraction and multiplication by 2 and floating point operations are avoided in Bresenham's algorithm. The computer can perform the operations of integer addition and subtraction very rapidly. The lines drawn are of superior quality as compared to DDA method.
- Screen coordinates increase from left to right and from top to bottom. The resolution of the system used to write programs in this book was 640 × 480 (VGA resolution).
- Circle has eight-point symmetry so it is very easy to plot the circle. Bresenham's circle algorithm uses this symmetry to draw the circle. Circle mid-point and ellipse generation algorithm is also given in the book.

REVIEW QUESTIONS

1. Write a program to draw a circle using Bresenham's algorithm.
2. The endpoints of a given line are (0, 0) and (6, 18). Compute each value of y as x steps from 0 to 6 and plot the result using DDA method.

3. What are the steps required to plot a line whose slope is between 0 and 45 degrees using Bresenham's method?
4. Indicate which raster locations would be chosen by Bresenham's algorithm when scan converting a line from pixel coordinate (1, 1) to pixel coordinate (8, 5).
5. Which one is better line algorithm: DDA or Bresenham's algorithm? Explain your answer.
6. Modify Bresenham's circle algorithm to circle mid point algorithm and develop a C program for the same.
7. What do you understand by Polygon Filling? Explain stack base full algorithm.
8. Discuss the techniques of anti-aliasing and describe the methods of anti-aliasing.

NOTES

FURTHER READINGS

- **Computer Graphic**: V.K. Pachghare, Laxmi Publications, 2007, Second edition.
- **Computer Graphics**: Prabhakar Gupta, Vineet Agarwal and Manish Varshney, Laxmi Publications, 2011.
- **Computer Graphics**: Rajiv Chopra, S. Chand Publisher, 2011.
- **Computer Graphics**: C.S. Verma, Ane Books, 2011.
- **Computer Graphics**: Pradeep K. Bhatia, I.K. International, 2009, pbk, Second Edition.
- **Computer Graphics**: Ruchi Mishra, Global Vision Publisher, 2010.

UNIT III

NOTES

2-DIMENSIONAL TRANSFORMATION

STRUCTURE

3.0 LEARNING OBJECTIVES

After going through this unit, you should be able to:

- describe about two dimensional transformation
- discuss about homogeneous coordinate system
- explain the term windowing and clipping
- describe the interaction picture construction techniques.

NOTES

3.1 INTRODUCTION

Almost all graphics systems allow the programmer to define picture that include a variety of transformations. For example—The programmer is able to magnify a picture so that detail appears more clearly, or reduce it so that more of the picture is visible.

Basic operations are as follows :

(*a*) Translation
(*b*) Scaling
(*c*) Rotation.

3.2 TWO DIMENSIONAL TRANSFORMATION

In 2D Transformation two dimensions are available, *i.e.*, X-Axis and Y-Axis.
Graphically it is represented as follows :

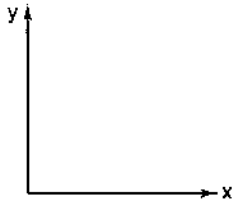


Fig. 3.1

(*a*) **Translation:** Translation means to move a picture from one point to another point *i.e.*, in translation the position of the picture is changed.

Example: If P (X, Y) and we translate it at (*h*, *k*) then the point P' becomes (X + *h*, Y + *k*).

So we can say that $P' = P + T$ ... [A]

$X' = X + t\,x$ ... [B]

$Y' = Y + t\,y$ ... [C]

Example:

If P (2, 3) and T = (5, 6) then calculate P'.

$X' = x + t\,x$; $X' = 2 + 5 = 7$

$Y' = y + t\,y$; $Y' = 3 + 6 = 9$

So the value of $P' = (7, 9)$. **Ans.**

NOTE We cannot represent the translation in the form of Matrix.

Example: Translate a polygon with coordinates A (2, 5), B (7, 10) and C (10, 2) by 3 units in X-direction and 4 units in Y-direction.

$$A' = A + T$$

$$A' = \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

NOTES

$$A' = \begin{bmatrix} 5 \\ 9 \end{bmatrix} \text{ Ans.}$$

$$B' = B + T$$

$$B' = \begin{bmatrix} 7 \\ 10 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$B' = \begin{bmatrix} 10 \\ 14 \end{bmatrix} \text{ Ans.}$$

$$C' = C + T$$

$$C' = \begin{bmatrix} 10 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$C' = \begin{bmatrix} 13 \\ 6 \end{bmatrix} \text{ Ans.}$$

(*b*) **Scaling:** Scaling means to change the size of the picture. If (P) is the initial point and after scaling the point is (P'), then

$$P' = S * P$$

Where $X' = Sx + X$ $\qquad Y' = Sy + Y$

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} * \begin{bmatrix} X \\ Y \end{bmatrix}$$

In the scaling we represent the Scaling as follows

$$P' = S * P$$

Where (S) is a two dimensional matrix, which is known as Scaling Matrix.

Example: *Scale the polygon with coordinates A (2, 5), B (7, 10) and C(10, 2) by two units in X-direction and two units in* Y*-direction.*

Solution:

Given $Sx = 2$ and $Sy = 2$

Therefore transformation matrix is

$$S = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The object matrix is $\begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix}$

So $$\begin{bmatrix} X_1' & Y_1' \\ X_2' & Y_2' \\ X_3' & Y_3' \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} * \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\begin{bmatrix} X_1' & Y_1' \\ X_2' & Y_2' \\ X_3' & Y_3' \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 14 & 20 \\ 20 & 4 \end{bmatrix} \text{ Ans.}$$

(*c*) **Rotation:** Rotation means, we want to rotate the picture by an angle.

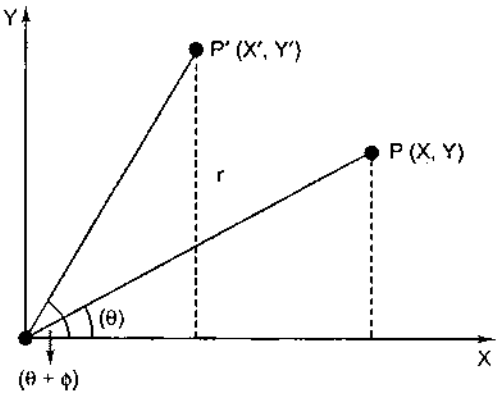


Fig. 3.2

In this case

$$X' = r\cos(\theta + \phi)$$

$$X' = r\cos\theta\cos\phi - r\sin\theta\sin\phi \quad \ldots [A]$$

But $X = r\cos\theta$

$Y = r\sin\theta$

Now from equation [A] we have

$$X' = X\cos\phi - Y\sin\phi \quad \ldots [B]$$

Now

$$Y' = r\sin(\theta + \phi)$$

$$Y' = r\sin\theta\cos\phi + r\cos\theta\sin\phi \quad \ldots [C]$$

But $X = r\cos\theta$

$Y = r\sin\theta$

Now from equation [C] we have

$$Y' = X\sin\phi + Y\cos\phi \quad \ldots [D]$$

Now from equation [B] and [D] we have

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} * \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$P' = R_{(\phi)} * P$$

Where $R_{(\phi)}$ is called Rotation Matrix.

NOTE Now in two dimensions, we cannot assign the value of translation in the Matrix form. Now to solve this problem we consider a new coordinate system that is known as Homogeneous Coordinate System.

3.3 HOMOGENEOUS COORDINATE SYSTEM

In a Homogeneous coordinate system, we add a new additional logical dimension. In Homogeneous coordinate system, the coordinate is

P (*wx*, *wy*, *w*) where *w* ! = 0

If we want to convert it in 2D, then the coordinate is

$$X = wx/w \text{ and } Y = wy/w$$

For Example : If in Homogeneous system, the coordinates is (1, 1, 0.5) then in 2D coordinate system the coordinates will be $\left(\frac{1}{0.5}, \frac{1}{0.5}\right) = (2, 2)$

The advantage of Homogeneous Coordinate System is that, by this coordinate system, we can convert the translation in the form of Matrix.

(a) **Translation:** If the initial coordinate is (X, Y, 1) and after the translation, the coordinate is (X', Y', 1) and the translation factor is t_x and t_y, then we can represent this as follows :

NOTES

$$X' = X + t_x$$
$$Y' = Y + t_y$$
$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$
$$P' = T * P$$

Where $T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$ is called Translation Matrix.

(*b*) **Scaling:** In Homogeneous coordinate system, the Scaling is defined as follows: If the initial point is (X, Y, 1) and after the Scaling the point is (X′, Y′, 1) then we can define it as follows :

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

(*c*) **Rotation:** In Homogeneous coordinate system, the Rotation is defined as follows: If the initial point is (X, Y, 1) and after the rotation, the point is (X′, Y′, 1) then we can define it as follows :

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Scaling a picture about an arbitrary point Q

If the picture is as follows and if we want to scale the picture about the point Q (*h, k*) then the following points are to be followed :

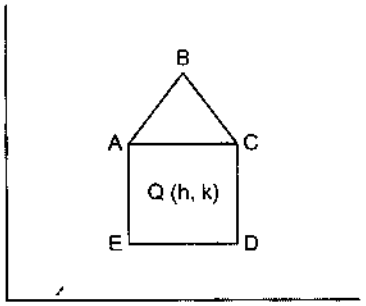
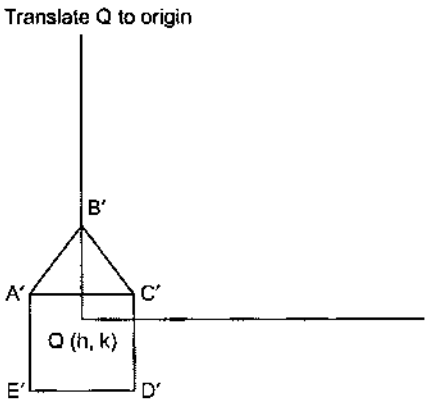
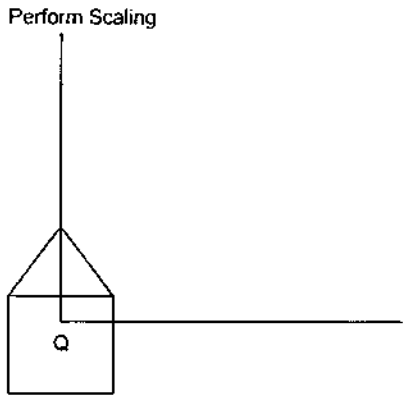


Fig. 3.3

Step 1.

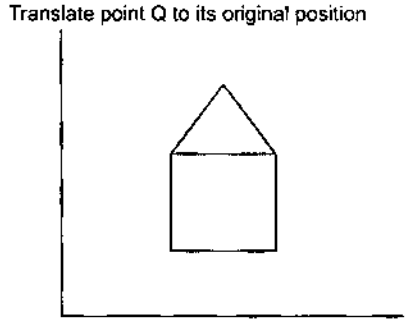


Step 2.



Step 3.

Translate point Q to its original position.



Mathematically it defines as follows

$$P' = T_{(-h,-k)} * P \quad \ldots [A]$$

$$P'' = S_{(Sx, Sy)} * P' \quad \ldots [B]$$

$$P''' = T_{(h,k)} * P'' \quad \ldots [C]$$

Now from equation [A], [B], and [C] we have

$$P''' = T_{(h,k)} * S_{(Sx,Sy)} * T_{(-h,-k)} * P \quad \ldots [D]$$

$$P''' = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -h \\ 0 & 1 & -k \\ 0 & 0 & 1 \end{bmatrix}$$

$$P''' = \begin{bmatrix} Sx & 0 & h \\ 0 & Sy & k \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -h \\ 0 & 1 & -k \\ 0 & 0 & 1 \end{bmatrix} * P$$

$$P''' = \begin{bmatrix} Sx & 0 & -Sxh + h \\ 0 & Sy & -Syk + k \\ 0 & 0 & 1 \end{bmatrix} * P$$

$$P''' = \begin{bmatrix} Sx & 0 & (1-Sx)h \\ 0 & Sy & (1-Sy)k \\ 0 & 0 & 1 \end{bmatrix} * P$$

$$P''' = M * P \quad \ldots [E]$$

Where (M) is called Composite Matrix.

Example: *Write composition of matrix for (2, 2) magnifying the picture ABCE about its center and center is at (10, 10).*

NOTES

Solution:

Step 1. Translate the picture at origin

$$P' = T_{(-10,-10)} * P \quad \ldots [A]$$

Step 2. $$P'' = S_{(2,2)} * P' \quad \ldots [B]$$

Step 3. $$P''' = T_{(10,10)} * P'' \quad \ldots [C]$$

Now from equation [A], [B], and [C] we have

$$P''' = T_{(10,10)} * S_{(2,2)} * T_{(-10,-10)} * P$$

$$P''' = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} * P$$

$$P''' = \begin{bmatrix} 2 & 0 & 10 \\ 0 & 2 & 10 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} * P$$

$$P''' = \begin{bmatrix} 2 & 0 & -10 \\ 0 & 2 & -10 \\ 0 & 0 & 1 \end{bmatrix} * P$$

So the composite matrix is

$$M = \begin{bmatrix} 2 & 0 & -10 \\ 0 & 2 & -10 \\ 0 & 0 & 1 \end{bmatrix} \text{ Ans.}$$

3.4 ROTATION OF A PICTURE ABOUT AN ARBITRARY POINT Q (H, K)

The following steps are followed for the rotation of a picture about an arbitrary point Q(*h, k*)

Step 1. Translate Q to origin.

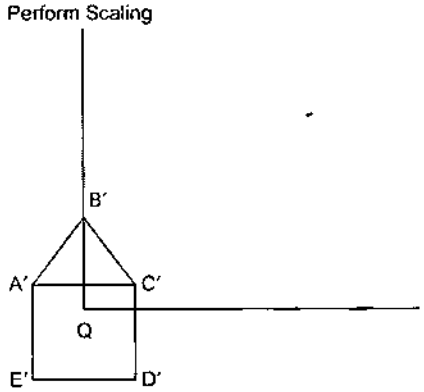
Step 2. Perform rotation.

Step 3. Back Translation of point Q to its original position.

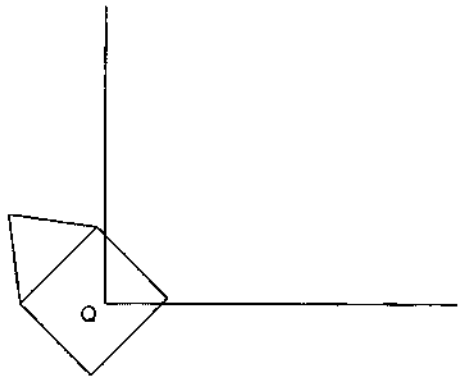
Graphically it is represented as follows :

Fig. 3.4

Step 1.

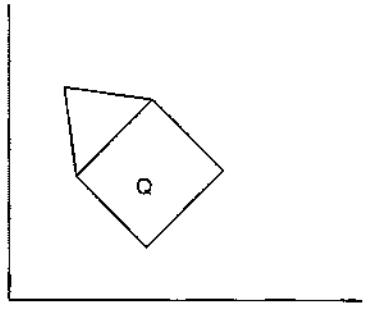


Step 2.



NOTES

Step 3.



Mathematically it is represented as follows

$$P' = T_{(-h,-k)} * P \quad \ldots [A]$$

$$P'' = R_{(\theta)} * P' \quad \ldots [B]$$

$$P''' = T_{(h,k)} * P'' \quad \ldots [C]$$

Now from equation [A], [B], and [C] we have

$$P''' = T_{(h,k)} * R_{(\theta)} * T_{(-h,-k)} * P$$

$$P''' = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -h \\ 0 & 1 & -k \\ 0 & 0 & 1 \end{bmatrix} * P$$

$$P''' = M * P$$

Where M is the Composite Matrix.

NOTE We know that $M_1 * M_2 \neq M_2 * M_1$. Where M_1 and M_2 are two Matrices. But in Computer Graphics $M_1 * M_2 = M_2 * M_1$ In the following case

Where are doing simultaneous types of a parameter.

(*a*) Firstly we translate any picture and then again translate the picture.

$$M_1 * M_2 = M_2 * M_1$$

(*b*) Firstly we scale the picture and again we scale the picture then

$$M_1 * M_2 = M_2 * M_1$$

(*c*) Firstly we rotate the picture and then again rotate the picture then

$$M_1 * M_2 = M_2 * M_1$$

Example:

Verify that $R_{(\theta)} * R_{(-\theta)} = I$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

NOTES

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \cos^2\theta + \sin^2\theta & \sin\theta\cos\theta - \sin\theta\cos\theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

I [Identity Matrix]

NOTE Anticlockwise Rotation (+)

Clockwise Rotation (–)

Example: *Given line PQ where P (1, 2) and Q (3, 10). Rotate this line about Q by 90°.*

Solution:

For rotation we perform the following steps :

Step 1. Translate Q to origin.

$$P' = T_{(-3,-10)} * P \quad \text{... [A]}$$

$$P'' = R_{(90°)} * P' \quad \text{... [B]}$$

$$P''' = T_{(3,10)} * P'' \quad \text{... [C]}$$

Now from equation [A], [B], and [C] we have

$$P''' = T_{(3,10)} * R_{(90°)} * T_{(-3,-10)} * P$$

$$\begin{bmatrix} X_1' & X_2' \\ Y_1' & Y_2' \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos 90 & -\sin 90 & 0 \\ \sin 90 & \cos 90 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 3 \\ 2 & 10 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} X_1' & X_2' \\ Y_1' & Y_2' \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 3 \\ 2 & 10 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} X_1' & X_2' \\ Y_1' & Y_2' \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 3 \\ 1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -2 & 0 \\ -8 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} X_1' & X_2' \\ Y_1' & Y_2' \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 3 \\ 8 & 10 \\ 1 & 1 \end{bmatrix}$$

So the value of P' (X_1', Y_1') = P' (11, 8)

And the value of Q' (X_2', Y_2') = Q' (3, 10) **Ans.**

Example: *Calculate A'B'C'D' where a picture is rotated 30° about point A.*

A (3, 1), B (3, 7), C (1, 3), and D (5, 3)

Solution:

We perform the following steps for rotation

$$P' = T_{(-3,-1)} * P \quad \text{... [A]}$$

$$P'' = R_{(30°)} * P' \quad \text{... [B]}$$

$$P''' = T_{(3,1)} * P'' \quad \text{... [C]}$$

Now from equation [A], [B], and [C] we have

$$P''' = T_{(3,1)} * R_{(30°)} * T_{(-3,-1)} * P$$

$$\begin{bmatrix} X_1' & X_2' & X_3' & X_4' \\ Y_1' & Y_2' & Y_3' & Y_4' \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 3 & 3 & 1 & 5 \\ 1 & 7 & 3 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

By this equation, we can calculate

(x_1', y_1'), (x_2', y_2'), (x_3', y_3'), and (x_4', y_4') Ans.

NOTES

3.5 MIRROR REFLECTION

The mirror reflection of a picture is shown as follows :

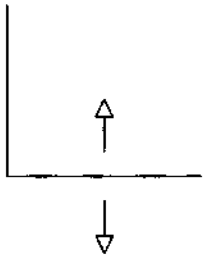


Fig. 3.5

This is Mirror Reflection

If the mirror reflect about X-Axis then the equation is

$$X' = X \text{ and } Y' = -Y$$

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

So $$P' = Mx * P$$

Where $$Mx = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ is called the Mirror Reflection Matrix about X-Axis.

If the mirror reflect about Y-Axis then the equation is

$$X' = -X \text{ and } Y' = Y$$

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

So $$P' = My * P$$

Where $$My = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ is called the Mirror Reflection Matrix about Y-Axis.

NOTES

Mirror Reflection of a Picture About a Line $y = m\,x$

If we apply the Mirror Reflection of a picture about a line $y = m\,x$ then we followed the following points

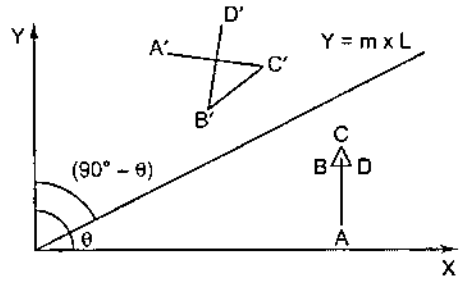


Fig. 3.6

The following are the points which we can mirror reflect about a line through X-axis.

Step 1. Rotate Line L by $(-\theta)$

Where $\theta = \tan^{-1}(m)$

$$P' = R_{(\theta)} * P \quad \text{... [A]}$$

Step 2. Take Mirror reflection about X-Axis

$$P'' = Mx * P' \quad \text{... [B]}$$

Step 3. Rotate Line L by $(+\theta)$

$$P''' = R_{(\theta)} * P'' \quad \text{... [C]}$$

Now from equation [A], [B], and [C] we have

$$P''' = R_{(\theta)} * Mx * T_{(-\theta)} * P \quad \text{Ans.}$$

The following are the points by which we can Mirror Reflect the image about line through Y-Axis.

Step 1. Rotate Line L by $(90 - \theta)$

$$P' = R_{(90-\theta)} * P \quad \text{... [A]}$$

Step 2. Take Mirror reflection about Y-Axis.

$$P'' = My * P' \quad \text{... [B]}$$

Step 3. Rotate Line L by $(90 - \theta)$

$$P''' = R_{(-(90-\theta))} * P'' \quad \text{... [C]}$$

Now from equation [A], [B], and [C] we have

$$P''' = R_{[-(90-\theta)]} * My * T_{(90-\theta)} * P \quad \text{Ans.}$$

Mirror Reflection of a Picture About a Line $y = m\,x + c$

If we apply the Mirror Reflection of a picture about a line $y = m\,x$ then we followed the following points

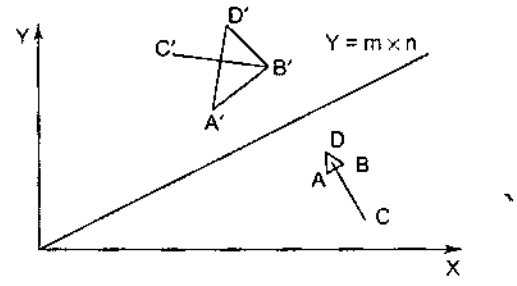


Fig. 3.7

In this case the following steps are to be followed

Step 1. Translate the Line by (0, – C)

$$P' = T_{(0,-C)} * P \quad \ldots [A]$$

Step 2. Rotate by (– θ) Where $\theta = \tan^{-1}(m)$

$$P'' = R_{(-\theta)} * P' \quad \ldots [B]$$

Step 3. Take Mirror Reflection about X-Axis

$$P''' = Mx * P'' \quad \ldots [C]$$

Step 4. Rotate the Line by (θ)

$$P'''' = R_{(\theta)} * P''' \quad \ldots [D]$$

Step 5. Back Translate (0, C)

$$P''''' = T_{(0,C)} * P'''' \quad \ldots [E]$$

Now from equation [A], [B], [C], [D] and [E] we have

$$P''''' = T_{(0,C)} * R_{(\theta)} * Mx * R_{(-\theta)} * T_{(0-C)} * P \text{ Ans.}$$

NOTES

3.6 SHEARING

A transformation that change the shape of an object is called the Shear transformation. Two common shearing transformations are used. One shift X coordinate values and other shift Y coordinate values. However in both the cases only one coordinates (X or Y) changes its coordinates and other preserve its values.

Shearing About X-axis

The X Shear preserves the Y coordinates, but changes the X values which changes vertical lines to tilt right or left as shown in the following figure

Fig. 3.8(a) Original object

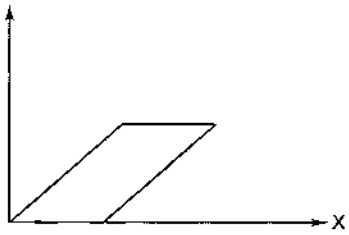


Fig. 3.8(b) Objects after X shear

$$X' = X + S_{hx} * Y$$

$$Y' = Y$$

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$P' = S_{(\text{Shear X})} * P$$

Where $S_{(\text{Shear X})} = \begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is called shearing Matrix about X-Axis.

NOTES

Shearing About Y-axis

The Y Shear preserves the X coordinates, but changes the Y values which caused horizontal lines to transform in to lines which slope up or down as shown in the following figure

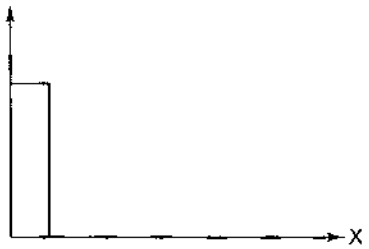


Fig 3.9(a) Original object

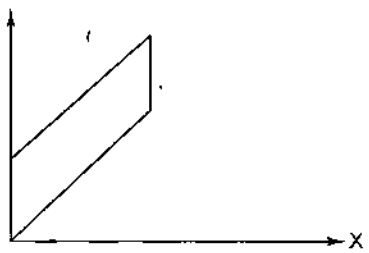


Fig 3.9(b) Object after Y shear

$$X' = X$$

$$Y' = Y + S_{hy} * X$$

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ S_{hy} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$P' = S_{(Shear\ Y)} * P$$

Where $S_{(Shear\ Y)} = \begin{bmatrix} 1 & 0 & 0 \\ S_{hy} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is called shearing Matrix about Y-Axis.

3.7 TRANSFORMATIONS ROUTINES

In the Transformation Routines, we have to write the functions for translation scaling and rotation and mirror reflection etc.

3.8 DISPLAYING PROCEDURE

The calls which involve establishment of a transformation are called display procedure calls and these calls can be nested, *i.e.*, there can be multiple transformations and subprograms which draw sub pictures are known as Display Procedures.

For example if we want to scale any object about a given point then first we translate the point in to origin and then apply scaling and again we translate back to original position. In this situation user would add one or more translation levels to the system.

Basically a procedure call involve the following steps :

1. Saving the overall transformation matrix
2. Multiplying the overall transformations matrix on the left by the transformation in the call to form a new overall transformation matrix

A routine from a display procedure involved the following steps :

1. Restoring the overall transformation matrix-from the value saved
2. Returning control to the calling program.

NOTES

Example. *Perform a 60° rotation of triangle A (0, 0), B (1, 1), and C (– 1, – 1).*

(a) About Origin *(b) About P (0, 1)*

Solution.

(a) When we rotate any image about origin, then there is no need for transformation.

$$P' = R_{(60)} * P$$

$$\begin{bmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \cos 60 & -\sin 60 & 0 \\ \sin 60 & \cos 60 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{2}\right) & \left(\frac{-\sqrt{3}}{2}\right) & 0 \\ \left(\frac{\sqrt{3}}{2}\right) & \left(\frac{1}{2}\right) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & -1 \\ 0 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & \left(\frac{1-\sqrt{3}}{2}\right) & \left(\frac{\sqrt{3}-1}{2}\right) \\ 0 & \left(\frac{\sqrt{3}+1}{2}\right) & \left(\frac{-\sqrt{3}-1}{2}\right) \\ 1 & 1 & 1 \end{bmatrix}$$

Now comparing the values we have

$$X_1' = 0$$

$$Y_1' = 0$$

$$X_2' = \left(\frac{1-\sqrt{3}}{2}\right)$$

$$Y_2' = \left(\frac{\sqrt{3}+1}{2}\right)$$

$$X_3' = \left(\frac{\sqrt{3}-1}{2}\right)$$

$$Y_3' = -\left(\frac{\sqrt{3}+1}{2}\right) \quad \text{Ans.}$$

NOTES

(b) About P (0, 1)

Step 1. Translate point P to origin

$$P' = T_{(0,-1)} * P \quad \ldots [A]$$

Step 2. Perform the Rotation

$$P'' = R_{(60)} * P' \quad \ldots [B]$$

Step 3. Back Translate to point P

$$P''' = T_{(0,1)} * P'' \quad \ldots [C]$$

Now from equation [A], [B], and [C] we have

$$P''' = T_{(0,1)} * R_{(60)} * T_{(0,-1)} * P$$

From this equation we can find the values of (X_1', Y_1'), (X_2', Y_2'), and (X_3', Y_3'). **Ans.**

Example. *Reflect the triangular polygon whose vertices are A (– 1, 0), B (0, – 2), and C (1, 0) about the line y = x +2.*

Solution.

$$Y = x + 2 \quad \ldots [1]$$

Compare equation [1] by $y = mx + c$

$$M = 1$$

$$C = 2$$

Step 1. Translate the Line by (0, – 2)

$$P' = T_{(0,-2)} * P \quad \ldots [A]$$

Step 2. Rotate line L by (– 45)

$$P'' = R_{(-45)} * P' \quad \ldots [B]$$

Step 3. Take Mirror Reflection about X-Axis

$$P''' = Mx * P'' \quad \ldots [C]$$

Step 4. Back Rotate the Line by (45)

$$P'''' = R_{(45)} * P''' \quad \ldots [D]$$

Step 5. Back Translate (0, 2)

$$P''''' = T_{(0,2)} * P'''' \quad \ldots [E]$$

Now from equation [A], [B], [C], [D] and [E] we have

$$P''''' = T_{(0,2)} * R_{(45)} * Mx * R_{(-45)} * T_{(0-2)} * P$$

Now from this equation we can calculate the desired points. **Ans.**

3.9 WINDOWING AND CLIPPING

Typically, a graphics package allows us to specify which part of a defined picture is to be displayed and where that part is to be displayed on the display device. Furthermore, the package also provides the use of scaling, translation and rotation techniques described in the previous chapter to generate a variety of different views of a single picture. We can generate different views of a picture by applying the appropriate scaling, translation and rotation. The process of selecting and viewing the picture with different views is called Windowing and a process which divides each element of the picture into its visible and invisible positions, allowing the invisible position to be discarded is called Clipping.

3.10 VIEWING TRANSFORMATION

We know that the picture is stored in the computer memory using any convenient cartesian coordinate system, referred to as Word Coordinate System (WCS).

However, when picture is displayed on the display device it is measured in Physical Device Coordinate System (PDCS) corresponding to the display device. Therefore displaying an image of a picture involves mapping the coordinates of the points and lines that form the picture into the appropriate physical device coordinate where the image is to be displayed. This mapping of coordinates is achieved with the use of coordinate transformation known as Viewing Transformations.

NOTES

Window

The portion of the picture which we want to display is called Window.

View Port

View Port means area of display device on to which window is mapped.

The viewing transformation which maps picture coordinates in the Word Coordinate System to display coordinate in Physical Device Coordinate System is performed by the following transformations :

(*a*) **Normalization Transformation (N)**

(*b*) **Workstation Transformation (W)**

(*a*) **Normalization Transformation:** We know that, different display devices may have different screen sizes as measure in pixels. Size of the screen in pixel increases as resolution of the screen increases. To avoid this and make our programs to be device independent, we have to define the picture coordinates in some units other than pixels and use the interpreter to control these coordinates to appropriate pixel values for the particular display device. The device independent units are called the Normalized device coordinates. In these units the screen measures 1 unit width and 1 unit length which is shown in the following figure.

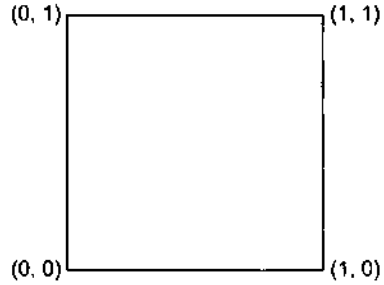


Fig. 3.10(a)

The lower left corner of the screen is the origin, and the upper left corner is the point (1, 1). The point (0.5, 0.5) is the center of the screen no matter what the physical dimensions or resolutions of the actual display device may be. The INTERPRETER uses a simple linear formula to convert the Normalized device coordinate to the actual device coordinates.

$$X = X_n * X_w \quad [A]$$

$$Y = Y_n * Y_h \quad [B]$$

Where

X = Actual device X coordinates.

Y = Actual device Y coordinates.

X_n = Normalized X coordinates.

Y_n = Normalized Y coordinates.

X_w = Width of actual screen in pixels.

Y_h = Height of actual screen in pixels.

NOTES

The transformation which maps the word coordinates to Normalized device coordinate is called Normalized transformation. It involves Scaling of X and Y, thus it is also referred to as Scaling transformation.

(*b*) **Workstation Transformation**: The transformation which maps the Normalized device coordinates to physical device coordinates is called Workstation transformation.

The Viewing transformation is the combination of Normalization transformation and Workstation transformation which is defined as follows :

$$V = W * N \quad \text{... [A]}$$

Where N is the Normalized transformation which mapped the word coordinate system to Normalized coordinate system and W is the transformation which mapped the Normalized device coordinate system to device coordinate system.

Now if there is a Window (X W min, Y W min, X V max, Y V max). Now we want to map the window on to the View Port.

It is described in the following figure :

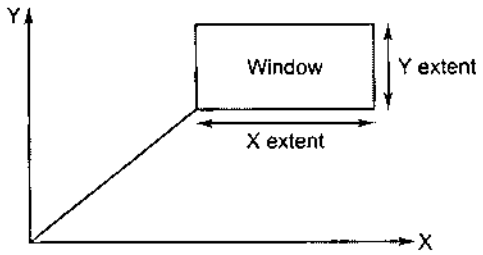


Fig. 3.10(b)

The following steps are to be followed :

Step 1. Translate the window on to origin

$$P' = T_{(-X\,W\,min.\,-Y\,W\,min)} \quad \text{... [A]}$$

Step 2. Taking the scaling

$$P'' = S_{(s\,x.\,s\,y)} * P' \quad \text{... [B]}$$

Step 3. Translate it to View Port

$$P''' = T_{(X\,W\,min.\,Y\,W\,min)} \quad \text{... [C]}$$

Now from equations [A], [B], and [C], we have

$$P''' = T_{(X\,W\,min.\,Y\,W\,min)} * S_{(s\,x.\,s\,y)} * T_{(-X\,W\,min.\,-Y\,W\,min)} \quad \text{... [D]}$$

But in the case of Normalized device coordinate system coordinates of View Port will be (0, 0, 1, 1).

So
$$X\,W\,min = 0$$
$$Y\,V\,min = 0$$
$$X\,V\,max = 1$$
$$Y\,V\,max = 1$$

Now

$$T_{(X\,W\,min.\,Y\,W\,min)} = \begin{bmatrix} 1 & 0 & XV\,min \\ 0 & 1 & YV\,min \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_{(0.0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

So from equation [D] we have

$$V = I * S_{(s x, s y)} * T_{(-X W min, -Y W min)}$$

$$V = S_{(s x, s y)} * T_{(-X W min, -Y W min)} \quad \ldots [E]$$

Equation [E] represent the viewing transformation matrix in the Normalized device coordinate system.

Where $$Sx = \frac{XV\ max - XV\ min}{XW\ max - XW\ min}$$

And $$Sy = \frac{YV\ max - YV\ min}{YW\ max - YW\ min}$$

NOTES

The following figure shows the complete viewing transformation

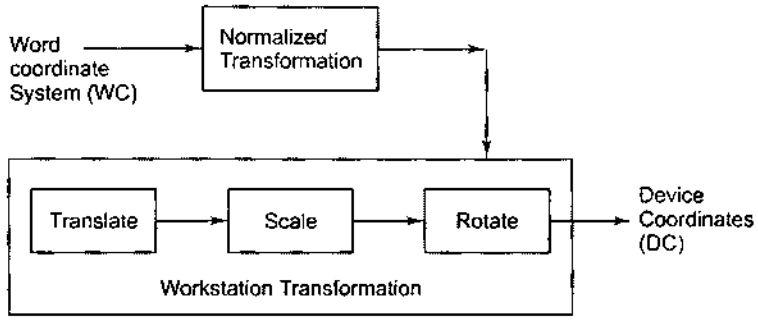


Fig. 3.11

3.11 APPLICATION OF VIEWING TRANSFORMATION

The following are the applications of viewing transformation:

1. By changing the position of the View Port, we can view object at different positions on the display area of an output device
2. By viewing the size of view ports, we can change the size and properties of displayed objects
3. We can achieve zooming effects by successively mapping different sized windows on a fixed size view port.

Example. *Find the Normalized transformation for mapping a window (1, 1, 3, 5) on to a view port (A) (0, 0, 0.5, 0.5) and on to a view port (B) (0, 0, 1, 1).*

Solution.

(A) Given Window (1, 1, 3, 5)

View port (0, 0, 0.5, 0.5)

X W min = 1
Y W min = 1
X W max = 3
Y W max = 5
X U min = 0
Y U min = 0
X U max = 0.5
Y U max = 0.5

So this is Normalized Device Coordinate System.

$$Sx = \frac{XV\ max - XV\ min}{XW\ max - XW\ min}$$

NOTES

$$Sx = \frac{0.5 - 0}{3 - 1}$$

$$Sx = \frac{0.5}{2}$$

$$Sx = \frac{1}{4}$$

$$Sy = \frac{YV\ max - YV\ min}{YW\ max - YW\ min}$$

$$Sy = \frac{0.5 - 0}{5 - 1}$$

$$Sy = \frac{0.5}{4}$$

$$Sy = \frac{1}{8}$$

Now for viewing transformation matrix

Step 1. Translate the window on to origin

$$P' = T_{(-XW\ min, -YU\ min)} \quad \ldots [A]$$

Step 2. Taking the scaling

$$P'' = S_{(sx, sy)} * P' \quad \ldots [B]$$

Step 3. Translate it to View Port

$$P''' = T_{(XV\ min, YV\ min)} \quad \ldots [C]$$

Now from equations [A], [B], and [C], we have

$$P''' = T_{(XV\ min, YV\ min)} * S_{(sx, sy)} * T_{(-XW\ min, -YW\ min)} \quad \ldots [D]$$

Now the viewing transformation matrix

$$V = T_{(XV\ min, YV\ min)} * S_{(sx, sy)} * T_{(-XW\ min, -YW\ min)}$$

But in case of Normalized Device Coordinate System, we know that

$$T_{(XV\ min, YV\ min)} = T_{(0,0)} = I$$

So

$$V = S_{(sx, sy)} * T_{(-XW\ min, -YW\ min)}$$

$$V = S\left(\frac{1}{4}, \frac{1}{8}\right) * T(-1, -1)$$

$$V = \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{8} & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$V = \begin{bmatrix} \frac{1}{4} & 0 & \frac{-1}{4} \\ 0 & \frac{1}{8} & \frac{-1}{8} \\ 0 & 0 & 1 \end{bmatrix} \text{ Ans.}$$

(B) Given Window (1, 1, 3, 5)
View port (0, 0, 1, 1)

$$Sx = \frac{XU\ max - XU\ min}{XW\ max - XW\ min}$$

$$Sx = \frac{1-0}{3-1}$$

$$Sx = \frac{1}{2}$$

$$Sy = \frac{YU\ max - YU\ min}{YW\ max - YW\ min}$$

$$Sy = \frac{1-5}{5-0}$$

$$Sy = \frac{1}{4}$$

In this case the Viewing transformation matrix-

$$V = S_{(sx,\,sy)} * T_{(-XW\,min.\,-YW\,min)}$$

$$V = S\left(\frac{1}{2}, \frac{1}{4}\right) * T(-1, -1)$$

$$V = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$V = \begin{bmatrix} \frac{1}{2} & 0 & \frac{-1}{2} \\ 0 & \frac{1}{4} & \frac{-1}{4} \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{Ans.}$$

Example. *Find the viewing transformation for mapping a window A B C D A (1, 1), B (5, 3), C (4, 5), and D (0, 3) on to the Normalized view port (0, 0, 1, 1).*

Solution.

So $\tan\theta = (2/4) = (1/2)$

Then $\sin\theta = \frac{1}{\sqrt{5}}$

And $\cos\theta = \frac{2}{\sqrt{5}}$

$$AB = \sqrt{(5-1)^2 + (3-1)^2}$$

$$AB = \sqrt{16+4}$$

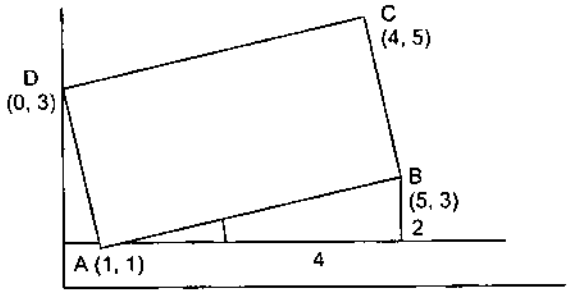
$$AB = \sqrt{20}$$

$$AB = 2\sqrt{5}$$

NOTES

NOTES

Now the following steps are to be followed



Step 1. Translate the window by (– 1, – 1)

Step 2. Rotate it by – θ

Step 3. Taking Scaling

After performing these steps the shape is as follows :

In this case

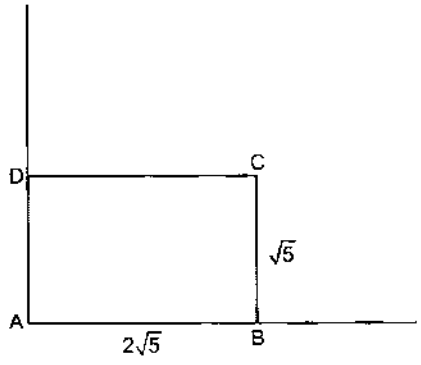
$$Sx = \frac{1-0}{2\sqrt{5}}$$

$$Sx = \frac{1}{2\sqrt{5}}$$

$$Sy = \frac{1-0}{\sqrt{5}}$$

$$Sy = \frac{1}{\sqrt{5}}$$

So the Viewing transformation matrix



$$V = S_{(s\,x,\,s\,y)} * R_{(-\theta)} * T_{(-1,\,-1)}$$

$$V = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$V = \begin{bmatrix} \frac{1}{2\sqrt{5}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

After multiplying these matrices we can find the viewing transformation matrix.

3.12 ASPECT RATIO

There are two types of aspect ratio :

(a) Windows Aspect Ratio

(b) View Port Aspect Ratio.

(a) **Windows Aspect Ratio:** The Window Aspect Ratio is calculated as follows :

$$a_w = \frac{XW\ max - XW\ min}{YW\ max - YW\ min} \quad \ldots |A|$$

(b) **View Port Aspect Ratio:** The View Port Aspect Ratio is calculated as follows :

$$a_v = \frac{XV\ max - XV\ min}{YV\ max - YV\ min} \quad \ldots |B|$$

NOTES

3.13 CLIPPING

Clipping means that the portion which is inside of the window is displayed, but the portion which is outside of the window is not displayed. The procedure that identifies the portion of a picture that are either inside or outside of a specified region of space is referred to as Clipping. The region against which an object is to be clipped is called a Clip Window or Clipping Window.

It unusually is in a rectangular shape as shown in the following figure :

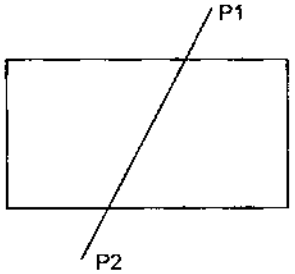


Fig. 3.12 (a) Before clipping

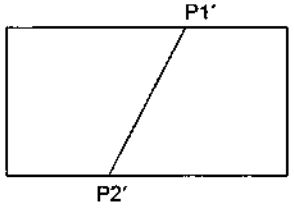


Fig. 3.12 (b) After clipping

3.14 POINT CLIPPING

The points are said to be interior to the clipping window if

$XW\ min \leq X \leq XW\ max$

AND $YW\ min \leq Y \leq YW\ max$

NOTE The equal sign indicates that points on the window boundary are included within the window.

3.15 LINE CLIPPING

The Line are said to be interior to the Clipping Window and hence visible if both end points are interior to the Window.

We will now discuss the Line Clipping Algorithms

1. **Sutherland Cohen Line Clipping Algorithm:** The algorithm is used for clipping of a line against a rectangular window (X W min, Y W min, X W max, Y W max).

This algorithm uses a 4 bit code.

NOTES

Graphically it is represented as follows :

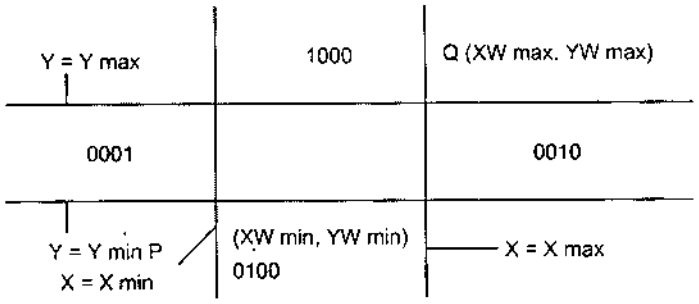


Fig. 3.13

In this algorithm, the following steps are followed :

Calculate the end points code of |A| and calculate the end point code of |B|.

The following procedure is followed for calculating the end point code :

(*a*) For Bit 1

If YW min – YW max ≤ 0 then

Code = 0

Else

Code = 1

(*b*) For Bit 2

If YW – YW max – Y ≤ 0 then

Code = 0

Else

Code = 1

(*c*) For Bit 3

If X – XW max ≤ 0 then

Code = 0

Else

Code = 1

(*d*) For Bit 4

If XW min – X ≤ 0 then

Code = 0

Else

Code = 1

- If the end point code of both points (A) and (B) is 0000 and 0000 then
- The line is completely visible
- Else we take AND operation of end point code of A B.
- If after the AND operation, the result is 0000 then the line is partially visible and it is called as Clipping Coordinates.
- If after the AND operation the result is not 0000 then the line completely clipped or completely invisible.

3.16 CLIPPING COORDINATE

1. The line which is partially visible, that is called Clipping Coordinate.
2. Now we find that which portion of line is visible and which portion of line is invisible.

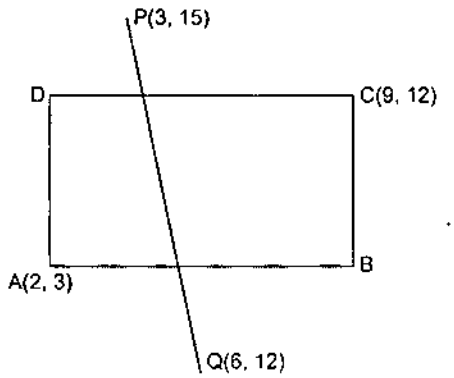
3. In this case, we find the intersection point of line with windowing boundary and the portion, which is inside the window is visible and the portion, which is outside of the window is called Clipping Portion.

NOTE For finding the intersection point, we have to use the following equation of the line :

$$X = X_1 + (X_2 - X_1)\, t \quad \text{... [A]}$$

$$Y = Y_1 + (Y_2 - Y_1)\, t \quad \text{... [B]}$$

Example. *We find that which portion of the line is clipped and which portion is visible.*



Solution. In this case

XW min = 2

YW min = 3

XW max = 9

YW max = 12

Step 1. Firstly we find the end point code of P and Q.

So for the end point code of P

X = 3 and Y = 15

Bit-1 (Y – YW max) → (15 – 12) → 3 > 0 → code = 1

Bit-2 (YW min – Y) → (3 – 15) → –12 < 0 → code = 0

Bit-3 (X – XW max) → (3 – 9) → – 6 < 0 → code = 0

Bit-4 (XW min – X) → (2 – 3) → – 1 < 0 → code = 0

So the 4 bit code of point P is 1000

Now for the end point code of Q

X = 6 and Y = 2

Bit-1 (Y – YW max) → (2 – 12) → –10 < 0 → code = 0

Bit-2 (YW min – Y) → (3 – 2) → 1 > 0 → code = 1

Bit-3 (X – XW max) → (6 – 9) → – 3 < 0 → code = 0

Bit-4 (XW min – X) → (2 – 6) → – 4 < 0 → code = 0

So the 4 bit code of point P is 0100

Now the 4 bit code of P and Q, *i.e.*, not 0000 and 0000, so the line PQ is not completely visible.

Step 2. Now we impose the AND operation between the code of P and Q.

4 bit code of P = 1000

4 bit code of Q = 0100

AND operation → 0000

Now after the AND operation the 4 bit code is 0000.

So now in this case line PQ is called Clipping coordinate.

Now we have to find the intersection point.

NOTES

NOTES

Now from the 4 bit code we know that it intersects.

$$Y = Y \text{ max and } Y = Y \text{ min}$$

Now for first point say I, it intersects at Y = Y max

$$X = X_1 + (X_2 - X_1)t \quad \text{... [A]}$$

$$Y = Y_1 + (Y_2 - Y_1)t \quad \text{... [B]}$$

Now from equation [B] we have, putting Y = Y max

$$Y \text{ max} = y_1 + (y_2 - y_1)\, t$$

$$t = (Y \text{ max} - Y_1) / (Y_2 - Y_1)$$

$$t = (12 - 15) / (2 - 15)$$

$$t = (-3 / -13)$$

$$t = (3 / 13)$$

Now we put the value of t in equation [A]

$$X = X_1 + (X_2 - X_1)t$$

$$X = 3 + (6 - 3) * (3/13)$$

$$X = (48 / 13)$$

So the coordinate of I_1 is (48/13, 12)

Now for point I_2

From the 4 bit code it will intersect at Y = Y min

Now putting Y = Y min in equation [B]

$$Y \text{ min} = Y_1 + (Y_2 - Y_1)t$$

$$t = (Y \text{ min} - Y_1) / (Y_2 - Y_1)$$

$$t = (3 - 15) / (2 - 15)$$

$$t = (-12 / -15)$$

$$t = (12 / 13)$$

Now we put the value of t in equation [B]

$$X = X_1 + (X_2 - X_1)t$$

$$X = 3 + (6 - 3) * (12/13)$$

$$X = (75 / 13)$$

So the coordinate of I_2 is (75/13, 3)

Now we find the code of I_2 is (75/13, 3)

I_1 (48/13, 12) and I_2 (75/13, 3)

For 4 bit code of I

Bit-1 (Y – YW max) → (12 – 12) → – 0 < 0 → code = 0

Bit-2 (YW min – Y) → (3 – 3) → 0 < 0 → code = 0

Bit-3 (X – XW max) → (75/13 – 9) → code = 0

Bit-4 (XW min – X) → (2 – 75/13) → – 4 < 0 → code = 0

So the end point code of I_2 is 0000

Now the end point code of I_1 and I_2 is (0, 0, 0, 0) and (0, 0, 0, 0). So the portion I_1 I_2 is visible. And the portion P I_1 and I_2 Q is clipped.

3.17 MIDPOINT SUBDIVISION ALGORITHM

1. We have seen that, the Sutherland Cohen subdivision line clipping requires the calculation of the intersection of the line with the window edge.
2. This calculation can be avoided by repetitively subdividing the line at its mid-point.

3. Like previous algorithm, initially the line is tested for visibility. If line is completely invisible it is rejected.
4. If line is partially visible then it is subdivided in two equal parts. The visibility tests are then applied to each half. This subdivision process is repeated until we get completely invisible line segments.

This is illustrated in following figure :

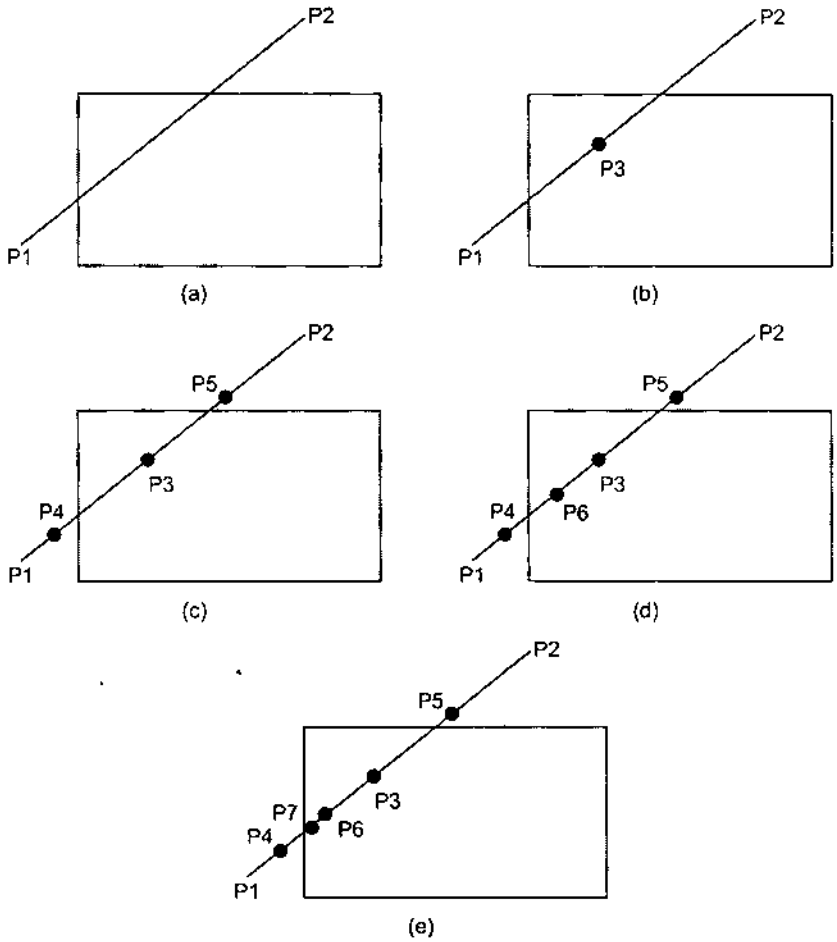


Fig. 3.14

Algorithm

1. Read two end points of the line say P_1 (x_1, y_1) and P_2 (x_2, y_2).
2. Read two corners (left top and right bottom) of the window say (XW min, YW min, XW max, YW max).
3. Assign the region codes for two end points using the following steps :

(A) For Bit 1

If Y – YW max ≤ 0 then

Code = 0

Else

Code = 1

(B) For Bit 2

If YW man Y ≤ 0 then

Code = 0

Else

Code = 1

NOTES

NOTES

(C) For Bit 3

If X – XW max ≤ 0 then

Code = 0

Else

Code = 1

(D) For Bit 4

If XW min X ≤ 0 then

Code = 0

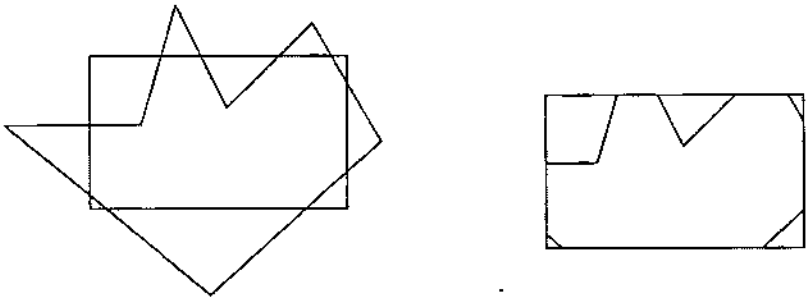
Else

Code = 1

4. Check for visibility of line.
   (*a*) If the end point code of both points are zero then
   The line is completely visible.
   Hence draw the line and go to step 6.
   (*b*) If the end point code of both points are not zero and the logical AND of them is also non zero then the line is completely invisible. So reject the line and go to Step 6.
   (*c*) If end points code for two end points do not satisfy the coordinate in 4 (*a*) and 4 (*b*). Then the line is partially visible.
5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both sub divided line segments until you get completely visible and completely invisible line segment.
6. Stop.

3.18 POLYGON CLIPPING

1. A Polygon is nothing but the collection of lines. Therefore, we might think that line clipping algorithm can be used directly for polygon clipping.
2. However, when a closed polygon is clipped as a collection of lines with line clipping algorithm, the original closed polygon becomes one or more open polygon or discrete lines as shown in the following figure. Thus we need to modify the line clipping algorithm to clip polygon.

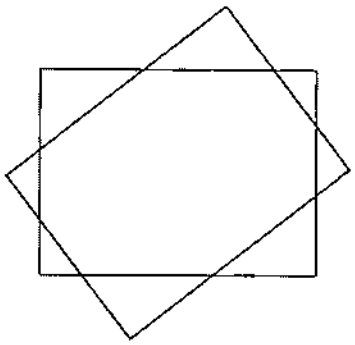


(a) Before clipping (b) After clipping

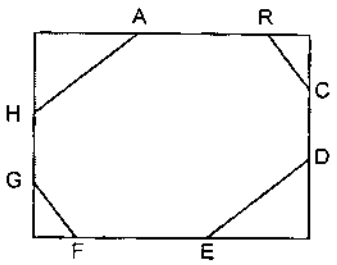
Fig. 3.15 Polygon clipping done by line clipping algorithm

3. We consider a polygon as a closed solid area. Hence after clipping it should remain close. To achieve this we require an algorithm that will generate additional line segment which make the polygon as a closed area.

For Example



(Before clipping)



(After clipping)

NOTES

In the above figure line A-B, C-D, E-F, and G-H are added to polygon description to make it closed.

3.19 SUTHERLAND HODGEMAN POLYGON CLIPPING

- A polygon can be clipped by processing its boundary as a whole against each window edge.
- This is achieved by processing all polygon vertices against each clip rectangle boundary in turn.
- Beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices.
- The new set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper as shown in the following figure :

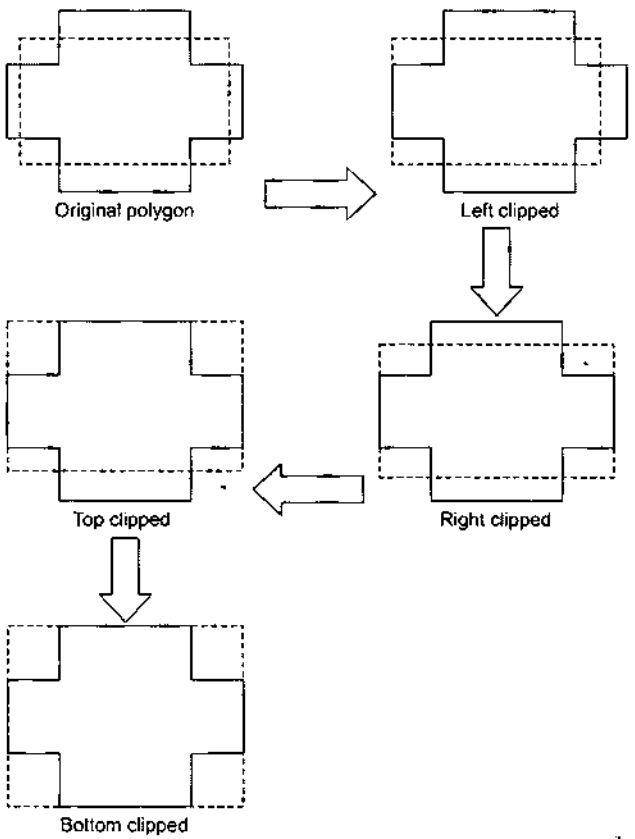


Fig. 3.16

NOTES

- At each step a new set of polygon vertices is generated and passed to the next window boundary clipper. This is the fundamental idea used in the Sutherland Hodgeman algorithm.
- The output of the algorithm is a list of Polygon vertices all of which are on the visible sides of a clipping plane. Such each edge of the polygon is individually compared with the clipping plane.
- This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane.

In this case we have to consider the following four points :

1. If the first vertex of the edge is outside the window boundary and the second vertex of the edge is inside then the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list.
2. If both vertices of the edge are inside the window boundary, then only the second vertex is added to the output vertex list.

   Graphically it is represented as follows :

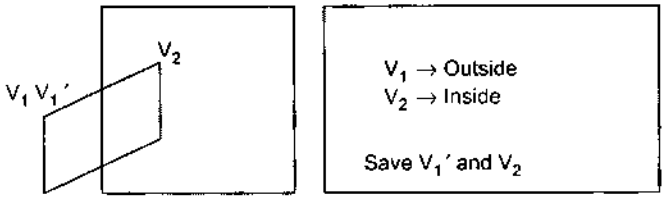


Fig. 3.17

3. If the first vertex of the edge is inside of the window boundary and the second vertex of the edge is outside then only the edge intersection with the window boundary is added to the output vertex list.

   Graphically it is represented as follows :

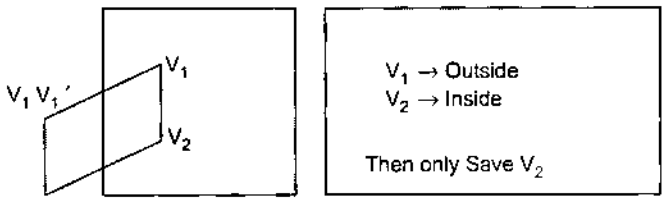


Fig. 3.18

4. If both vertices of the edge are outside of the window boundary, nothing is added to the output list.

   Graphically it is represented as follows :

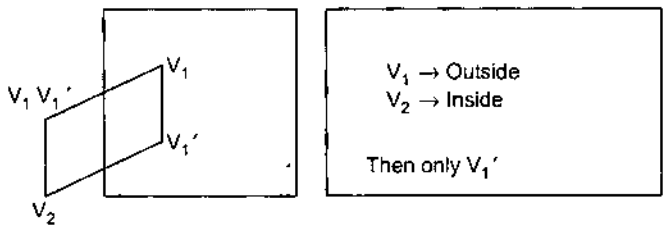


Fig. 3.19

Now from the above four points we can realize that there are two key processes in this algorithm

(*a*) Determining the visibility of a point or vertex (Inside - Outside Text).

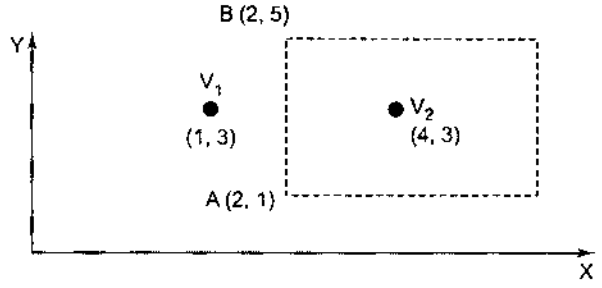
(*b*) Determining the intersection of the polygon edge and the clipping plane.

NOTE One way of determining the visibility of a point or vertex is described here.

- Consider that two points (A) and (B) define the window boundary and point under consideration is (V), then these three points define a plane.
- Two vectors which lie in that plane are AB and AV.
- If this plane is considered in the XY plane, then the vector cross product AV * AB has only a Z component, given by (XV – XA) (YB – YA) – (YB – YA) (XB – XA).
- The sign of the Z component decides the position of point (V) with respect to window boundary.
- If (Z) is positive, then the point is on the right side of the window boundary.
- If (Z) is negative, then the point is on the left side of the window boundary.

NOTES

Example. *Consider the clipping boundary as shown in the following figure and determine the position of points V_1 and V_2.*



Solution. Given

V_1 (1, 3)

V_2 (4, 3)

A (2, 1)

B (2, 5)

Using the cross product for (V1), we get

→ (XV – XA) (YB – YA) – (YB – YA) (XB – XA)

→ (1 – 2) (5 – 1) – (3 – 1) (2 – 2)

→ (–1) (4) – (2) (0)

→ –4 – 0

→ –4

The result of the cross product for (V_1) is negative hence (V_1) is on the left side of the window boundary.

Using the cross product for (V_2), we get

→ (XV – XA) (YB – YA) – (YB – YA) (XB – XA)

→ (4 – 2) (5 – 1) – (3 – 1) (2 – 2)

→ (2) (4) – (2) (0)

→ 8 – 0

→ 8

The result of the cross product for (V_2) is positive hence (V_2) is on the right side of the window boundary. **Ans.**

- The second key process is Sutherland Hodgeman Polygon Clipping algorithm is to determine the intersection of the polygon edge and the clipping plane.

Algorithm

1. Read coordinates of all vertices of the polygon.
2. Read coordinates of the clipping window.
3. Consider the left edge of the window.
4. Compare the vertices of each edge of the polygon, individually with the clipping plane.

5. Save the resulting intersection and vertices in the new list of vertices according to four possible relationships between the edge and clipping boundary as discussed earlier.
6. Repeat the step 4 and 5 for reversing edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

NOTES

3.20 GENERALIZED CLIPPING

We have seen that in Sutherland Hodgeman Polygon Clipping Algorithm, we need separate clipping routines, one for each boundary of the clipping window. But these routines are almost identical. They differ only in their test for determining whether a point is invisible or outside the boundary.

- It is possible to generalize these routines so that they will be exactly identical and information about the boundary is passed to the routine through their parameters.
- Using recursive technique, the generalized routine can be called for each boundary of the clipping window with a different boundary specified by its parameters.
- This form of algorithm allows us to have any number of boundaries to the clipping window, thus the generalized algorithm with recursive technique can be used to clip a polygon along an arbitrary convex clipping window.

3.21 MULTIPLE WINDOWING

Some systems allow the use of Multiple Windowing that is a first image is created by one or more window transformations on the object. Then windows are applied to this first image to create a second image. Further windowing transformation may be done until the desired picture is created.

Every application of a window transformation allows the user to slice up a portion of the picture and reposition it on the screen. Thus multiple windowing gives the user freedom to rearrange the components of a picture. The same effect may be obtaining by applying a number of single window transformations to the object.

3.22 SHIELDING

Shielding is the reverse of Clipping, *i.e.*, in Shielding, the portion which is inside of the window is not displayed only the portion which is outside of the window is displayed.

Example. *Use Sutherland Cohen Line Clipping method, to clip a line starting from (–13, 5) and ending at (17, 11) against the window having its lower left corner at (– 8, – 4) and upper right corner at (12, 8).*

Solution. Given

XW min = – 8
YW min = – 4
XW max = 12
YW max = 8

Given Line P (– 13, 5)
Q (17, 11)
Now we have to find the end points of P and Q.
P (–13, 5)

X = – 13 and Y = 5

Bit 1 → (Y – YW max) = (5 – 8) = – 3 < 0 code = 0

Bit 2 → (YW min – Y) = (– 4 – 5) = – 9 < 0 code = 0

Bit 3 → (X – XW max) = (–13 – 12) = – 27 < 0 code = 0

Bit 4 → (XW min – X) = (– 8 + 13) = 5 > 0 code = 1

So the end point code of point P is 0001.

Now the end point of Q

Q (17, 11)

X = 17 and Y = 11

Bit 1 → (Y – YW max) = (8 – 11) = – 3 < 0 code = 0

Bit 2 → (YW min – Y) = (11 + 4) = 15 > 0 code = 1

Bit 3 → (X – XW max) = (12 – 17) = – 5 < 0 code = 0

Bit 4 → (XW min – X) = (– 8 – 17) = – 25 < 0 code = 0

So the end point code of point Q is 0100.

Now the 4-bit code of X and Y are not 0000.

So the line PQ is not completely visible.

Now take the AND operation between the end point code of P and Q.

4 bit code of P → 0001

4 bit code of Q → 0100

AND operation → 0000

After the AND operation the result is 0000.

So the line is partially visible, *i.e.*, known as clipping coordinate. Now we have to find which portion is visible and which portion is clipped. For that particular purpose we have to use the following equation of line.

$$X = X_1 + (X_2 - X_1)\, t \quad \ldots [A]$$

$$Y = Y_1 + (Y_2 - Y_1)\, t \quad \ldots [B]$$

Now we have to find the (X, Y) coordinates for the intersecting points, *i.e.*, point (I_1) and point (I_2) as mentioned above.

NOTES

3.23 HARDWARE INPUT DEVICES HANDLING ALGORITHMS

Once the routine and their tasks for input device are specified, a user easily handles those devices.

It is not necessary for the user to know how these devices perform any function. The user can operate the device just by knowing the feature being available. The system acts as interface between user program and the device available.

There are different types of input devices. So the 'Insides' of the interface routines varies from device to device. Each device has its own algorithm however; the outsides will always look the same. The user comes in contact only with the outsides. A simulation of all graphics input is possible using only a keyboard device. A keyboard simulation needs drastic changes in the internal form of the routine of the general case. The reasons are stated below :

- The input applied from a keyboard via a high level language appears to be a sampled, rather than event driven. At any time, a READ may be done which always returns some value. So the processing is suspended until the input is obtained. Thus it acts a sampled device. That is when a READ occurs, a value is returned. To make it an event driven, the vale should be returned whenever

a READ occurs. independently of whether or not new information has been placed in a input buffer.

- Practically, this is not true for most high level languages.
- We can assume that there is only one device in use. Hence the procedure which occur become of two simulations events are not considered.

NOTES

3.24 CLASSES OF INPUT DEVICES

We have five classes of input devices, which are as follows :

1. **Button:** It is an event-driven device. When it is pressed an interrupt is generated.
2. **Pick:** This class used a light pen. It is again an event-driven device. It select a part of the display.
3. **Keyboard:** We all are familiar with this class. This is also an event-driven device which contained number of keys for applying input.
4. **Locator:** It is a sample device. It returns the coordinates of some position.
5. **Valuator:** It is also a sampled device. It returns only a single value. It may be set or reset by the user.

NOTE For specifying a particular piece of hardware, it is essential to indicate the class of input device and member of that class the device happen to be.

3.25 ENABLING AND DISABLING THE INPUT DEVICES

- Each device has ON-OFF facility to enable it when required in to operation and to disable after using it.
- This is achieved with the routines provided with each device.
- If any device is not required at any time, it is disabled which ignores its input, if any.
- The routines can be written to enable and disable the device classes instead of enabling and disabling individual device.
- In the routine, we can create a flag for each class.
- When a flag is set, it indicate, that particular class is enabled.

The following table describes the type of class and the equivalent class number for their type.

Type	Class Number
Button	1
Pick	2
Keyboard	3
Locator	4
Valuator	5

1. **Routine to Enable an Input Class:** The routine takes a class number; perform the necessary operations to turn the device-classes logically ON (enable) and sets the corresponding device flag to true.

   Algorithm: ENABLE_GRUP (CLASS) //Routine to enable an Input device class

   Argument: |CLASS| the code for the class to be enabled.

   Global: BUTTON, PICK, KEYBOARD, LOCATOR, VALUATOR device flags.

   Begin:

   If CLASS = 1 then

   Begin

PERFORM ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE BUTTON DEVICE;
BUTTON←TRUE;
End;
If CLASS = 2 then
Begin
PERFORM ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE PICK DEVICE;
PICK←TRUE;
End;
If CLASS = 3 then
Begin
PERFORM ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE KEYBOARD DEVICE;
KEYBOARD←TRUE;
End;
If CLASS = 4 then
Begin
PERFORM ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE LOCATOR DEVICE;
LOCATOR←TRUE;
End;
If CLASS = 5 then
Begin
PERFORM ALL OPERATIONS NEEDED TO PERMIT INPUT FROM THE VALUATOR DEVICE;
VALUATOR←TRUE;
End;
Return;
End;

NOTES

2. **Routine to Disable an Input Class:**

Algorithm: DISABLE_GRUP (CLASS) //Routine to disable an Input device class
Argument: [CLASS] the code for the class to be disabled.
Global: BUTTON, PICK, KEYBOARD, LOCATOR, VALUATOR device flags.
Begin:
If CLASS = 1 then
Begin
PERFORM ALL OPERATIONS NEEDED TO TURN OFF BUTTON DEVICE;
BUTTON←FALSE;
End;
If CLASS = 2 then
Begin
PERFORM ALL OPERATIONS NEEDED TO TURN OFF PICK DEVICE;
PICK←FALSE;

NOTES

```
        End;
        If CLASS = 3 then
        Begin
        PERFORM ALL OPERATIONS NEEDED TO TURN OFF KEYBOARD
DEVICE;
        KEYBOARD←FALSE;
        End;
        If CLASS = 4 then
        Begin
        PERFORM ALL OPERATIONS NEEDED TO TURN OFF LOCATOR
DEVICE;
        LOCATOR←FALSE;
        End;
        If CLASS = 5 then
        Begin
        PERFORM ALL OPERATIONS NEEDED TO TURN OFF VALUATOR
DEVICE;
        VALUATOR←FALSE;
        End;
        Return;
End;
```

NOTE We can also provide the user with a single routine for disabling all input devices.

3. **Routine to Disable all Input Devices:**

```
Algorithm: DISABLE_ALL  //Routine to disable all Input devices
Local: CLASS for stepping through the possible classes of devices.
Begin:
        For CLASS = 1 to 5
        Do
        Disable_Group(CLASS);
        Return;
End;
```

3.26 EVENT HANDLING

Generally we consider an event-driven device or one which generate an interrupt. When an interrupt occurs, the processor temporarily stops its current activity, services the interrupt and after completing the servicing, resumed its normal processing.

Servicing an Interrupt

- Servicing an interrupt means identifying which device caused the interrupt and obtaining the input data from that device. This information is then stored on the event queue.
- Strings are handled a little differently because they may require more storage. On the event queue, instead of trying to store the entire string in a data field, we store a pointer which tells us where the string may be found.

Algorithm (model) for the Processing of an Input Device Interrupt

NOTES

```
Algorithm EVENT (This algorithm is a model for the processing of an input device
interrupt)
BEGIN
    DISABLE THE PHYSICAL INTERRUPT.
    //To prevent interruption of the detected interrupt processing
    SAVE PROCESSOR STATUS;
    DETERMINE WHICH DEVICE (CLASS AND NUMBER) CAUSED THE INTER-
RUPT;

    IF DEVICE IS LOGICALLY ENABLED THEN
    BEGIN
        IF EVENT FROM A STRING INPUT DEVICE THEN
        BEGIN
            GET THE INPUT STRING;
            ADD_STRING_Q (STRING, DATA);
        END
        ELSE
            GET THE DATA FROM THE DEVICE;
            ADD_EVENT_Q (CLASS, NUMBER, DATA);
    END;
    RESTORE PROCESSOR STATUS;
    REENABLE PHISICAL INTERRUPT;
    RETURN;
END;
```

The queue data structure works on a principle, First-In-First-Out (FIFO). When the new data arrives, it is placed at the rear-end of the queue. The following algorithm can be used to add a new entry to the queue.

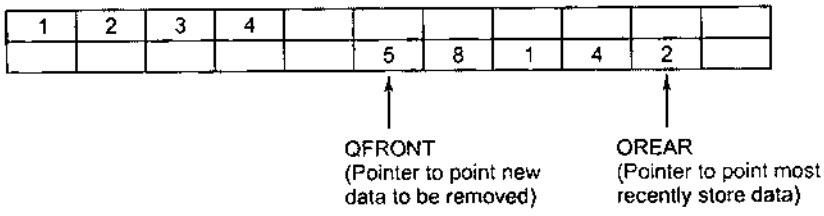
```
Algorithm ADD_EVENT_Q (CLASS, NUMBER, DATA)
    //Add the event to the event queue
Arguments  CLASS     Class of the input device.
           NUMBER    Number of the input device.
           DATA      Data from the input device.
GLOBAL     EVENTQC, EVENTQN, EVENTQD
           CLASS ARRAY, NUMBER ARRAY AND DATA ARRAY OF SIZE QSIZE,
           Which from the event queue.
           QFRONT, QREAR are the points to point front and rear ends of event queue.
CONSTANT-QMAX the Maximum size of the Event Queue
BEGIN
           If QREAR = QMAX then QREAR←1 else
               QREAR←QREAR + 1;
           if QFRONT=QREAR then RETURN ERROR 'EVENT OVERFLOW'
           EVENTQC[QREAR]←CLASS;
           EVENTQN[QREAR]←NUMBER;
           EVENTQD[QREAR]←DATA;
           If QFRONT = 0 then QFRONT←1
           RETURN;
```

NOTES

END;

- Here, we have used arrays to hold the queue data. The location of the last entry in the queue is indicated by a pointer. Here the pointer QREAR performs this feature. When the next event occurs, the new data is stored at the location pointed by incrementing the pointer.
- If the pointer reached at the end of the array, that is, if the queue is completely filled, the pointer points to the first array pointer.
- There is one more pointer, QFRONT to point the position (location) from where the next data is to be removed from the queue. If the QFRONT is equal to zero, it indicates that the queue is empty. When the first data is entered in the queue, it is set to one.

That means we can say that the above procedure is implemented with the help of circular queue. The following figure shows the various pointers of a queue.



3.27 STRING QUEUE

This is another way of storing information. The information may be stored in a string and the string may be stored in a special storage area, called String-Queue.

The following algorithm is written to add a string to a String-Queue. Here each new string is added to an array of strings. We also implement this algorithm with the help of circular queue.

Algorithm to add a string to a String-Queue

```
ALGORITHM   ADD_STRING_Q(STRING, DATA)
//Saves string and returns a pointer to it in DATA
ARGUMENTS   STRING a string to be saved in the string array.
            DATA for return of the index of the stored string
GLOBAL      STRINGQ an array of strings.
            SQREAR next free string storage area.
CONSTANT    SQMAX the size of the string queue array.
BEGIN
        If SQREAR = AQMAX then SQREAR←1 else SQREAR←SQREAR+1;
        STRINGQ[SQRAR]←STRING
        DATA←SQREAR;
        Return;
        END;
```

3.28 EVENT CHECKING

The event queue can be checked to determine if an

- Event has taken place.
- The routine can be written which returns information about which device has caused an event.

- After checking the queue, if we find that the queue is empty, it indicates no event has taken place. The routine may poll the queue for a specified period of time.
- At the end of this time, if the queue is still empty, an indication of failure may be returned.
- The routine is given below. It considered two type of devices
  (*a*) Interrupt Generation
  (*b*) Non Interrupt Generated.

The two cases are handled differently as discussed as follows :

NOTES

Case 1. Non Interrupt Generated Devices: If the input device not generates interrupts, then the event process should be simulated through polling the device directly. The Polling may be done by two ways

1. It may be built in to the primary language and/or operating system used. In this case, the input will acts as it from a sampled device.
   For Example
   READ statement of a high level language.
   In this routine given below, keyboard input is obtained using a simple READ statement.
2. The second way uses a loop written by us. This is a polled device. In the routine given below, the buttons are detected with polling loop.

Case 2. Interrupt Generated Devices: In this case, the routine polls the queue until the event has occurred. Then it obtain from queue, the class and number of the device which caused the event. In the routine queue below picks are done through true interrupts.

Algorithm to Check the Event Queue

```
Algorithm:    AWAIT_EVENT (WAIT, CLASS, DEVICE)
              //Routine to check the event queue
Arguments     WAIT the time to wait for an event to occur.
              CLASS, DEVICE to return the type of event which occurred.
Global        BUTTON, PICK, KEYBOARD device enabled flags.
              INPUT_STRING, PICKED_STORE storage for keyboard and pick in-
              put DETECTABLE segment detect ability] attribute array.
              BUTTON_FLAG, PICK_FLAG, KEYBOARD_FLAG flags indicating
              the status of polled devices.
Local         TIME_END the time at which to stop waiting.
              DATA for receiving data from the event queue.
BEGIN
    If buttons are simulated by sampled device, then include the following
    conditional statements
    If BUTTON then
    BEGIN
        READ DEVICE;
        CLASS←1;
        Return;
    END;
    If picks are simulated by sampled device, then include the following
    conditional statements
    If PICK then
    BEGIN
```

NOTES

```
PICKD_STORE←0;
While PICKE_STORE<1 OR PICKED_STORE>NO. OF SEGMENTS
OR NOTETECTABLE (PICKED_STORE) do
READ PICKED_STORE;
        CLASS←2;
        DEVICE←1;
        Return;
END;
```

If keyboard is simulated by sampled device, then include the following conditional statements

```
If KEYBOARD then
BEGIN
READ INPUT_STRING;
        CLASS←3;
        DEVICE←1;
        Return;
END;
```

If interrupt generating or polled devices are available then include the following loop.

```
TIME_END←TIME () + WAIT;
While TIME () <= TIME_END o
BEGIN
        GETQ (CLASS, DEVICE, DATA);
        If CLASS != 0 then
        If CLASS = 2 then PICKED_STORE←DATA;
        Else if CLASS = 3 then INPUT_STRING←STRINGQ [DATA];
        Return;
END;
```

If a pick is simulated on a polled device then include the following conditional statements

```
BEGIN
        READ_PICK (DEVICE, PICKED_STORE);
        If DETECTABLE [PICKED_STORE] then
        BEGIN
                CLASS←2;
                Return;
        END;
END;
```

If the keyboard is treated as a polled device, then include the following conditional statements

```
If KEYBOARD and KEYBOARD_FLAG then
BEGIN
        CLASS←3;
        READ_KEYBOARD (DEVICE, INPUTSTRING);
        Return;
END;
```

```
CLASS←0;
DEVICE←0;
Return;
END;
```

Getting the Event (Class, Device and Data)

- The algorithm given below returns the event (class, device, and data) at the front of the event queue. The position of the loading elements is indicated by the pointer QFRONT.
- If QFRONT is zero. It indicates that the queue is empty, otherwise, the value of the loading item is returned and QFRONT is incremented to point the next entry.

NOTES

Algorithm which returns the event at the front of the event queue

Algorithm GETS (CLASS, DEVICE, and DATA) returns the event of the front of the event queue. If the queue is empty, zero is returned.

```
Arguments    CLASS, class of the event
             DEVICE, device of the event
             DATA, input data from the event.
Global       EVENTQC, EVENTQN, EVENTQD the event queue arrays,
             QFRONT, QREAR pointers to front and rear of event queue.
Constants    QMAX, the size of the event queue.
BEGIN
     CLASS←0;
     If QFRONT = 0 then return;
     CLASS←EVENTQC [FRONT];
     DEVICE←EVENTQN [FRONT];
     DATA←EVENTQD [FRONT];
     If QFRONT = QREAR then
     BEGIN
          QFRONT←0;
          QREAR←0;
     END;
     Else if QFRONT == QMAX then QFRONT←1;
     Else QFRONT←QFRONT + 1;
     Return;
END;
```

Flushing all events from the EVENT QUEUE

- The event queue stores the event information. We can obtain this information whenever for future processing.
- Sometimes, we may want to start fresh, that is, a fully clear queue. In this case, it is necessary to flush or clear the event queue.

The following algorithm discarded the unwanted events from the event queue

```
Algorithm    FLUSH_ALL_EVENTS
             // Removes all events from event queue
Global       QFRONT, QREAR, event queue front and rear pointers.
```

SQREAR String file pointer.

```
BEGIN
        QFRONT←0;
        QREAR←0;
        QSREAR←1;
        Return;
END;
```

NOTES

Getting Device Data

We know that, how to store the information, when an event occurred. This information can be retrieved using, routine. The following routine is written to returns stored keyboard input :

Algorithm to write the stored keyboard input:

```
Algorithm     GET_KEYBOARD_DATA (STRING, LEN)
              // Routine to return the stored keyboard input.
Arguments     STRING, for the return of the string.
              LEN, the string's length.
Global        INPUT_STRING, keyboard input storage.
BEGIN
        LEN←LENGTH (INPUT_STRING);
        STRING←INPUT_STRING;
        Return;
END;
```

The following algorithm is written to return the selected pick value.

Algorithm to return the selected pick value:

```
Algorithm     GET_PICK_DATA (SEGMENT_NAME)
              // Routine to return the selected pick value.
Arguments     SEGMENT_NAME, the name of the selected segment.
Global        PICKED_STORE, Pick input storage.
BEGIN
        SEGMENT_NAME←PICKED_STORE;
        Return;
END;
```

AWAIT INPUT from Individual Devices

- We have studies AWAIT_EVENT routine. That routine is written by assuming several input devices in use. However, many applications require input from only one device at a time and prohibit the simultaneous use of several devices.

For example Time-Sharing system.

In such situations, we must use routines which await input from only a single class of devices. Such routines are device and system dependent.

Algorithm to await input from a BUTTON

```
Algorithm     AWAIT_BUTTON (WAIT, BUTTON_NUM)
              // User routine to await the pressing of a button.
Arguments     WAIT, the time of wait for a button event.
              BUTTON_NUM, for return the number of button device.
Global        BUTTON, device enable flag.
```

```
                BUTTON_FLAG, status flag if button is a polled device.
Local           TIME_END, the time at which to stop waiting.
                DUMMY, a dummy argument.
BEGIN
        If Buttons are simulated by a sampled device, then include the following
        segments-
        READ BUTTON_NUM;
        If interrupt generating or polled device are used then include the following
        loop
        TIME_LIMIT←TIME ( ) + WAIT;
        While TIME ( ) <= TIME_LIMIT do
        BEGIN
                If interrupt generating buttons are used, they may be formed by
                BEGIN
                        GETQ (CLASS, BUTTON_NUM, DUMMY);
                        If CLASS==1 then return;
                END;
                If buttons are simulated on a polled device, then include the
                Following
                BEGIN
                        READ_BUTTON (BUTTON_NUM);
                        Return;
                END;
        END;
                BUTTON_NUM←0;
                Return;
END;
```

Algorithm to await a pick:

```
Algorithm       AWAIT_PICK (WAIT, PICK_NUM);
                // User routine to await a pick.
Arguments       WAIT, the time to wait for a pick event.
                PICK_NUM, for return of the no. of the picked events.
Global          PICK, device enabled flag.
                PICKS_FLAG, status flag if pick is a polled device.
Local           TIME_END, the time at which to stoop waiting.
BEGIN
        If NOT PICK the return ERROR 'PICK NOT FOUND';
        If picks are simulated by a sampled device, then include the following-
        READ PICK_NUM;
        If interrupt generating or polled devices are used, then include the
        Following loop
        TIME_END←TIME ( ) + WAIT;
        While TIME( )<=TIME_END do
        BEGIN
                If interrupt generating picks are used, they may be formed by
                BEGIN
                        GETQ (CLASS, DUMMY, PICK_NUM0;
```

NOTES

```
                If (CLASS == 2) then return;
            END;
            If picks are simulated on a polled device then include the
            Following conditional statements-
            If PICK_FLAG then
            BEGIN
                ____________
                return;
            END;
        END;
        PICK_NUM←0;
        Return;
END;
```

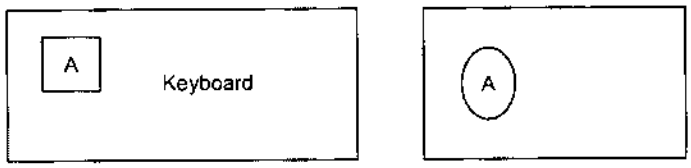
NOTES

3.29 ECHOING

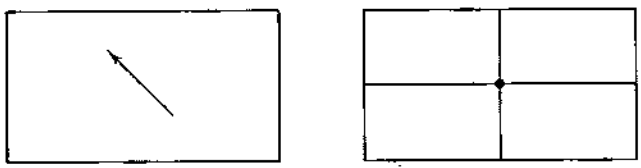
- In graphics, user can obtain the information about his/her actions with the technique called Echoing.
- With this information, user can compare what he has done with what he wants to do.
- Some form of Echoing should be present which allows user to work comfortable and confidently with the program.
- So Echoing is an important part of an interactive system.

Examples of Echoing

Example. *For keyboard input, the typed character is usually displayed on the screen. Graphically it is represented as follows :*

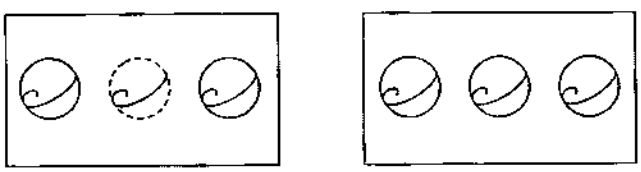


Example. *A screen cursor is used for echoing a location pointed by a locator. It displays the current locator position. So the user can see the current locator setting and rotate its position to the objects on the displays. Graphically it is represented as follows :*



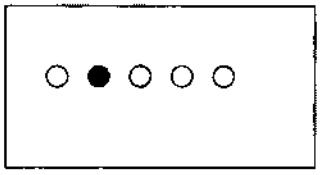
Echoing a location

Example. *A pick may be echoed by identifying the selected objects on the display. The selected objects can be differentiated by making them brighter, dotted, flashing them or changing their cooler. Graphically it is represented as follows :*



Echoing a pick

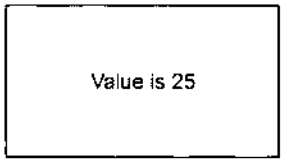
Example. *Buttons can be used to select menu items. They can be echoed by differentiating them from non-selected items by making them brighter, dotted, flashing or changing their color. Graphically it is represented as follows :*

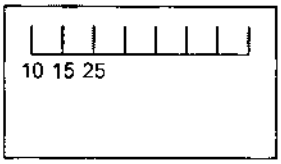


	ITEM 1
	ITEM 2
	ITEM 3
	ITEM 4
	ITEM 5

NOTES

Example. *For a valuator, echoing is possible by displaying the current setting in numerical form or as a position on a scale as shown below. Graphically it is represented as follows:*





3.30 INTERACTIVE TECHNIQUES

In graphics, Interactive Techniques are used for interlacing, creating and modifying pictures. Some of these techniques are as follows :

Point Plotting

- It allows user to select a particular point,on the screen.
- It can be performed by a combination of a locator and a button.

The process of point plotting is as follows :

The user selected point is located by locator and the button indicates when the locator is currently positioned. The Algorithm for point plotting is given below. It must await the button event. As soon as it occurs, the locator may be read. The following algorithm selects a point on a screen.

Algorithm to select a point:

```
Algorithm     AWAIT_BUTTON_GET_LOCATOR (WAIT, BUTTON_NUM, X, Y)
              // User routine to interactively select a point.
Arguments     WAIT, the time to wait for a button event.
              BUTTON_NUM, for return of the user's button selection.
              X, Y points the user selected.
BEGIN
        AWAIT_BUTTON (WAIT, BUTTON_NUM);
        READ_LOCATOR (X, Y);
        Return;
END;
```

NOTE The point plotting routine can be use to obtain line segments. In this case, the user can select the points and those points can be connected with a line segment.

```
Algorithm     DRAWLINE ( )
BEGIN
        BUTTON_NUM←continue;
        While BUTTON_NUM←CONTINUE;
        Do
        BEGIN
              AWAIT_BUTTON_GET_LOCATOR (WAIT, BUTTON_NUM, X, Y);
```

```
                LINE_ABS_2 (X, Y);
                MAKE_PICTURE_CURRENT;
            END;
        END;
```

Graphically it is represented as follows :

NOTES

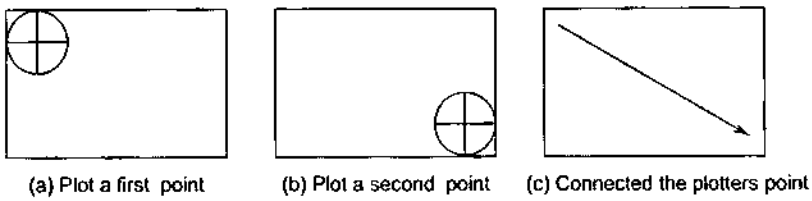


Fig. 3.20

There are various interactive techniques, which are as follows :

(*a*) Positioning Techniques

(*b*) Pointing and Selection

(*c*) Inking and Pointing.

(*a*) **Positioning Techniques:**

- Positioning sometimes knows as locating which is one of the most basic graphical input techniques.
- In this technique. the user indicates a position on the screen with an input device and this position is used to insert a symbolic or to define the end points of a line. Positioning operations can benefit in a number of ways from the use of feedback.
- There are various examples of positioning feedback. which are as follows :
  - Rubber Band Technique
  - Dragging
  - Dimensioning Techniques
  - Graphical Potentiometer.

(*b*) **Pointing and Selection:** Graphical input devices play a very important role in allowing the user to point to the screen.

- In many applications pointing rather than positioning is the basis for interaction.
- The user may have no need to add more information to the picture and may be interested solely in studying and asking questions about the information already displayed.

(*c*) **Inking and Pointing:** If we sample the position of a graphical input device at regular intervals and display a dot at each sampled position, a trail will be displayed of the movement of the device. This technique which closely simulates the effects of drawing on paper is called Inking. Graphically it is represented as follows :

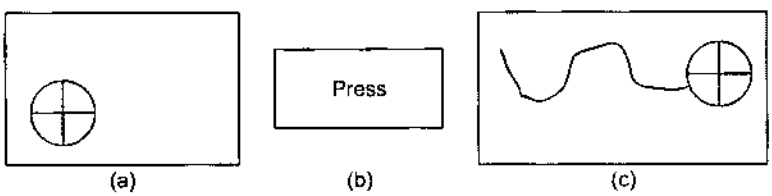


Fig. 3.21 Feedback inking

SUMMARY

NOTES

- The manipulation of image is done by performing appropriate geometric or coordinate transformation on the object.
- There are some basic transformation such as Translation, Scaling, and Rotation.
- Other than these transformations Reflection and Shearing are the other transformation.
- Translation is used to shift the object from one position to another position.
- Scaling is used to change the shape of the picture, *i.e.*, reduce or magnify the shape of the object.
- Rotation is used to rotate the object about a given point.
- Reflection gives the mirror image of the object.
- Shearing is the distortion in the shape of the object.
- Homogeneous coordinate system is used because due to this it is possible to use the combine Translation, Rotation, Scaling which increase the efficiency and elegance.
- A rectangular area specified in word coordinates is called Window, *i.e.*, the portion of the picture which we want to display is known as Window.
- A rectangular area specified in device coordinates is called View Port, *i.e.*, the portion of the screen, where we want to map the window is known as View Port.
- The portion which is inside the window is visible and the portion, which is outside of the window, is not visible. This process is known as clipping.
- Shielding is the opposite of clipping, *i.e.*, the portion, which is inside of the window, is not visible and the portion, which is outside of the window is visible.
- The viewing transformation can be performed by using three steps :
  (a) Translation
  (b) Scaling
  (c) Re-Translation.
- Cohen Sutherland algorithm and Midpoint Subdivision algorithm are based on 4 bit codes that are known as region codes. They are used for the clipping.
- The Sutherland Hodgeman algorithm is used for polygon clipping.
- We have the various Hardware Input Interaction techniques by which we can direct interact with the Hardware.
- We divide the Input devices in the form of classes.
- The classes are as follows:
  (a) BUTTON (b) PICK
  (c) KEYBOARD (d) LOCATOR
  (e) VALUATOR.
- The classes also have the unique numbers which are as follows :

TYPE	CLASS NUMBER
BUTTON	1
PICK	2
KEYBOARD	3
LOCATOR	4
VALUATOR	5

[INPUT DEVICE Type and Class Number]

- We can also enable and disable the complete group class according to our requirement.
- Event Queue is a circular queue, which holds the Number Queue, Data Queue and Class Queue.
- Event Queue is the collection of Number Queue, Data Queue and Class Queue.

NOTES

REVIEW QUESTIONS

1. Give general form for the rotation about any arbitrary point P(*h*, *k*).
2. Give the 2-D transformation matrix for
   (*a*) Translation
   (*b*) Scaling
   (*c*) Rotation.
3. Why the concept of Homogeneous Coordinate system is evolved? Give the Homogeneous Coordinate for Translation, Scaling, and Rotation.
4. What do you mean by composite transformation? How is it useful?
5. Derive the transformation matrix for rotation about an arbitrary point P(*h*, *k*).
6. Write a short note on :
   (*a*) Mirror reflection
   (*b*) Shearing transformation.
7. Show how reflection in the line $y = x$ and in the line $y = -x$ can be performed by a scaling operation followed by a rotation.
8. Find out the sequence of basic transformations which are equivalent to X-direction shearing.
9. Find out the sequence of basic transformations which are equivalent to Y-direction shearing.
10. Show that the successive reflection about any line passing through the coordinate origin is equivalent to a single rotation about the origin.
11. What is shearing transformation? Explain with suitable examples.
12. Prove that one scaling transformation and one rotation in 2-D transformation are commute, *i.e.*,
$$S * R = R * S$$
13. Prove that two 2-D transformation are commute, *i.e.*,
$$T_1 * T_2 = T_2 * T_1$$
14. What is composite transformation? Explain with suitable examples.
15. Prove that
$$R_{(\theta)}^{-1} = R_{(-\theta)}.$$
16. Define the term windowing, also define the importance of windowing.
17. Describe the method by which any point can be determine that it is left or right to the any line segment.
18. Find the Normalization transformation that maps a window whose lower left corner is at (0, 0) and the upper right corner is at (2, 4) on to
   (*a*) A view port that is the entire normalized device screen.
   (*b*) A view port that has lower left corner at (– 1, – 1) and the upper right corner at (– 1/2, – 1/2).
19. Differentiate between Window and a View Port with suitable examples.
20. Write short notes on the following:
   (*a*) Viewing Transformation
   (*b*) Clipping
   (*c*) Windowing
   (*d*) The Cohen Sutherland Line Clipping Algorithm
   (*e*) The MidPoint Subdivision Algorithm.
21. Show why the Sutherland Hodgeman clipping algorithm will only work for convex clipping regions.
22. Write a routine to clip an ellipse against a rectangular Window.

23. Explain Sutherland Hodgeman algorithm for polygon clipping.
24. Define the term point clipping and line clipping.
25. Define the term viewing transformation with suitable example.
26. Derive the transformation matrix for 2D viewing transformation.
27. Explain the advantages and disadvantages of Sutherland Hodgeman Polygon Clipping algorithm.
28. Find a normalized transformation from the window whose lower left corner is at (0, 0) and upper right corner is at (*m*, *n*) on to the normalized device screen so that aspect ratio remains same.
29. Let a line segment with end points L_1 (– 2, 1) and L_2 (2, 8), determine point P (1, – 1) is left or right to the line segment?
30. What do you mean by viewing transformation? Define it with suitable example.
31. Write routines to perform the following operations on input devices, button, pick, keyboard, locator, and valuator by considering each of them as a class.
    (*a*) Enable input device class individually
    (*b*) Disable input device class individually
    (*c*) Disable all input devices.
32. What is the function of event queue? Give the algorithm to add an event to the event queue.
33. Give algorithm which returns event (Class, Device and Data) from the event queue.
34. Give algorithms to await input from a button and a pick.
35. Define the term polling with suitable examples.
36. Explain the concept of Interrupt with suitable examples.
37. Write short note on the following:
    (*a*) Event Queue
    (*b*) On Line Character Recognition
    (*c*) Positioning Techniques
    (*d*) Inking and Pointing.
38. Explain the concept of event handling in detail.
39. Explain Interrupt scheme for retrieving input data. Also explain any one algorithm for input device handling.
40. What do you understand by Echoing? How does echoing allow the user to compare what he has done against what he wants to do?
41. What is Point Polling? Give algorithm to select a point.
42. Write an algorithm to draw a line segment.
43. What do you mean by "rubber band lines"? How these lines can be implemented?
44. Define the term Dragging. Write a routine for dragging.
45. Write an algorithm for the processing of an input device interrupt.

NOTES

FURTHER READINGS

- **Computer Graphic**: V.K. Pachghare, Laxmi Publications, 2007, Second edition.
- **Computer Graphics**: Prabhakar Gupta, Vineet Agarwal and Manish Varshney, Laxmi Publications, 2011.
- **Computer Graphics**: Rajiv Chopra, S. Chand Publisher, 2011.
- **Computer Graphics**: C.S. Verma, Ane Books, 2011.
- **Computer Graphics**: Pradeep K. Bhatia, I.K. International, 2009, pbk, Second Edition.
- **Computer Graphics**: Ruchi Mishra, Global Vision Publisher, 2010.

UNIT IV

NOTES

3-D TRANSFORMATION

STRUCTURE

4.0 LEARNING OBJECTIVES

After going through this unit, you should be able to:

- describe three dimensional transformation
- explain three dimensional viewing
- discuss projection and vanishing point.

4.1 INTRODUCTION

Some graphics applications are two dimensional such as charts and certain maps and so on. However to create a realistic picture, scene or model, we need to represent it in 3-D graphics.

The creation of realistic picture is an important task in various fields such as simulation, design, entertainment, advertising, research, education etc.

To create a realistic picture we must process the scene or picture through viewing-coordinate transformation and projection routines that transforms three dimensional viewing coordinate or to two-dimensional device coordinates.

4.2 3-D GEOMETRY

The three-dimensional system has three axes *x*, *y* and *z*. The orientation of coordinate system is determined by two systems.

(*a*) Right Handed Coordinate System

(*b*) Left Handed Coordinate System.

NOTES

(*a*) **Right Handed Coordinate System:** In the right handed system the thumb of the right hand points in the position *z* direction as the curls the fingers of the right hand from *x* to *y*.

Graphically, the Right Handed Coordinate System is represented as follows :

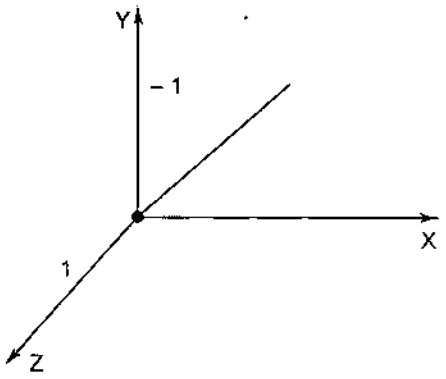


Fig. 4.1(a)

(*b*) **Left Handed Coordinate System:** In the left handed coordinate system , the left hand thumb to point the position *z* direction when we imagine the fingers of the left hand curl from the position *x*-Axis to the position *y*-Axis(through 90°) to grasp the *z*-Axis.

Graphically, the Left Handed Coordinate System is represented as follows :

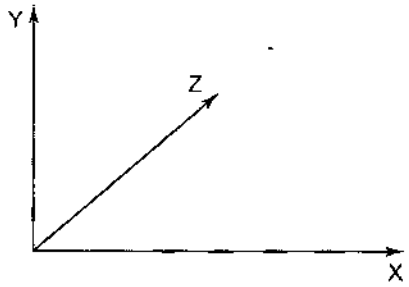


Fig. 4.1(b)

Here we adopt right handed system for computer graphics.

Now with this coordinate system we can specify any point in space by the order triple(*x*, *y*, *z*).

In 3-D geometry, the line is specified by a pair of equations

$$\left(\frac{Y - Y_1}{X - X_1}\right) = \left(\frac{Y_2 - Y_1}{X_2 - X_1}\right)$$

and

$$\left(\frac{Z - Z_1}{X - X_1}\right) = \left(\frac{Z_2 - Z_1}{X_2 - X_1}\right)$$

Where (x_1, y_1, z_1) and (x_2, y_2, z_2) are the two points which specify the line.

A Plane is specified by a single equation of the function

$$Ax + By + Cz + D = 0$$

4.3 3-D PRIMITIVES

Like 2-D primitives, we have 3-D primitives to draw points, lines, and plane in three dimensions. Here we have to provide the three coordinate specifications instead of two.

Let us consider three-dimensional LINE and MOVE algorithms

NOTES

Algorithm-1: (3-D Absolute Move)

```
MOVE_ABS_3D (X, Y, Z)
  Arguments- X, Y, Z are three coordinates of point to move the pen to.
  Global- DF_CUR_X, DF_CUR_Y, DF_CUR-Z Current pen position coordinates.
  Begin
        DF_CUR_X←X;
        DF_CUR_Y←Y;
        DF_CUR_Z←Z;
        DISPLAY_FILE_ENTER(1);
        Return;
  End;
```

Algorithm-2: (3-D Absolute Line)

```
LINE_ABS_3D (X, Y, Z)
  Arguments- X, Y, Z are three coordinates of point to move the line to.
  Global- DF_PEN_X, DF_ PEN _Y, DF_ PEN -Z the Current pen position.
  Begin
        DF_ PEN _X←X;
        DF_ PEN _Y←Y;
        DF_ PEN _Z←Z;
        DISPLAY_FILE_ENTER (2);
        Return;
  End;
```

Algorithm-3: (3-D Relational Move)

```
MOVE_REL_3D (DX, DY, DZ) [The 3-D relation Move]
  Arguments- DX, DY, DZ changes to be made to the pen position.
  Global- DF_PEN_X, DF_ PEN _Y, DF_ PEN -Z the Current pen position.
  Begin
        DF_ PEN _X←DF_PEN_X+DX;
        DF_ PEN _Y←DF_PEN_Y+DY;
        DF_ PEN _Z←DF_PEN_Z+DZ;
        DISPLAY_FILE_ENTER (1);
        Return;
  End;
```

Algorithm-4: (3-D Relational Line)

```
LINE_REL_3D (DX, DY, DZ) [The 3-D relation Line drawing Routine]
  Arguments- DX, DY, DZ displacement over which a line is to be drawn.
  Global- DF_PEN_X, DF_ PEN _Y, DF_ PEN -Z the Current pen position.
```

```
Begin
        DF_PEN _X←DF_PEN_X+DX;
        DF_PEN _Y←DF_PEN_Y+DY;
        DF_PEN _Z←DF_PEN_Z+DZ;
        DISPLAY_FILE_ENTER (2);
        Return;
End;
```

NOTES

Algorithm-5: Absolute Polygon

```
POLYGON_ABS_3 (AX, AY, AZ, N) [The 3-D absolute Polygon drawing Routine]
  Arguments- [N] The no. of Polygon sides.
  AX, AY, AZ arrays of the coordinates of the vertices.
  Global- DF_PEN_X, DF_ PEN _Y, DF_ PEN -Z the Current pen position.
  Local- I for looping.
  Begin
  If N<3 then return error 'size error';
        DF_ PEN _X←AX [N];
        DF_ PEN _Y←AY [N];
        DF_ PEN _Z←AZ [N];
        DISPLAY_FILE_ENTER (N);
  For I=1 to N
  Do LINE_ABS_3 (AX[I], AY[I], AZ[I]);
  Return;
  End;
```

Algorithm-6: (Relational Polygon)

```
POLYGON_REL_3 (AX, AY, AZ, N) [The 3D relative Polygon drawing Routine]
  Arguments- [N] The no. of Polygon sides
  AX, AY, AZ arrays of the displacement for the Polygon sides.
  Global- DF_PEN_X, DF_ PEN _Y, DF_ PEN -Z the Current pen position.
  Local- I for looping, TMPX, TMPY, TMPZ storage of points at which Polygon is closed.
  Begin
  If N<3 then return error 'size error';
        // Move for starting vertex
        DF_ PEN _X← DF_ PEN _X +AX [I];
        DF_ PEN _Y← DF_ PEN _Y +AY [I];
        DF_ PEN _Z← DF_ PEN _Z+ AZ [I];

        //Save vertex for displaying the Polygon
        TEMPX←DF_PEN_X;
        TEMPY←DF_PEN_Y;
        TEMPZ←DF_PEN_Z;

        DISPLAY_FILE_ENTER (N);
  //Enter the Polygon Sides
  For I=2 to N
```

```
Do LINE_REL_3 (AX[I], AY[I], AZ[I]);

//Close the Polygon
LINE_ABS_3 (TEMPX, TEMPY, TEMPZ);
Return;
End;
```

NOTES

4.4 THREE-DIMENSIONAL TRANSFORMATION

Like two dimensional transformation, three-dimensional transformations are formed by composing the basic transformations of Translation, Scaling and Rotation. Each of these transformations can be represented as a Matrix transformation with homogenous coordinates. Therefore any sequence of transformations can be represented as a single matrix, formed by combining the matrices for the individual transformations in the sequence.

1. **Translation:** Initially Pont P(X, Y, Z) is in the 3 coordinate system and the translation factor T(*tx*, *ty*, *tz*) are also given then after the translation the Point P′(X′, Y′, Z′) is as follows

$$X' = X + tx \quad \text{... [A]}$$
$$Y' = Y + ty \quad \text{... [B]}$$
$$Z' = Z + tz \quad \text{... [C]}$$

Then we can represent the translation in the Homogenous coordinate system as follows

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = T * P;$$

Where $$T = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is known as translation matrix in 3-D coordinate system.

2. **Scaling:** If we want to change the size of the picture in 3-D coordinate system then it is called as Scaling in 3-D coordinate system. Initially the point P(X, Y, Z) and the scaling factor (Sx, Sy, Sz) then after the scaling the point P′(X′, Y′, Z′) is as follows :

$$X' = X + Sx \quad \text{... [A]}$$
$$Y' = Y + Sy \quad \text{... [B]}$$
$$Z' = Z + Sz \quad \text{... [C]}$$

We can represent the scaling in the form of matrix as follows :

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = S_{(Sx, Sy)} * P$$

Where $S_{(Sx, Sy)} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

is called Scaling Matrix in 3-dimensional coordinate system.

3. **Rotation:** In 3-D coordinate system, Rotation can be performed in 3 ways

(*a*) **Rotation About Z-Axis:** If we want to rotate the image (point) about Z-Axis then the equation is as follows

$$\left.\begin{aligned} X' &= X\cos\theta - Y\sin\theta \\ Y' &= X\sin\theta + Y\cos\theta \\ Z' &= Z \end{aligned}\right] \quad \ldots [A]$$

Now we can represent this in the form of matrix as follows

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = Rz(\theta) * P$$

Where

$$Rz(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is called Rotation Matrix about Z-Axis.

(*b*) **Rotation About X-Axis:** If we change in equation no. (*a*) $X \rightarrow Y$, $Y \rightarrow Z$ and $Z \rightarrow X$ then the following equation may occur

$$\left.\begin{aligned} Y' &= Y\cos\theta - Z\sin\theta \\ Z' &= Y\sin\theta + Z\cos\theta \\ X' &= X \end{aligned}\right] \quad \ldots [B]$$

Now we can represent this in the form of matrix as follows

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = Rx(\theta) * P$$

Where $Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

is called Rotation Matrix about X-Axis.

(*c*) **Rotation About Y-Axis:** If we change in equation no. (*b*) $X \rightarrow Y$, $Y \rightarrow Z$ and $Z \rightarrow X$ then the following equation may occur

$$\left.\begin{aligned} Z' &= Z\cos\theta - X\sin\theta \\ X' &= Z\sin\theta + X\cos\theta \\ Y' &= Y \end{aligned}\right] \quad \ldots [C]$$

Now we can represent this in the form of matrix as follows

NOTES

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = Ry(\theta) * P$$

Where

NOTES

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Is called Rotation Matrix about Y-Axis.

Example. *Given point P(0,4,0,1) Rotate it by 90° anticlockwise about X-Axis.*

Solution.

$$P' = Rx_{(90°)} * P$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 90 & -\sin 90 & 0 \\ 0 & \sin 90 & \cos 90 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 4 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 4 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix}$$

So in this case $P' = (0,0,4,1)$ **Ans.**

Now we want to Rotate P′ by 90° about Y-Axis.

$$P'' = Ry_{(90°)} * P'$$

$$P'' = \begin{bmatrix} \cos 90 & 0 & \sin 90 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 90 & 0 & \cos 90 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X'' \\ Y'' \\ Z'' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X'' \\ Y'' \\ Z'' \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

So in this case $P'' = (4,0,0,1)$ **Ans.**

4.5 TILTING

NOTES

Tilting is equivalent to rotation about one axis by some angle θ followed by rotation about some other axis by ϕ.

For example

$$P' = Rx_{(\theta)} * P \quad \text{... [A]}$$

$$P'' = Ry_{(\phi)} * P' \quad \text{... [B]}$$

From equation [A] and [B] we have

$$P'' = Ry_{(\phi)} * Rx_{(\theta)} * P$$

But if we consider this

$$P' = Rx_{(\theta_1)} * P$$

$$P'' = Rx_{(\theta_1)} * P'$$

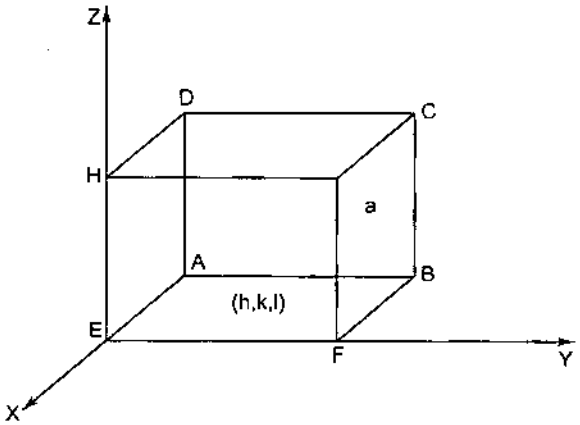
$$P'' = Rx_{(\theta_2)} * Rx_{(\theta_1)} * P$$

$$P'' = Rx_{(\theta_1 + \theta_2)} * P$$

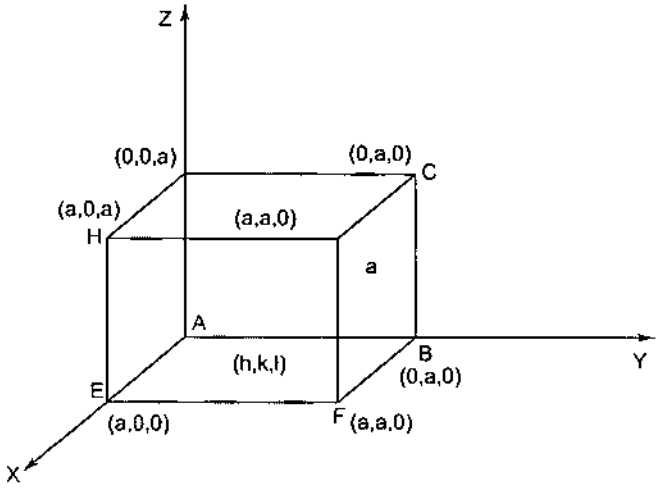
That is not the example of Tilting, because here we rotate the point about the same Axis.

Example. *A cube of side (a) is placed at (h, k, l) such that its edges are parallel to three axes. Rotate cube by θ about point A about X-Axis.*

Solution.



Firstly we describe how to calculate the points of a cube.



NOTES

Step 1. Translate point A to origin.

$$P' = T_{(-h,-k,-k)} * P \quad \ldots [A]$$

Step 2. Rotate by θ about X-Axis.

$$P'' = Rx_{(\theta)} * P' \quad \ldots [B]$$

Step 3. Back translate the point.

$$P''' = T_{(h,k,k)} * P'' \quad \ldots [C]$$

Now from equation [A] and [B] we have

$$P''' = T_{(h,k,k)} * Rx_{(\theta)} * T_{(-h,-k,-k)} * P \quad \ldots [D]$$

Now

$$T_{(h,k,k)} = \begin{bmatrix} 1 & 0 & 0 & h \\ 0 & 1 & 0 & k \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{(-h,-k,k)} = \begin{bmatrix} 1 & 0 & 0 & -h \\ 0 & 1 & 0 & -k \\ 0 & 0 & 1 & -l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rx_{(\theta)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And the Matrix P is as follows if we treat that point is at origin

$$P = \begin{bmatrix} A & B & C & D & E & F & G & H \\ 0 & 0 & 0 & 0 & a & a & a & a \\ 0 & a & a & 0 & 0 & a & a & 0 \\ 0 & 0 & a & a & 0 & 0 & a & a \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

And if the point is at (j, k, l) then the point will be

$$P = \begin{bmatrix} h & h & h & h & h+a & h+a & h+a & h+a \\ k & k+a & k+a & k & k & k+a & k+a & k \\ l & l & l+a & l+a & l & l & l+a & l+a \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

4.6 ALIGNING A VERTEX WITH Z-AXIS

Aligning Matrix is equivalent to two rotations, *i.e.*, Rotation by some angle about same Axis followed by rotation by some other angle about some other Axes.

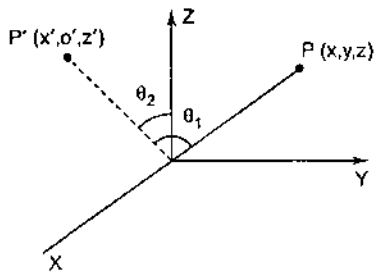


Fig. 4.2

$$AVz = Ry_{(-\theta_2)} * Rx_{(\theta_1)} \quad [A] \quad \text{X-Z Plane}$$

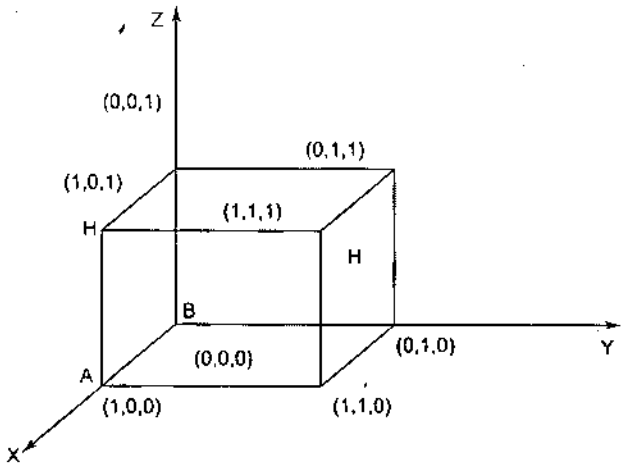
Similarly we have

$$AVx = Rz_{(-\theta_2)} * Ry_{(\theta_1)} \quad [B] \quad \text{X-Y Plane}$$

Similarly we have

$$AVy = Rx_{(-\theta_2)} * Rz_{(\theta_1)} \quad [C] \quad \text{Y-Z Plane}$$

Example. *Rotate a unit cube by 45° about its main diagonal.*



Solution.

Step 1. Align vertex BH with Z-Axis

$$P' = Ry_{(-\theta_2)} * Rx_{(\theta_1)} * P \quad \ldots [A]$$

Step 2. Rotate about Z-Axis by 45°

$$P'' = Rz_{(45°)} * P' \quad \ldots [B]$$

Step 3. Back alignment of BH

$$P''' = Ry_{(\theta_2)} * Rx_{(-\theta_1)} * P'' \quad \ldots [C]$$

Now from equation [A], [B] and [C] we have-

$$P''' = Ry_{(\theta_2)} * Rx_{(-\theta_1)} * Rz_{(45°)} * Ry_{(-\theta_2)} * Rx_{(\theta_1)} * P \quad \ldots [D]$$

In this case $a = b = c = 1$

$$\sin\theta_1 = \left(\frac{b}{\sqrt{b^2 + c^2}}\right)$$

$$\theta_1 = \sin^{-1}\left(\frac{1}{\sqrt{1+1}}\right) = \sin^{-1}\left(\frac{1}{\sqrt{2}}\right)$$

$$\theta_1 = 45°$$

$$\theta_2 = \sin^{-1}\left(\frac{a}{\sqrt{a^2 + b^2 + c^2}}\right) = \sin^{-1}\left(\frac{1}{\sqrt{1+1+1}}\right) = \sin^{-1}\left(\frac{1}{\sqrt{3}}\right)$$

$$\cos\theta_2 = \frac{2}{\sqrt{3}} \quad \text{Ans.}$$

Example. *Rotate a unit cube by 45° about its main diagonal. Unit cube is placed such that its three edges are parallel to three principle Axes and lowest vertex is at (2, 3, 4).*

Solution.

The following steps are to be followed

NOTES

NOTES

Step 1. Translate the lowest point to the origin

$$P' = T_{(-2,-3,-4)} * P \quad \ldots [A]$$

Step 2. Align the main diagonal to the Z-Axis

$$P'' = Ry_{(\theta_2)} * Rx_{(-\theta_1)} * P \quad \ldots [B]$$

Step 3. Perform Rotate about Z-Axis by 45°

$$P''' = Rz_{(45°)} * P'' \quad \ldots [C]$$

Step 4. Back alignment

$$P'''' = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * P''' \quad \ldots [D]$$

Step 5. Back Translate

$$P''''' = T_{(2,3,4)} * P'''' \quad \ldots [E]$$

Now from equation [A], [B], [C], [D] and [E] we have

$$P''''' = T_{(2,3,4)} * Rx_{(-\theta_1)} * Ry_{(\theta_2)} * Rz_{(45°)} * Ry_{(-\theta_2)} * Rx_{(\theta_1)} * T_{(-2,-3,-4)} * P \text{ Ans.}$$

4.7 MIRROR REFLECTION

In 3-D, we have mirror reflect the object about the planes, *i.e.*, we mirror reflect the object about XY plane, we mirror reflect the object about YZ plane, we mirror reflect the object about ZX plane.

Mirror Reflection of an Object about XY Plane

If we mirror reflect the object about XY plane then the Z coordinate is negative, *i.e.*

$$X' = X$$

$$Y' = Y$$

And $Z' = -Z$

We can represent this as follows

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = Mxy * P$$

Where

$$Mxy = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ is called the mirror reflection matrix about XY Plane.

Mirror Reflection of an Object about YZ Plane

If we mirror reflect the object about YZ plane then the X coordinate is negative, *i.e.*

$$X' = -X$$

$$Y' = Y$$

And

$$Z' = Z$$

We can represent this as follows

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = Myz * P$$

Where

$$Myz = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ is called the mirror reflection matrix about YZ Plane.

NOTES

Mirror Reflection of an Object about XZ Plane

If we mirror reflect the object about XZ plane then the Y coordinate is negative, *i.e.*

$$X' = X$$

$$Y' = -Y$$

And

$$Z' = Z$$

We can represent this as follows

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = Mxz * P$$

Where

$$Mxz = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ is called the mirror reflection about XZ Plane.

Example. *Mirror reflect a point in 3-D about a given plane, space is fixed by a normal to the plane* $(\overline{N})$ *and a point known as reference point* R_0 (X_0, Y_0, Z_0) *on the plane.*

Solution.

Step 1. Translate the reference point to the origin

$$P' = T_{(-x_0, -y_0, -z_0)} * P \quad \ldots [A]$$

Step 2. Align the normal of the plane with the Z-Axis

$$P'' = Ry_{(-\theta_2)} * Rx_{(\theta_1)} * P' \quad \ldots [B]$$

Step 3. Take mirror reflection about XY Plane

$$P''' = Mxy * P'' \quad \ldots [C]$$

Step 4. Back alignment

$$P'''' = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * P''' \quad \ldots [D]$$

Step 5. Back Translate

$$P''''' = T_{(x_0, y_0, z_0)} * P'''' \quad \ldots [E]$$

Now from equation |A|, |B|, |C|, |D| and |E| we have

$$P'''' = T_{(x_0, y_0, z_0)} * Rx_{(-\theta_1)} * Ry_{(\theta_2)} * Mxy * Ry_{(-\theta_2)} * Rx_{(\theta_1)} * T_{(-x_0, -y_0, -z_0)} * P \text{ Ans.}$$

Example. *Find the matrix for mirror reflection with respect to the plane passing through the origin and having a normal vector whose direction is M = I + J + K.*

Solution.

$$N = I + J + K \text{ and } i = 1, j = 1, k = 1$$

NOTES

Step 1. Align the normal to the Z-Axis

$$P' = Ry_{(-\theta_2)} * Rx_{(\theta_1)} * P \quad \ldots |A|$$

Step 2. Take mirror reflection about XY Plane

$$P'' = Mxy * P' \quad \ldots |B|$$

Step 3. Back alignment

$$P''' = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * P'' \quad \ldots |C|$$

Now from equation |A|, |B| and |C| we have

$$P''' = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * Mxy * Ry_{(-\theta_2)} * Rx_{(\theta_1)} * P$$

$$P''' = M * P$$

So the mirror M is as follows

$$M = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * Mxy * Ry_{(-\theta_2)} * Rx_{(\theta_1)} \quad \ldots |D|$$

$$\sin\theta_1 = \left(\frac{b}{\sqrt{b^2 + c^2}}\right)$$

$$\theta_1 = \sin^{-1}\left(\frac{1}{\sqrt{1+1}}\right) = \sin^{-1}\left(\frac{1}{\sqrt{2}}\right)$$

$$\theta_1 = 45°$$

$$\sin\theta_2 = \left(\frac{a}{\sqrt{a^2 + b^2 + c^2}}\right) = \left(\frac{1}{\sqrt{1+1+1}}\right) = \left(\frac{1}{\sqrt{3}}\right)$$

$$\cos\theta_2 = \left(\frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}}\right) = \left(\frac{\sqrt{1+1}}{\sqrt{1+1+1}}\right) = \left(\sqrt{\frac{2}{3}}\right)$$

Now putting these values in equation |D| we can calculate the value M Ans.

4.8 THREE-DIMENSIONAL VIEWING

The 3-D viewing process is inherently more complex than the 2-D viewing.

In 2-dimensional viewing we have 2-D window and 2 view port and objects in the word coordinates are clipped, against the window and are then transformed into the view port for display.

The complexity added in the three-dimensional viewing is because of the added diversion and the fact that even though objects are three-dimensional and the display device are only 2-D.

The mismatch between 3-D objects and 2-D displays is compensated by introducing Projections.

The Projection transforms 3-D objects into a 2-D projection plane. The following figure shows the conceptual model of the 3-D transformation process.

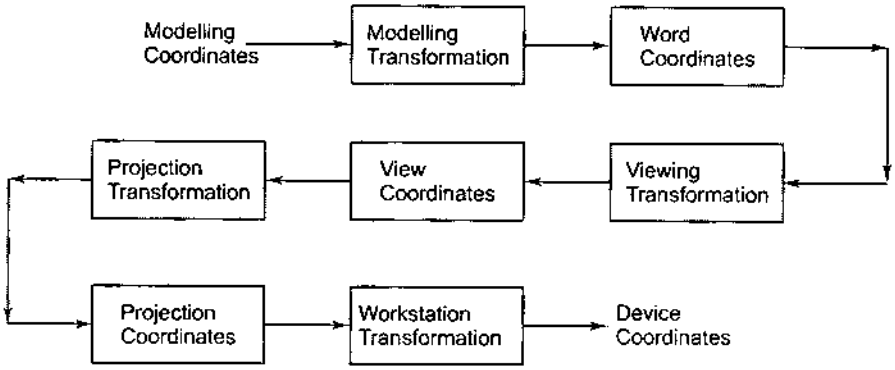


Fig. 4.3 Conceptual model of 3-D transformation process

In 3-D viewing, we specify a view volume in the word coordinates with modeling transformation. The word coordinate positions of the objects are then converted into viewing coordinates by viewing transformation. The Projection transformation is then used to convert 3-D description of objects in viewing coordinates to the 2-D projection coordinates. Finally the Workstation transformation transfers the projection coordinates into the device coordinates.

Let us consider that we have to write a flight simulator program. The first thing to be done is to construct a model of the world over which the pilot is to fly. Buildings, fields, runways, lakes and other scenes may be constructed using 3-D line and polygon primitives. Windowing and modeling allows us real world dimensions. Therefore model of the world is represented using the word coordinates. Then object description is converted from world coordinates to viewing coordinates. This produces the view which the pilot can see from his Airplane. The projection transformation is then used to convert 3-D description of the object in viewing coordinates to the 2-D projection coordinates. The 2-D projection coordinates are converted into device coordinates which are used to display the picture on the video display.

4.9 VIEWING PARAMETERS

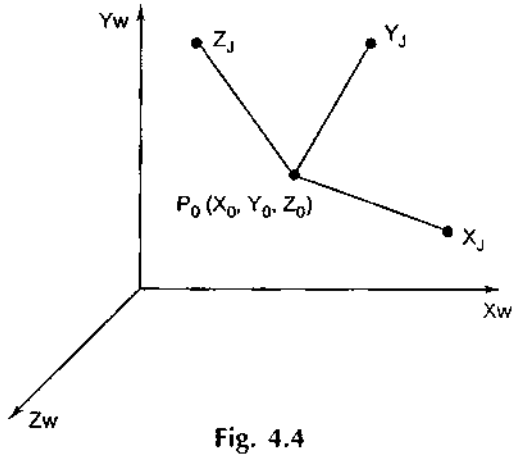


Fig. 4.4

As mentioned earlier, we can view the objects from the side or the top or even from behind. Therefore it is necessary to choose a particular view from a picture by first defining a view plane.

NOTES

A view plane is nothing but the film plane in a camera which is positioned and oriented for a particular shot of the scene. Word coordinate position in the scene is transformed to viewing coordinates, and then viewing coordinates are projected on to the view plane. A view plane can be defined by establishing the viewing coordinates system or view reference coordinate system.

View Reference Point

NOTES

The first viewing parameter we must consider is the view reference point. This point is the center of our viewing coordinate system. It is often choosing to be close to or the surface of the object in a scene. Its coordinates are specified as X_R, Y_R and Z_R.

View Plane Normal Vector

The next viewing parameter is a View Plane Normal Vector. This Normal Vector is the direction perpendicular to the view plane and it is defined as [DXN, DYN, DZN].

WE KNOW THAT THE View Plane is the film in the camera and we focus camera towards the View Reference Point. This means that the camera is pointed in the direction of the View Plane Normal. Graphically it is represented as follows :

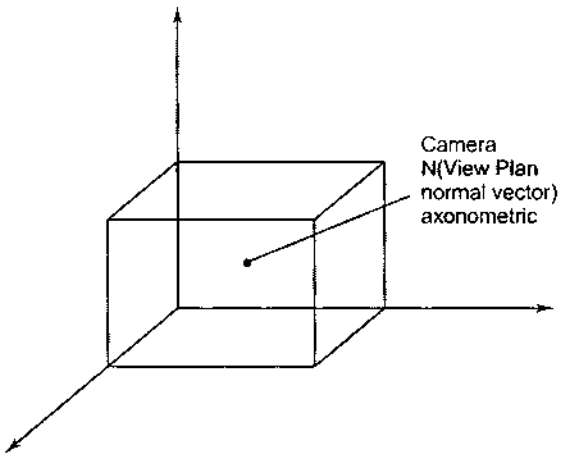


Fig. 4.5

As shown in the above figure, the View Plane Normal Vector is a directed line system from the view plane to the View Reference Point. The length of this directed line segment is referred to as View Distance.

NOTE It is possible to obtain the different views by rotating the camera about the View Plane Normal Vector and keeping view reference point and direction of [N] vector fixed.

At different angles, the view plane will show the same scene, but rotated so that a different part of the object is up. The rotation of a camera or view plane is specified by a View Up Vector V[XUP, YUP, ZUP] which is another important viewing parameter.

4.10 PROJECTION

Projection means displaying a 3-D object (in real world) as a 2-D object in physical device coordinate system.

Classification of Projection

The classification of the Projection is as follows:

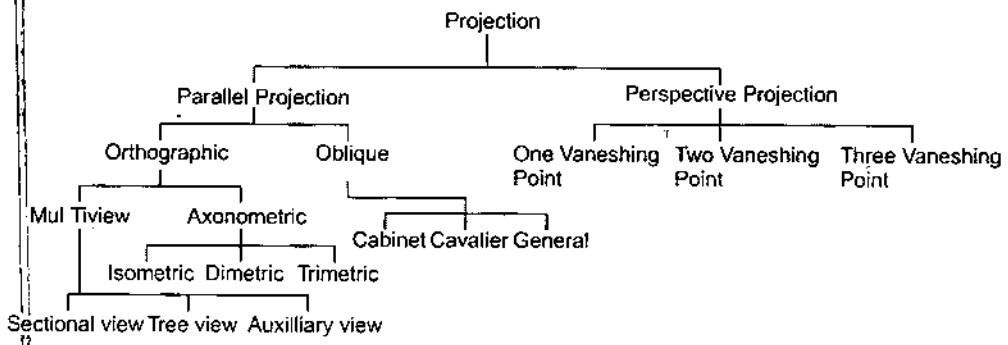


Fig. 4.6

Parallel Projection

Parallel Projection is specified by

(*a*) Direction of Projection

(*b*) View Plane.

In Parallel Projection, Z coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane. The point of intersection is the Projection of the vertex. We connect the projected vertex by line segments which correspond to coordinates on the original object.

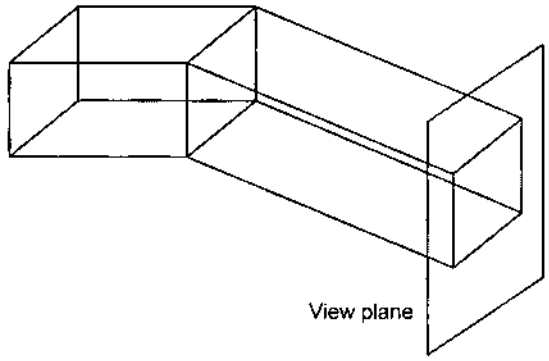


Fig. 4.7 Parallel projection of an object to the view plane

Perspective Projection

Perspective Projection is specified by

(*a*) Point of Projection

(*b*) View Plane.

Orthographic Projection

Orthographic Projection is a special kind of parallel projection in which the direction of projection is perpendicular to the View Plane.

Oblique Projection

Oblique projection is a special kind of parallel projection in which the direction of projection make any angle to the View Plane except 90°.

NOTES

NOTES

Cabinet Projection

Cabinet Projection is a special type of Oblique Projection in which the length of the projection point is half than the actual position of the point.

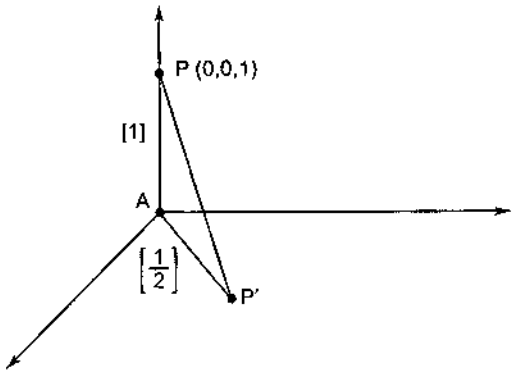


Fig. 4.8

IF AP = 1
THEN AP' = 1/2
Then it is called as Cabinet Projection.

Cavalier Projection

Cavalier Projection is the special type of Oblique Projection in which the length of the projection point is same as the actual position of the point.

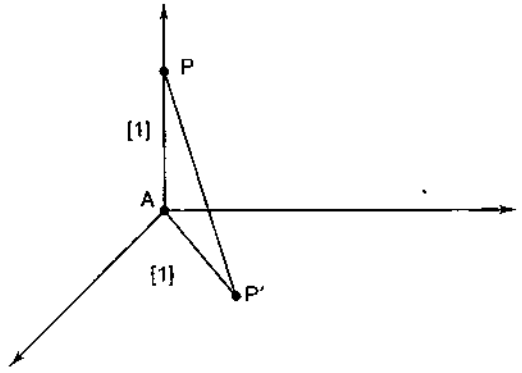


Fig. 4.9

IF AP = 1
THEN AP' = 1
Then it is called as Cavalier Projection.

General Projection

If the projection is neither Cabinet nor Cavalier then the projection is called General Projection.

Multi View Projection

Multi View is a special type of Orthographic Projection, in which we can see the various views of the picture.

Axonometric Projection

Axonometric is a special type of Orthographic projection which tells how the direction of projection makes the angle with the three principles Axis.

Isometric Projection

Isometric Projection is a special type of Axonometric Projection, in which the direction of projection makes equal angle with the three principles Axis.

Diametric Projection

Diametric Projection is a special type of Axonometric Projection, in which the direction of projection makes equal angle with two principles Axis.

NOTES

Trimetric Projection

Trimetric Projection is a special type of Axonometric Projection, in which the direction of projection makes unequal angle with three principles Axis.

Sectional View

Sectional View is a special type of Multi View Projection, in which the image is ivied in the form of sections.

Tree View

Tree View is a special type of Multi View Projection, in which the tree like structure i.e. hierarchical structure is visible.

Auxiliary View

If the projection is neither Sectional View nor Tree View, then it is called as Auxiliary View.

Perspective Projection

Perspective Projection is specified by a point called center point and a View Plane. Graphically the Perspective Projection of any line is represented as follows :

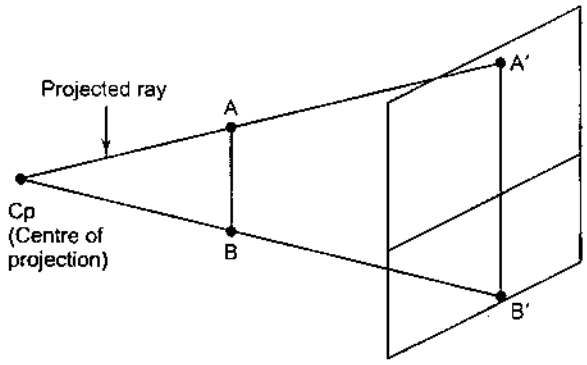


Fig. 4.10 A'B' is the perspective projection image of AB.

The Classification of Perspective Projection is as follows :

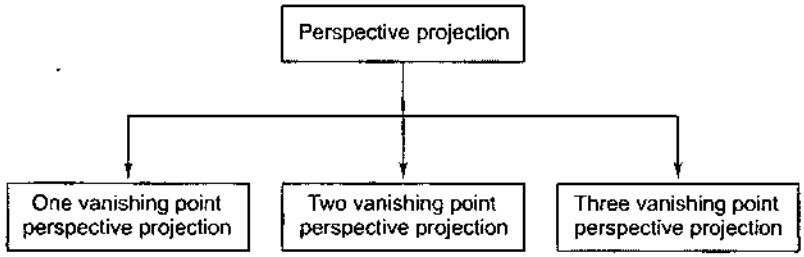


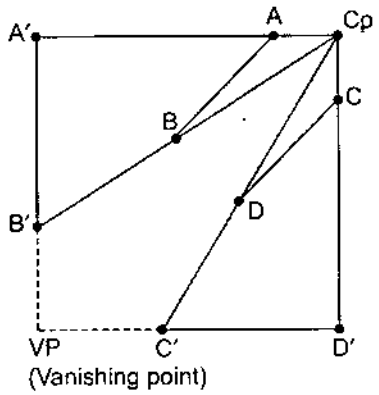
Fig. 4.11

NOTES

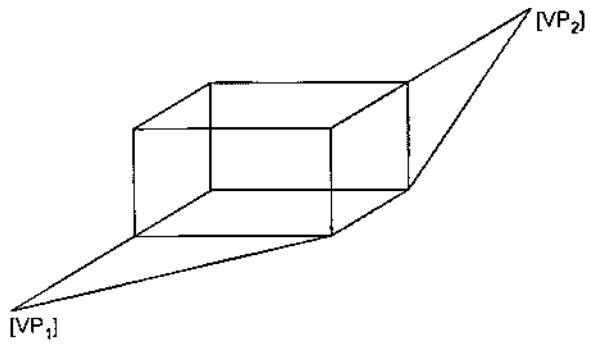
4.11 VANISHING POINT

Perspective projected image of two parallel lines will not be two parallel lines. They will be non parallel and on extending these lines they meet at some point is known as Vanishing Point.

Example of One Vanishing Point Perspective Projection



Example of Two Vanishing Point Perspective Projection



If we have two Vanishing points, then it is called as Two Vanishing Point Perspective Projection.

4.12 THREE VANISHING POINT PERSPECTIVE PROJECTION

If we have three Vanishing points, then it is called as Three Vanishing Point Perspective Projection. Graphically it is represented as follows :

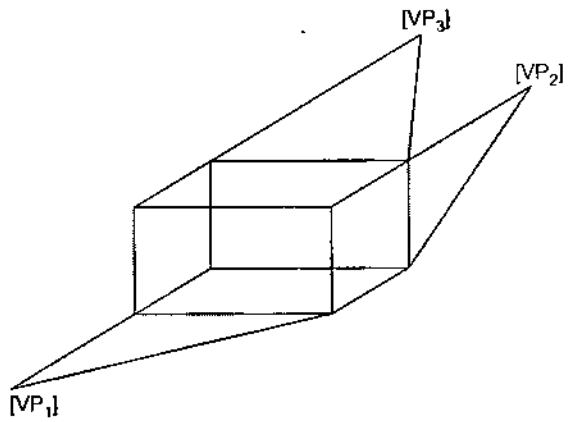


Fig. 4.12

Four Anomalies of Perspective Projection

In Perspective Projection, we have four anomalies and that four anomalies are as follows :

(*a*) Perspective foreshortening (*b*) Concept of vanishing point

(*c*) View confusion (*d*) Topological distortion.

(*a*) **Perspective Foreshortening:** Perspective foreshortening means, the perspective image of the farther object from the center of point will be smaller. Graphically it is represented as follows :

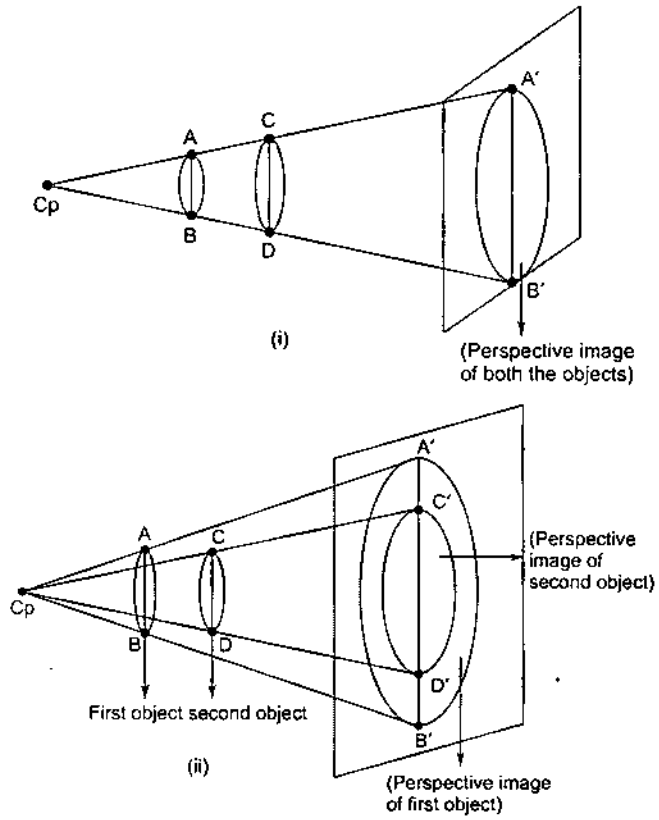


Fig. 4.13(a)

(*b*) **Concept of Vanishing Point:** Perspective projected image of two parallel lines will not be two parallel lines. They will be non parallel and on extending these lines they meet at some point is known as Vanishing Point. Graphically it is represented as follows :

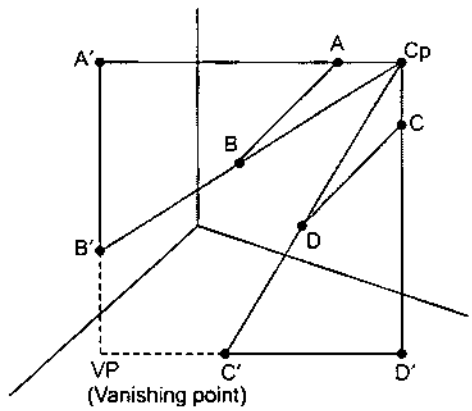


Fig. 4.13(b)

NOTES

NOTES

(c) **View Confusion:** If center of point is between object and view plane, then the perspective image of the object will be inverted then it is called as View Confusion. Graphically it is represented as follows :

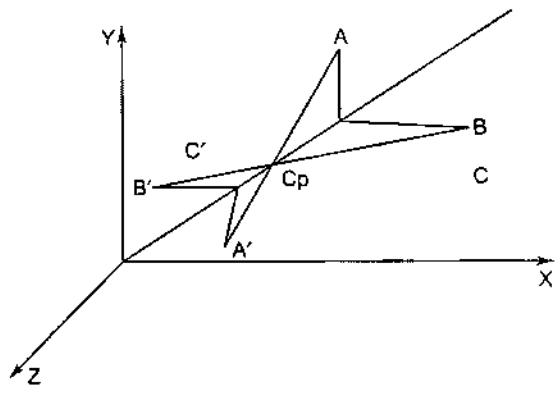


Fig. 4.13(c)

(d) **Topological Distortion:** Assume a plane passing through center of point and parallel to View Plane, then perspective image of any point lying on this plane will be formed at infinity, then it is known as Topological distortion. Graphically it is represented as follows :

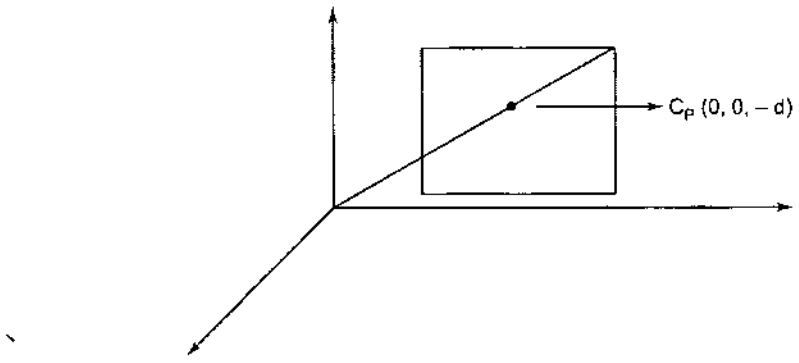


Fig. 4.13(d)

4.13 STANDARD PERSPECTIVE PROJECTION MATRIX

When center of projection line lies at negative Z-Axis and View Plane is XY Plane, then it is known as Standard Perspective Projection Matrix.

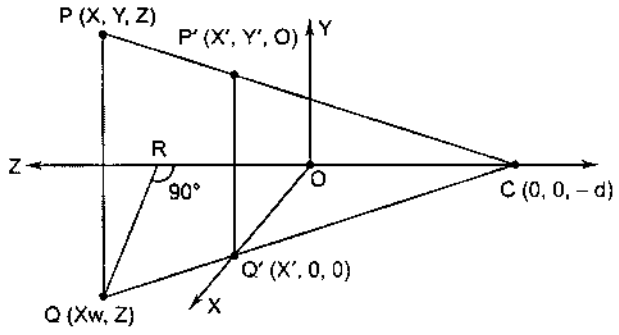


Fig. 4.14

Now in this case

$$\left(\frac{CO}{RC}\right) = \left(\frac{OQ'}{RQ}\right)$$

$$\left(\frac{d}{d+z}\right) = \left(\frac{X'}{X}\right)$$

$$X' = \left(\frac{dx}{d+z}\right) \quad \ldots [A]$$

NOTES

Similarly we have

$$Y' = \left(\frac{dY}{d+z}\right) \quad \ldots [B]$$

$$Z' = 0$$

Now we can represent this in the form of Matrix

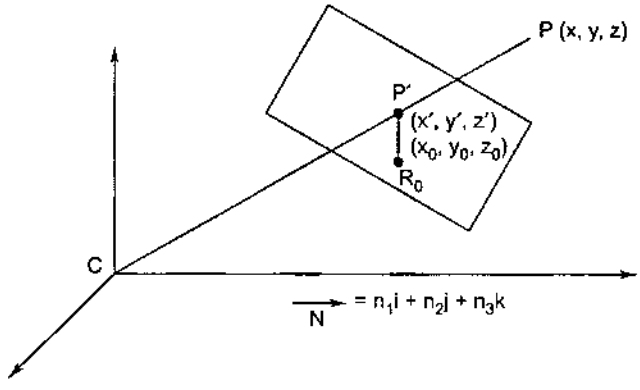
$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P' = P_{Prespective} * P$$

Where

$$P_{Prespective} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix}$$ is known as Perspective Projection Matrix.

Example. *Find the Projection Matrix. If the View Plane is any whose Normal is* $\overrightarrow{N}$ *and reference point is* $R_0(X_0, Y_0, Z_0)$ *and center of point is origin.*



Since P and P' both are on the same line then we have

$$P' = \alpha CP \quad \ldots [A]$$

Where α is the constant.

We have the point P(x, y, z) and the point P' (x', y', z'), so we have

$$X' = \alpha X \quad \ldots [B]$$

$$Y' = \alpha Y \quad \ldots [C]$$

$$Z' = \alpha Z \quad \ldots [D]$$

And we also have

NOTES

$$\overrightarrow{PR} \cdot \overrightarrow{N} = 0$$

$$= [(x' - x_0)i + (y' - y_0)j + (z' - z_0)k] \cdot [n_1 i + n_2 j + n_3 k] = 0$$

$$= (x' - x_0)n_1 + (y' - y_0)n_2 + (z' - z_0)n_3 = 0 \quad \ldots [E]$$

Since we have

$$i.i = j.j = k.k = 1$$

Now putting the values of (x', y', z') from equation |B|, |C|, and |D| in equation |E| we have

$$= (\alpha x - x_0)n_1 + (\alpha y - y_0)n_2 + (\alpha z - z_0)n_3 = 0$$

$$= \alpha (n_1 x + n_2 y + n_3 z) = x_0 n_1 + y_0 n_2 + z_0 n_3$$

$$\alpha = \left(\frac{x_0 n_1 + y_0 n_2 + z_0 n_3}{n_1 x + n_2 y + n_3 z} \right) \quad \ldots [F]$$

Let $d = (x_0 n_1 + y_0 n_2 + z_0 n_3)$

Now from equation No. [F], we have

$$\alpha = d/(n_1 x + n_2 y + n_3 z) \quad \ldots [G]$$

$$x' = \alpha x = dx/(n_1 x + n_2 y + n_3 z) \quad \ldots [H]$$

$$y' = \alpha y = dy/(n_1 x + n_2 y + n_3 z) \quad \ldots [I]$$

$$z' = \alpha z = dz/(n_1 x + n_2 y + n_3 z) \quad \ldots [J]$$

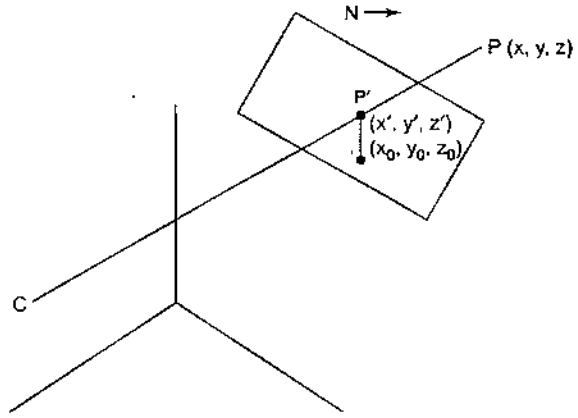
Now we can represent this in the form of Matrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ n_1 & n_2 & n_3 & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = P_{pers} * P$$

Where $P_{pers} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ n_1 & n_2 & n_3 & 0 \end{bmatrix}$ is called Projection Matrix.

Example. *If center of projection is not origin and it is some other point (a, b, c) and view plane is any plane then find the projection matrix.*



The following steps takes place.

Step 1. Translate the whole system into origin

$$P' = T_{(-a, -b, -c)} * P \quad \ldots [A]$$

Step 2. Take the Perspective Projection

$$P'' = P_{perspective} * P' \quad \ldots [B]$$

Step 3. Back translate the whole system into new position

$$P''' = T_{(a,b,c)} * P'' \quad \text{... [C]}$$

Now from equation [A], [B], and [C] we have

$$P''' = T_{(a,b,c)} * P_{perspective} * T_{(-a,-b,-c)} * P$$
$$P''' = M * P$$

Where M is the Composition Matrix.

$$M = T_{(a,b,c)} * P_{perspective} * T_{(-a,-b,-c)}$$

NOTES

Example. *If P = (– 1, 1, – 2d), Q = (2, – 2, 0) and CP = (0, 0, – d) calculate P′Q′, taking parametric equation of the line.*

$$X = X_1 + (X_2 - X_1)t$$
$$Y = Y_1 + (Y_2 - Y_1)t$$
$$Z = Z_1 + (Z_2 - Z_1)t$$

This is the case, when the topological distortion take place.

Solution.

Given equations are

$$X = X_1 + (X_2 - X_1)t \quad \text{... [A]}$$
$$Y = Y_1 + (Y_2 - Y_1)t \quad \text{... [B]}$$
$$Z = Z_1 + (Z_2 - Z_1)t \quad \text{... [C]}$$

Now putting the values we have

$$X = -1 + (2 + 1)t$$
$$X = -1 + 3T \quad \text{... [D]}$$
$$Y = 1 + (-2 - 1)t$$
$$Y = 1 - 3T \quad \text{... [E]}$$
$$Z = -2d + (0 + 2d)t$$
$$Z = -2d + 2dt \quad \text{... [F]}$$

Now this is the case of standard Perspective Projection Matrix because the View Plane is XY and CP is on negative Z-Axis. So we have

$$P' = P_{perspective} * P$$

$$P' = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix} * \begin{bmatrix} -1+3t \\ 1-3t \\ -2d+2dt \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} d(-1+3t) \\ d(1-3t) \\ 0 \\ -d+2dt \end{bmatrix}$$

Now by comparing these values we have

$$X' = \frac{d(-1+3t)}{-d+2dt} \quad \text{... [G]}$$

$$Y' = \frac{d(1-3t)}{-d+2dt} \quad \text{... [H]}$$

$$Z' = 0 \quad \text{... [I]}$$

Now from [G], and [H] we have, that the denominator is 0 when t = ½

So at $t = \frac{1}{2}$

$X' = \infty$ and $Y' = \infty$

NOTES

Now when $t = 0$, then we calculate point P'

$$X' = \frac{d(-1+3*1)}{-d+2d*1} = d(-1) / d = 1$$

$$Y' = \frac{d(-1+3*0)}{-d+2d*0} = d / -d = -1$$

$$Z' = 0$$

So the coordinate of P' is (1, – 1, 0)

Now when $t = 1$, we calculate point Q

$$X' = \frac{d(-1+3*1)}{-d+2d*1} = d(2) / d = 2$$

$$Y' = \frac{d(1-3*1)}{-d+2d*1} = -2d / d = -2$$

$$Z' = 0$$

Then the coordinate of Q' is (2, – 2, 0)

The Perspective image of PQ is P' Q', but it is not continuous and at $t = 1/2$ $P' = \infty$ and $Q' = \infty$ Ans.

4.14 3-D CLIPPING

We know that, in two-dimensional coordinate system, the concepts of window are served as clipping boundary. But in three-dimensional coordinate system, the concept can be extended to a clipping volume or view volume. The clipping volume can be either a box (rectangular parallelepiped) or transform of vision (a truncated pyramidal volume). The box is normally used for parallel projections, where the transform of visions is used for perspective projection.

Graphically it is represented as follows :

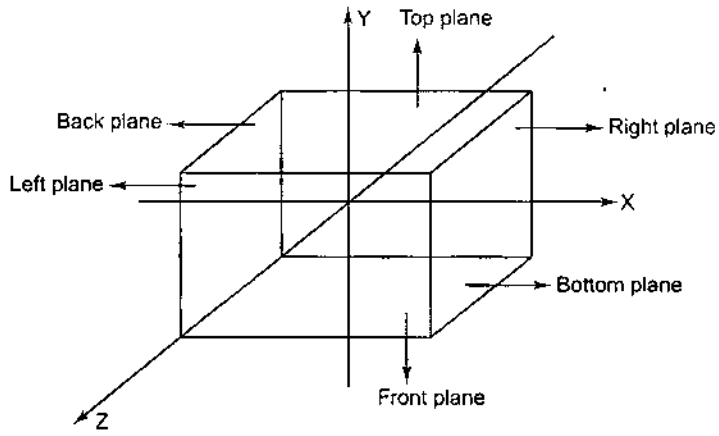


Fig. 4.15

We can make use of Cohen and Sutherland end-point code for identifying totally or partially visible lines. Here we can use 6 bit end-point code. Now starting from left to right (First bit is leftmost bit), we have the bit setting as indicated below:

Bit 1 is set if the point is behind the volume.

Bit 2 is set if the point is in front of the volume.

Bit 3 is set if the point is above the volume.

Bit 4 is set if the point is below of the volume.

Bit 5 is set if the point is right of the volume.

Bit 6 is set if the point is left of the volume.

The summary of the end-bit code setting is as follows

Bit	1	2	3	4	5	6
	Behind	Front	Above	Below	Right	Left

As an example, a region code of 101000 identified a point as above and behind the view volume and the region code 000000 indicates a point within the view volume.

NOTES

A line segment can be immediately identified as completely within the view volume if both end-points have a region code of 000000. If either end-point of a line segment does not have a region code of 000000, we perform the logical AND operation on the two end-point codes. If the result of this AND operation is nonzero then both end-points are outside the view volume and line segment is completely invisible. On the other hand if the result of AND operation is zero, then the line segment is partially visible. In this case, it is necessary to determine the intersection of the line and the clipping volume.

4.15 THREE-DIMENSIONAL MIDPOINT ALGORITHMS

Algorithm

1. Find the location of end points (end-point codes) of line segments with respect to display volume.
2. Check visibility of each line segment.
   (*a*) If codes for both end points are zero, then the line is completely visible. Hence draw the line and go to step-4.
   (*b*) If codes for end points are not zero and the logical AND operation of them is also nonzero, then the line is completely invisible. So reject the line and go to step-4.
   (*c*) If codes of both end points do not satisfy the conditions in 2(*a*) and 2(*b*) then the line is partially visibly.
3. Divide the partially visible line segment in equal parts and repeat step-1 and step-2 for subdivided line segments until you got completely visible and completely invisible line segments. Draw the visible line segment and discard the invisible one.
4. Stop.

Example. *Obtain transform matrix for rotation about the line joining two points (0, 0, 0) and (1, 1, 1) with the angle of rotation 45° in outer clockwise area.*

Solution.

Step 1. Align the line with Z-Axis

$$P' = Ry_{(-\theta_2)} * Rx_{(\theta_1)} * P \quad \ldots [A]$$

Step 2. Perform the Rotation

$$P'' = Rz_{(45°)} * P' \quad \ldots [B]$$

Step 3. Back Alignment

$$P''' = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * P'' \quad \ldots [C]$$

From equation [A], [B], and [C]; we have

$$P''' = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * Rz_{(45°)} * Ry_{(-\theta_2)} * Rx_{(\theta_1)} * P$$

$$P''' = M * P$$

NOTES

Now by comparing the values we get

$$M = Rx_{(-\theta_1)} * Ry_{(\theta_2)} * Rz_{(45°)} * Ry_{(-\theta_2)} * Rx_{(\theta_1)} \quad \ldots [D]$$

$a = 1, b = 1, c = 1$

$$\sin\theta_1 = \left(\frac{b}{\sqrt{b^2 + c^2}}\right) = \left(\frac{1}{\sqrt{1+1}}\right) = \left(\frac{1}{\sqrt{2}}\right)$$

$\theta_1 = 45°$

$$\sin\theta_1 = \left(\frac{c}{\sqrt{a^2 + b^2 + c^2}}\right) = \left(\frac{1}{\sqrt{1+1+1}}\right) = \left(\frac{1}{\sqrt{3}}\right)$$

$$\cos\theta_2 = \left(\frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}}\right) = \left(\frac{\sqrt{1+1}}{\sqrt{1+1+1}}\right)\left(\sqrt{\frac{2}{3}}\right)$$

Now with the help of equation [D] we can directly find the values of the Matrix. Ans.

SUMMARY

- Transformation is a process carried out by means of transformation these objects, or changing the size of the object or changing the orientation of the object or may be any combination of these.
- Translation is a process of changing the position of an object.
- Scaling is a transformation, which either magnifies or reduces the size of the object.
- In the rotation, we try to rotate the object by a given angle.
- In mirror reflection, we need to know the reference plane, *i.e.* plane about which the reflection is to be taken.
- Tilting is equivalent to rotation about same Axis by some angle θ followed by rotation about some other Axis by some other angle ϕ.
- Projection is a process of representing a three-dimensional object or scene into two dimensional medium.
- Natural object can be shown by 3-D.
- For Rotation about an arbitrary Axis we first translate it to the origin, then align the Axis on the Z-Axis and then rotate by a given angle and then Back Alignment.
- There are different viewing parameters such as the view reference point, view plane normal vector, view distance etc.
- For 3-D Clipping Sutherland Cohen algorithm are used.
- The Shape between the front plane and the back plane is called view volume, which are used for 3-D Clipping.

REVIEW QUESTIONS

1. Derive the Matrix for the rotation about Z-Axis by an angle θ.
2. Find Matrix for rotation about any arbitrary line by angle θ, whose direction vector V = *ai* + *bj* + *ck*.
3. Rotate a cube by angle 450 about Y-Axis whose coordinates are A(0, 0, 0), B(0, 1, 0), C(0, 1, 1), D(0, 0, 1), E(1, 0, 0), F(1, 0, 1), G(1, 1, 0), H(1, 1, 1) and find the coordinate after rotation.
4. Differentiate between Parallel and Perspective Projection with suitable example.
5. Derive the 3-D transformation matrix to transform World Coordinate to Viewing Coordinate.

6. Define Parallel Projection with suitable example.
7. Define Perspective Projection with suitable example.
8. Explain various types of Parallel Projection with suitable example.
9. Explain various types of Perspective Projection with suitable example.
10. Why we need 3-D Clipping? Explain in detail.
11. Write short note on the following:
    (a) 3-D Clipping
    (b) 3-D Scaling
    (c) Perspective Projection
    (d) Parallel Projection
    (e) 3-D Rotation
12. Write a routine to implement rotations by any specified angle in a frame butter block.
13. Derive the 3-D transformation matrix for reflecting a point about a plane.
14. Write an algorithm to display a cube on the screen and rotate it through any angle. The distortion of the image should be minimum before and after Rotation.
15. Give the mathematical description of the Perspective Projection.

NOTES

FURTHER READINGS

- **Computer Graphic:** V.K. Pachghare, Laxmi Publications, 2007, Second edition.
- **Computer Graphics:** Prabhakar Gupta, Vineet Agarwal and Manish Varshney, Laxmi Publications, 2011.
- **Computer Graphics:** Rajiv Chopra, S. Chand Publisher, 2011.
- **Computer Graphics:** C.S. Verma, Ane Books, 2011.
- **Computer Graphics:** Pradeep K. Bhatia, I.K. International, 2009, pbk, Second Edition.
- **Computer Graphics:** Ruchi Mishra, Global Vision Publisher, 2010.

UNIT V

NOTES

HIDDEN LINES, SURFACES CURVE GENERATION AND ANIMATION

STRUCTURE

5.0 LEARNING OBJECTIVES

After going through this unit, you should be able to:

- explain hidden surface algorithms
- explain fractals
- discuss about generation of curves
- describe about rendering and animation.

NOTES

5.1 HIDDEN SURFACE ALGORITHMS

A major part of rendering (making images more realistic) is the visible surface problem, *i.e.*, only display those surfaces which should be visible.

Graphically it is represented as follows:

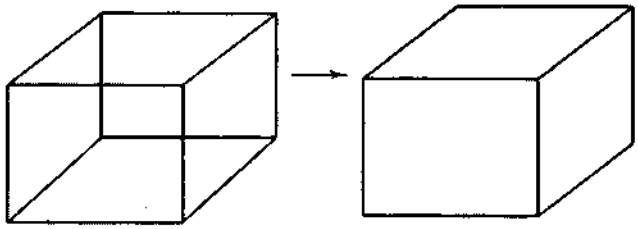


Fig. 5.1

This was one of the major areas of activity in computer graphics and many different algorithms were developed. The algorithms can be divided into two classes:

Object space ——→ Word Coordinates

Image space ——→ PDC Coordinates

Hidden Surface Algorithms are usually image space or a combination of object and image space. Below are discussed some of the many visible surface determination algorithms.

Object Space Method

Object space method is implemented in the physical device coordinated system in which objects are described. It compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole we should label as visible. Object space methods are generally used in Line-display algorithms.

Image Space Method

Image space method is implemented in the screen coordinate system in which the objects are viewed. In an image space algorithm, visibility is decided point by point at each pixel position in the view plane. Most hidden line/surface algorithms use Image Space Methods.

5.2 TECHNIQUES FOR EFFICIENT VISIBLE SURFACE ALGORITHMS

We have seen that there are two basic approaches used for visible surface detection which are as follows:

1. Object Precision Algorithm
2. Image Precision Algorithm.

In both the algorithms we require to perform a number of potentially costly operations such as determination of projections of objects, whether or not they intersect and where

they intersect, closest object in case of intersection and so on. To create and display picture in minimum time we have to perform visible surface algorithms more efficiently.

The techniques to perform visible surface algorithms efficiently are discussed in further sections.

5.3 COHERENCE

NOTES

The issue of coherence is often mentioned in relation to hidden surface removal, since many of these algorithms make use of it in some fashion. It does have widespread application to many areas of computer graphics. Coherence denotes similarities between items or entities. It describes the extent to which these items or entities are locally constant. In many situations properties do not change drastically but rather in a smooth or continuous way. Coherence is based on the principle of locality, whereby "nearby" things do have the same or similar characteristics. In the field of computer science and computer graphics coherence properties have been exploited in a variety of different methods and techniques. With coherence processing of data can be accelerated considerably, *e.g.*, hidden line/hidden surface removal.

Data can be stored more efficiently by eliminating redundancies due to coherence, *e.g.*, run length encoding of raster images. Some approximate or incremental techniques are only useful in coherent situations, *e.g.*, Newton iteration to determine the roots of a polynomial. In a coherent situation incremental methods can be designed that process an item by reusing the results obtained for a similar (coherent) item. Exploiting coherence leads to algorithms based on an incremental application of very simple operations that in many cases can use "integer logic" to replace time-consuming floating-point arithmetic.

Types of Coherence

In this section a survey of different types of coherence is given. Some of these types of coherence are not mutually exclusive and they are overlapping in the sense that in certain situations they can be used interchangeably.

1. **Scan-line coherence:** Scan-line algorithms for visible surface determination have been one of the earliest examples in computer graphics that make use of coherence properties. With scan-line algorithms an image is generated sequentially, one scan line at a time. The fact that there are relatively few changes from one scan line to the next and almost the same objects (*e.g.*, polygons) are visible on consecutive scan-lines is called scan-line coherence. Therefore a scan-line is processed by only updating the information of the previous scan-line.
2. **Span coherence:** Coherence does not only exist between consecutive scan lines but is usually present within a single scan-line too. Spans are portions of a scan-line with some constant property, *e.g.*, the same object is visible over a given span. All scan line based algorithms (*e.g.*, hidden surface removal, polygon filling) do exploit span coherence, which is given between adjacent spans and within a single span as well.
3. **Depth coherence:** Depth coherence expresses the fact that the depth or distance to the viewer at some surface point changes gradually. Adjacent parts of a surface are usually close in depth. Therefore the depth at some surface point can be efficiently calculated by incrementally updating the depth information of an already processed nearby surface point. Furthermore the depth ordering of surfaces at one pixel is likely to be similar to the ordering at adjacent pixels. An incremental update operation is again feasible.

NOTES

4. **Area coherence:** Area coherence follows from image coherence and is given if adjacent pixels of a raster image do have the same or similar colour or intensity values. This situation arises if, for example, a group of pixels is covered by the same object or the same visible surface. In area coherent portions of an image the calculation of one pixel value allows for the calculation of values of nearby pixels with significantly less computation. Warnock's area subdivision algorithm for visible surface determination of a scene consisting of polygons is a prominent example of a method that makes use of area coherence.
5. **Object coherence:** Object coherence is based on some known relationships between objects or between parts of the same object. Objects may be disjoint, may be closely clustered or may consist of collections of low-level geometry, which usually are connected, smooth and bounded (*e.g.*, polygons, surfaces). Local neighborhoods of space are likely to be occupied by the same objects. Depending on the given relations, manipulations like clipping, sorting, comparison, or intersection can first be performed at the object level to reduce the complexity and volume of calculations on low-level geometry.
6. **Spatial coherence:** Spatial coherence describes spatial homogeneities. These are a consequence of constant or slow varying relationships in the spatial arrangement of objects or data. Volume data like flow fields of a viscous fluid often exhibit a high degree of homogeneity. Considering the spatial arrangement of objects, spatial coherence is somewhat similar to object coherence.
7. **Temporal coherence:** Temporal coherence occurs whenever a dynamic environment changes smoothly over time, *e.g.*, small viewpoint and/or object movement (object space temporal coherence). Discretization of a temporal coherent situation allows the calculation of the relevant information by using results obtained for previous time steps. Temporal coherence has been used in a variety of methods for accelerating the calculation of animation sequences. Often only restricted situations with high temporal coherence are handled by these techniques. Restrictions can be one of the following: Fixed viewpoint or restricted camera movement, fixed light sources, restricted class of objects like planar or convex objects etc.
8. **Frame coherence:** Frame coherence denotes the fact that successive frames of an animation sequence or video sequence are likely to be very similar if the difference in time is small, *i.e.*, the projection of an environment tends to change continuously over time. Frame coherence is an immediate consequence of temporal coherence and object coherence and may be considered to be an image space temporal coherence. Frame coherence allows an efficient calculation and storage of video sequences.
9. **Image coherence:** Image coherence is the view-dependent analogue to object coherence. It results from the transfer of object coherence properties to the image plane by well-behaved projections, *e.g.*, orthogonal or perspective projections. In the 2-D image plane there is at least the same degree of connectedness and smoothness as among the original 3-D objects. Local constancy of object space translates to local constancy in image space with only gradual changes. Some additional coherence is due to the projection technique itself. A given part of the image plane may be influenced by only a small subset of the items in object space.
10. **Ray coherence:** Tracing rays through an object environment has been used extensively in computer graphics for image rendering purposes. The classical ray-tracing approach has been as follows: for each pixel of an image a ray is cast from the eye point into object space to determine the intersection point of the first visible object. To account for global illumination effects an approximation to the light distribution

NOTES

in the object scene is done by recursively casting further rays from this point of intersection. Depending on the surface properties of the intersected object, rays are cast in the direction of reflection and refraction.

Shadow calculation is done by tracing rays from the point of intersection to the various light sources. Therefore, a ray tree is constructed to calculate the colour information for each pixel. Due to the spatial coherence and object coherence of the scene, similar rays, *i.e.*, rays with similar origin and direction, often have almost the same behavior. Similar rays often intersect the same object; the points of intersection are close together and so on. This property is called ray coherence. The similarity between ray trees is analogously called ray tree coherence. One of the major drawbacks of ray tracing has been its excessive computational cost. An extensive amount of research has been done to accelerate ray tracing by taking advantage of the various types of coherence that are inherent in the raytracing technique.

11. **Other types of coherence:** The following less frequently used types of coherence are listed only for reasons of completeness: cell coherence, cluster coherence, cube coherence, data coherence, edge coherence, external coherence, face coherence, frustum coherence, geometric coherence, hierarchical coherence, implied edge coherence, intervisibility coherence, invisibility coherence, objective coherence, patch-to-patch coherence, path coherence, predictive coherence, screen-area coherence, shadow coherence, space coherence, surface coherence and volume coherence.

Data Structures for Coherence

Some specific data structures have been developed that are well suited to exploit coherence properties. Only few examples are shortly mentioned here.

- A grid is a subdivision of 3-D space into regular cubical elements to exploit object and hierarchical coherence.
- An octree describes an adaptive hierarchical subdivision of 3-D space.
- The BSP-tree (Binary Space Partitioning) is a hierarchical data structure designed for storing polygonal objects.
- CSG (Constructive Solid Geometry) defines an object through a binary tree. A CSG tree usually contains an efficient bounding volume hierarchy (hierarchical coherence).
- Many data structures utilize subdivisions to reduce the manipulation cost from linear to logarithmic, such as interval trees, segment trees and range trees.

Spatial Partitioning

In this technique, subdivision rule is applied to break down a large problem into a number of smaller ones. In this objects and their projections are assigned to spatically coherent groups as a preprocessing step. This partitioning speed up the process of determining which object intersects with a projector. Because now it is necessary to test only the objects lying within the partitions which intersect with projector.

5.4 BACK FACE REMOVAL

A simple object space algorithm is back face removal (or back face cull) where no faces on the back of the object are displayed. Since in general about half of the faces of objects are back faces this algorithm will remove about half of the total polygons in the image.

Graphically it is represented as follows:

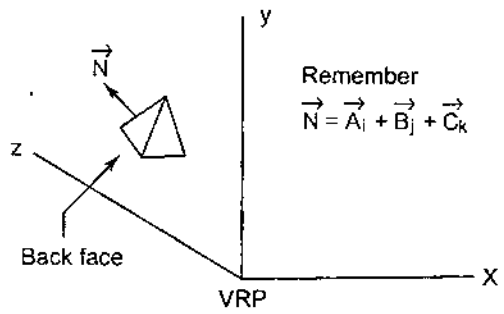


Fig. 5.2(a)

Look at a left handed viewing system

If C > 0 then N is in the direction of +z, and it faces away from the VRP, therefore, if C > 0 then it is a back face (for a left handed system).

For a right handed system it is graphically represented as follows :

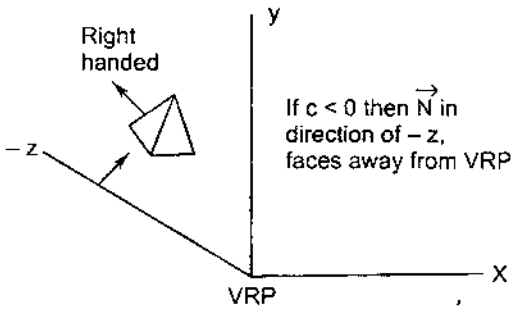


Fig. 5.2(b)

It is just the opposite condition for a right handed viewing system. Therefore, if C < 0 then a back face in a right-handed system.

So the algorithm (for left-handed system)

1. Compute N for every face of Object
2. If C (z component) > 0 then a back face and don't draw

Note that we must be able to identify the polygons for the object.

This simple method is only correct for an orthographic projection. For a perspective projection, it is a little more complicated. For the scene below, the visible surfaces are different for orthographic or perspective projection. The sides will be invisible for an orthographic projection, but not for a perspective projection.

For a perspective projection, we must determine if the Center of Projection (COP) is inside or outside of the planes of the polygons of the object. If the COP is inside then that plane is not visible, if the COP is outside then it is visible.

There are two methods to compute this

1. Put COP into plane equation and determine if inside or outside.

   NOTE ➤ Must compute plane equation before the perspective transformation.
2. If the angle between the plane normal (N) and the vector from any point on the plane to the COP, V is > 90° (N·V < 0) then that plane is not visible.

NOTES

NOTES

Graphically it is represented as follows:

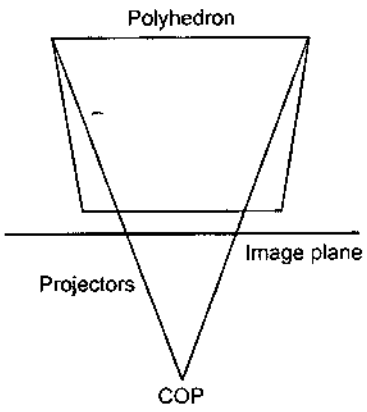


Fig. 5.2(c)

Limitations on back face removal algorithm

1. It can only be used on solid objects modeled as a polygon mesh. This is the most general modeling construct for scan-line graphics systems. Even if objects are defined in a different manner, *e.g.*, by parametric cubic patches or implicit equations, the renderer usually converts everything to polygons to render.
2. It works fine for convex polyhedra but not necessarily for concave polyhedra as shown below in the example of a partially hidden face, that will not be eliminated by back face removal.

Even if we use another algorithm for visible surface determination, the back face cull is a good pre-processing step., once it removes about half of the polygons in the scene. Also, for color shading we must compute the normal for all of the polygons anyway.

5.5 Z-BUFFER ALGORITHM

The basic idea is to test the z-depth of each surface to determine the closest (visible) surface. Declare an array z buffer(*x*, *y*) with one entry for each pixel position. Initialize the array to the maximum depth. So initialize all values to maximum depth. Then the algorithm is as follows:

```
for each polygon P
  for each pixel (x, y) in P
    compute z_depth at x, y
    if z_depth < z_buffer (x, y) then
       set_pixel (x, y, color)
       z_buffer (x, y) = z_depth
```

The easiest way to achieve hidden-surface removal is to use the depth buffer (sometimes called a z-buffer). A depth buffer works by associating a depth, or distance from the viewpoint, with each pixel on the window. Initially, the depth values for all pixels are set to the largest possible distance, and then the objects in the scene are drawn in any order. Graphical calculations in hardware or software convert each surface that's drawn to a set of pixels on the window where the surface will appear if it isn't obscured by something else. In addition, the distance from the eye is computed. With depth buffering enabled, before each pixel is drawn, a comparison is done with the depth value already stored at the pixel.

If the new pixel is closer to the eye than what's there, the new pixel's colour and depth values replace those that are currently written into the pixel. If the new pixel's depth is greater than what's currently there, the new pixel would be obscured, and the colour

and depth information for the incoming pixel is discarded. Since information is discarded rather than used for drawing, hidden-surface removal can increase your performance.

5.6 SCAN-LINE ALGORITHM

NOTES

The scan-line algorithm is another image-space algorithm. It processes the image one scan-line at a time rather than one pixel at a time. By using area coherence of the polygon, the processing efficiency is improved over the pixel oriented method.

Using an active edge table, the scan-line algorithm keeps track of where the projection beam is at any given time during the scan-line sweep. When it enters the projection of a polygon, an IN flag goes on, and the beam switches from the background colour to the colour of the polygon. After the beam leaves the polygon's edge, the colour switches back to background colour. To this point, no depth information need be calculated at all. However, when the scan-line beam finds itself in two or more polygons, it becomes necessary to perform a z-depth sort and select the colour of the nearest polygon as the painting colour.

Accurate bookkeeping is very important for the scan-line algorithm. We assume the scene is defined by at least a polygon table containing the (A, B, C, D) coefficients of the plane of each polygon, intensity/colour information, and pointers to an edge table specifying the bounding lines of the polygon. The edge table contains the coordinates of the two end points, pointers to the polygon table to indicate which polygons the edge bounds, and the inverse slope of the *x-y* projection of the line for use with scan-line algorithms. In addition to these two standard data structures, the scan-line algorithm requires an active edge list that keeps track of which edges a given scan-line intersects during its sweep. The active edge list should be sorted in order of increasing x at the point of intersection with the scan line. The active edge list is dynamic, growing and shrinking as the scan line progresses down the screen.

In the following figure scan-line S_1 must deal only with the left-hand object. S_2 must plot both objects, but there is no depth conflict. S_3 must resolve the relative *z*-depth of both objects in the region between edge E_5 and E_3. The right-hand object appears closer.

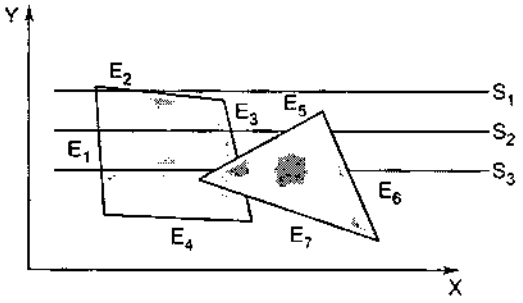


Fig. 5.3

The active edge list for scan line S_1 contains edges E_1 and E_2. From the left edge of the viewport to edge E_1, the beam paints the background colour. At edge E_1, the IN flag goes up for the left-hand polygon, and the beam switches to its colour until it crosses edge E_2, at which point the IN flag goes down and the colour returns to background.

For scan-line S_2, the active edge list contains E_1, E_3, E_5, and E_6. The IN flag goes up and down twice in sequence during this scan. Each time it goes up pointers identify the appropriate polygon and look up the colour to use in painting the polygon.

For scan line S_3, the active edge list contains the same edges as for S_2, but the order is altered, namely E_1, E_5, E_3, E_6. Now the question of relative z-depth first appears. The IN flag goes up once when we cross E_1 and again when we cross E_5, indicating that the

projector is piercing two polygons. Now the coefficients of each plane and the (x, y) of the E_5 edge are used to compute the depth of both planes. In the example shown the z-depth of the right-hand plane was smaller, indicating it is closer to the screen. Therefore the painting colour switches to the right-hand polygon colour which it keeps until edge E_6.

Note that the technique is readily extended to three or more overlapping polygons and that the relative depths of overlapping polygons must be calculated only when the IN flag goes up for a new polygon. Since this occurrence is far less frequent than the number of pixels per scan-line, the scan-line algorithm is more computationally efficient than the z-buffer algorithm.

NOTES

The scan-line hidden surface removal algorithm can be summarized as:

1. Establish the necessary data structures.
   (*a*) Polygon table with coefficients, colour, and edge pointers.
   (*b*) Edge table with line end points, inverse slope, and polygon pointers.
   (*c*) Active edge list, sorted in order of increasing *x*.
   (*d*) An IN flag for each polygon.
2. Repeat for all scan lines:
   (*a*) Update active edge list by sorting edge table against scan line *y* value.
   (*b*) Scan across, using background colour, until an IN flag goes on.
   (*c*) When 1 polygon flag is on for surface P, enter intensity (colour) into refresh buffer.
   (*d*) When 2 or more surface flags are on, do depth sort and use intensity I_n for surface *n* with minimum z-depth.
   (*e*) Use coherence of planes to repeat for next scan-line.

The scan-line algorithm for hidden surface removal is well designed to take advantage of the area coherence of polygons. As long as the active edge list remains constant from one scan to the next, the relative structure and orientation of the polygons painted during that scan does not change. This means that we can "remember" the relative position of overlapping polygons and need not recompute the z-depth when two or more IN flags go on. By taking advantage of this coherence we save a great deal of computation.

5.7 PAINTER'S ALGORITHM

The idea behind the Painter's algorithm is to draw polygons far away from the eye first, followed by drawing those that are close to the eye. Hidden surfaces will be written over in the image as the surfaces that obscure them are drawn.

The concept is to map the objects of our scene from the world model to the screen somewhat like an artist creating an oil painting. First she paints the entire canvas with a background colour. Next, she adds the more distant objects such as mountains, fields, and trees. Finally, she creates the foreground with "near" objects to complete the painting. Our approach will be identical. First we sort the polygons according to their z-depth and then paint them to the screen, starting with the far faces and finishing with the near faces.

The algorithm initially sorts the faces in the object into back to front order. The faces are then scan converted in this order onto the screen. Thus a face near the front will obscure a face at the back by overwriting it at any points where their projections overlap. This accomplishes hidden-surface removal without any complex intersection calculations between the two projected faces.

The algorithm is a hybrid algorithm in that it sorts in object space and does the final rendering in image space.

The basic algorithm is as follows:

1. Sort all polygons in ascending order of maximum z-values.
2. Resolve any ambiguities in this ordering.
3. Scan convert each polygon in the order generated by steps (1) and (2).

The necessity for step (2) can be seen in the simple case shown in following figure.

NOTES

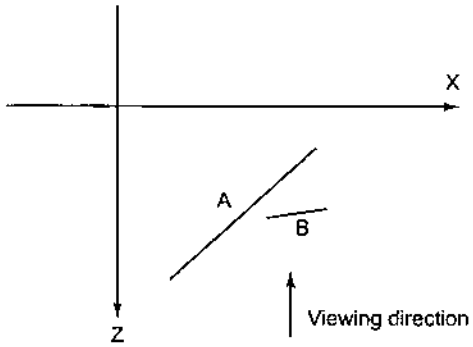


Fig. 5.4 Ambiguities in the painter's algorithm

B precedes A in order of maximum z but A should precede B in writing order. At step (2) the ordering produced by (1) must be confirmed. This is done by making more precise comparisons between polygons whose z-extents overlap. Assume that polygon P is currently at the head of the sorted list, before scan converting it to the frame-buffer it is tested against each polygon Q whose z-extent overlaps that of P. The following tests of increasing complexity are then carried out:

1. If the *x*-extents of P and Q do not overlap then the polygons do not overlap, hence their ordering is immaterial.
2. If the *y*-extents of P and Q do not overlap then the polygons do not overlap, hence their ordering is immaterial.
3. If P is wholly on the far away side of Q then P is written before Q.
4. If Q is wholly on the viewing side of P then P is written before Q.
5. If the projections of P and Q do not overlap then the order P and Q in the list is immaterial.

If any of these tests are satisfied then the ordering of P and Q need not be changed. However if all five tests fail then it is assumed that P obscures Q and they are interchanged in the list. To avoid looping in the case where P and Q inter-penetrate Q must be marked as having been moved in the list. When polygon Q which has been marked fails all tests again when tested against P then it must be split into two polygons and each of these polygons treated separately. Q is split in the plane of P. Often the last test will not be done because it can be very complex for general polygons.

5.8 WARNOCK'S AREA SUBDIVISION ALGORITHM

John Warnock proposed an elegant divide-and-conquer hidden surface algorithm. The algorithm relies on the area coherence of polygons to resolve the visibility of many polygons in image space. Depth sorting is simplified and performed only in those cases involving image-space overlap. Warnock's algorithm classifies polygons with respect to the current viewing window into trivial or non-trivial cases. Trivial cases are easily handled. For nontrivial cases, the current viewing window is recursively divided into four equal subwindows, each of which is then used for reclassifying remaining

polygons. This recursive procedure is continued until all polygons are trivially classified or until the current window reaches the pixel resolution of the screen. At that point the algorithm reverts to a simple z-depth sort of the intersected polygons, and the pixel colour becomes that of the polygon closest to the viewing screen.

All polygons are readily classified with respect to the current window into the four categories

which are as follows:

NOTES

1. **Surrounding Polygon:** One that completely encloses the shaded area of interest. Graphically it is represented as follows:

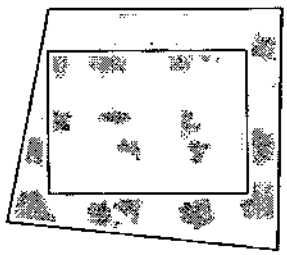


Fig. 5.5(a) Surrounding

2. **Overlapping or Intersecting Polygon:** One that is partly inside and partly outside the area.
   Graphically it is represented as follows:

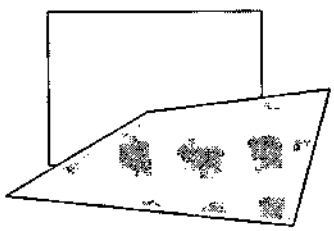


Fig 5.5(b) Interlacing or overlapping

3. **Inside or Contained Polygon:** One that is completely inside the area.
   Graphically it is represented as follows:

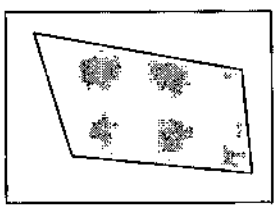


Fig 5.5(c) Inside or contained

4. **Outside or Disjoint Polygon:** One that is completely outside the area.
   Graphically it is represented as follows:

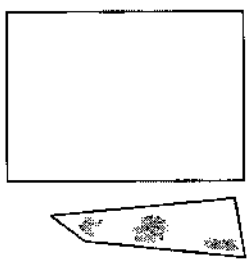


Fig. 5.5(d) Outside or disjoint

The classification scheme is used to identify certain trivial cases that are easily handled. These "easy choice tests" and the resulting actions include:

1. For polygons outside from the window, set the colour/intensity of the window equal to the background colour.
2. There is only one inside or intersecting polygon. Fill the window area with the background colour then render the polygon.
3. There is only one surrounding polygon. Fill the window with the polygon's colour.
4. If more than one polygon intersects, is inside, or surrounds, and at least one is a surrounding polygon.
   (a) Is one surrounding polygon, P, in front of all others? If so, paint window with the colour of P. The test is: Calculate the z-depths for each polygon plane at the corners of the current window. If all four z-depths of the P plane are all smaller than any z-depths of other polygons in the window, then P is in front.

NOTES

If the easy choice tests do not classify the polygon configuration into one of these four trivial action cases, the algorithm recurs by dividing the current window into four equal subwindows. Rather than revert to the complex geometrical tests of the Painter's algorithm, Warnock's algorithm simply makes the easy choices and invokes recursion for non-trivial cases.

A noteworthy feature of Warnock's algorithm concerns how the divide-and-conquer area subdivision preserves area coherence. That is, all polygons classified as surrounding and outside retain this classification with respect to all subwindows generated by recursion. This aspect of the algorithm is the basis for its efficiency. The algorithm may be classified as a radix four quick sort. Windows of 1024 × 1024 pixels may be resolved to the single pixel level with only ten recursive calls of the algorithm.

While the original Warnock algorithm had the advantages of elegance and simplicity, the performance of the area subdivision technique can be improved with alternative subdivision strategies. Some of these include:

1. Divide the area using an enclosed polygon vertex to set the dividing boundary.
2. Sort polygons by minimum z and use the front polygon as the window boundary.

5.9 BINARY SPACE PARTITIONING

Binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyperplanes. This subdivision gives rise to a representation of the scene by means of a tree data structure known as a **BSP tree**. In simpler words, it is a method of breaking up intricately shaped polygons into convex sets, or smaller polygons consisting entirely of non-reflex angles (angles smaller than 180°). For a more general description of space partitioning, see space partitioning.

Originally, this approach was proposed in 3-D computer graphics to increase the rendering efficiency. Some other applications include performing geometrical operations with shapes (constructive solid geometry) in CAD, collision detection in robotics and 3-D computer games, and other computer applications that involve handling of complex spatial scenes.

In computer graphics it is desirable that the drawing of a scene be both correct and quick. A simple way to draw a scene correctly is the painter's algorithm: draw it from back to front, painting the background over with each closer object. However, that approach is

NOTES

quite limited since time is wasted drawing objects that will be overdrawn later, and not all objects will be drawn correctly.

Z-buffering can ensure that scenes are drawn correctly and eliminate the ordering step of the painter's algorithm, but it is expensive in terms of memory use. BSP trees will split up objects so that the painter's algorithm will draw them correctly without need of a Z-buffer and eliminate the need to sort the objects as a simple tree traversal will yield them in the correct order. It also serves as base for other algorithms, such as visibility lists, which seek to reduce overdraw.

The downside is the requirement for a time consuming pre-processing of the scene, which makes it difficult and inefficient to directly implement moving objects into a BSP tree. This is often overcome by using the BSP tree together with a Z-buffer, and using the Z-buffer to correctly merge movable objects such as doors and monsters onto the background scene.

BSP trees are often used by 3-D computer games, particularly first-person shooters and those with indoor environments. Probably the earliest game to use a BSP data structure was Doom. Other uses include ray tracing and collision detection.

Generation

Binary space partitioning is a generic process of recursively dividing a scene into two until they satisfy one or more requirements, the specific method of division varying depending on its final purpose. For instance, in a BSP tree used for collision detection the original object would be partitioned until each part becomes simple enough to be individually tested, and in rendering it's desirable that each part be convex so that the painter's algorithm can be used.

The final number of objects will inevitably increase since lines or faces that cross the partitioning plane must be split into two, and it is also desirable that the final tree remains reasonably balanced. Therefore the algorithm for correctly and efficiently creating a good BSP tree is the most difficult part of an implementation. In 3-D space, planes are used to partition and split an object's faces; in 2-D space lines split an object's segments.

The following picture illustrates the process of partitioning an irregular polygon into a series of convex ones. Notice how each step produces polygons with fewer segments until arriving at G and F, which are convex and require no further partitioning. In this particular case, the partitioning line was picked between existing vertices of the polygon and intersected none of its segments. If the partitioning line intersects a segment, or face in a 3-D model, the offending segment(s) or face(s) have to be split into two at the line/plane because each resulting partition must be a full, independent object.

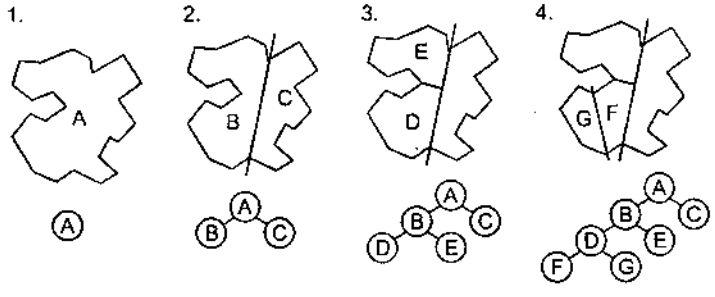


Fig. 5.6

The explanation of the above figure is as follows:

1. A is the root of the tree and the entire polygon
2. A is split into B and C
3. B is split into D and E
4. D is split into F and G, which are convex and hence become leaves on the tree.

BSP Tree

NOTES

A Binary Space Partitioning (BSP) tree represents a recursive, hierarchical partitioning, or subdivision, of *n*-dimensional space into convex sub-spaces. BSP tree construction is a process which takes a subspace and partitions it by any hyperplane that intersects the interior of that subspace. The result is two new sub-spaces that can be further partitioned by recursive application of the method. A "hyperplane" in *n*-dimensional space is an *n*-1 dimensional object which can be used to divide the space into two half-spaces. For example, in three-dimensional space, the "hyperplane" is a plane. In two dimensional space, a line is used. BSP trees are extremely versatile, because they are powerful sorting and classification structures. They have uses ranging from hidden surface removal and ray tracing hierarchies to solid modeling and robot motion planning. BSP trees are closely related to Quadtrees and Octrees. Quadtrees and Octrees are space partitioning trees which recursively divide sub-spaces into four and eight new sub-spaces, respectively. A BSP Tree can be used to simulate both of these structures.

BSP Tree Construction

Given a set of polygons in three-dimensional space, we want to build a BSP tree which contains all of the polygons. For now, we will ignore the question of how the resulting tree is going to be used. The algorithm to build a BSP tree is very simple:

1. Select a partition plane.
2. Partition the set of polygons with the plane.
3. Recurse with each of the two new sets.

The choice of partition plane depends on how the tree will be used, and what sort of efficiency criteria you have for the construction. For some purposes, it is appropriate to choose the partition plane from the input set of polygons. Other applications may benefit more from axis aligned orthogonal partitions. In any case, you want to evaluate how your choice will affect the results. It is desirable to have a balanced tree, where each leaf contains roughly the same number of polygons. However, there is some cost in achieving this. If a polygon happens to span the partition plane, it will be split into two or more pieces. A poor choice of the partition plane can result in many such splits, and a marked increase in the number of polygons. Usually there will be some trade off between a well balanced tree and a large number of splits. Partitioning a set of polygons with a plane is done by classifying each member of the set with respect to the plane. If a polygon lies entirely to one side or the other of the plane, then it is not modified, and is added to the partition set for the side that it is on. If a polygon spans the plane, it is split into two or more pieces and the resulting parts are added to the sets associated with either side as appropriate. The decision to terminate tree construction is, again, a matter of the specific application. Some methods terminate when the number of polygons in a leaf node is below a maximum value. Other methods continue until every polygon is placed in an internal node. Another criteria is a maximum tree depth.

Partitioning a polygon with a plane is a matter of determining which side of the plane the polygon is on. This is referred to as a front/back test, and is performed by testing each point in the polygon against the plane. If all of the points lie to one side of the plane, then the entire polygon is on that side and does not need to be split.

NOTES

If some points lie on both sides of the plane, then the polygon is split into two or more pieces. The basic algorithm is to loop across all the edges of the polygon and find those for which one vertex is on each side of the partition plane. The intersection points of these edges and the plane are computed, and those points are used as new vertices for the resulting pieces.

Classifying a point with respect to a plane is done by passing the (x, y, z) values of the point into the plane equation, $ax + by + cz + d = 0$. The result of this operation is the distance from the plane to the point along the plane's normal vector. It will be positive if the point is on the side of the plane pointed to by the normal vector, negative otherwise. If the result is 0, the point is on the plane. For those not familiar with the plane equation, the values a, b, and c are the coordinate values of the normal vector. The value of d can be calculated by substituting a point known to be on the plane for x, y, and z into the plane equation.

Convex polygons are generally easier to deal with in BSP tree construction than concave ones, because splitting them with a plane always results in exactly two convex pieces. Furthermore, the algorithm for splitting convex polygons is straightforward and robust.

Hidden Surface Removal Using BSP

Probably the most common application of BSP trees is hidden surface removal in three dimensions. BSP trees provide an elegant, efficient method for sorting polygons via a depth first tree walk. This fact can be exploited in a back to front "painter's algorithm" approach to the visible surface problem, or a front to back scan-line approach. BSP trees are well suited to interactive display of static (not moving) geometry because the tree can be constructed during a preprocessing stage. Then the display from any arbitrary viewpoint can be done in linear time.

One reason that BSP trees are so elegant for the painter's algorithm is that the splitting of difficult polygons is an automatic part of tree construction. To draw the contents of the tree, perform a back to front tree traversal. Begin at the root node and classify the eye point with respect to its partition plane. Draw the subtree at the far child from the eye, then draw the polygons in this node, then draw the near subtree. Repeat this procedure recursively for each subtree.

When building a BSP tree specifically for hidden surface removal, the partition planes are usually chosen from the input polygon set. However, any arbitrary plane can be used if there are no intersecting or concave polygons. If the eye point is classified as being on the partition plane, the drawing order is unclear. This is not a problem if the rendering routine is smart enough to not draw polygons that are not within the viewing frustum. It is possible to substantially improve the algorithm by including the viewing direction vector in the computation. You can determine that entire subtrees are behind the viewer by comparing the view vector to the partition plane normal vector.

5.10 COMPARISON OF ALGORITHMS

The following are the comparisons between back face removal algorithm, Painter's algorithm, Warnock algorithm and Z-Buffer algorithm.

- Backface removal algorithm is fast, but insufficient by itself.
- Painter's algorithm is device independent.

NOTES

5.11 CURVE GENERATION

The term *computer graphics* was often associated with creation of realistic scenes and animated images. With the advent of both computers and applied mathematics more ambitious applications of computer graphics were sought in the last few decades. Today, one of the most important areas where computer graphics plays a central role is *Computer Aided Design* (CAD) and *Computer Aided Manufacture* (CAM). Both CAD and CAM are extensively used in a large number of areas, including aerospace, automotive engineering, marine engineering, civil engineering, and electronic engineering. The successful applications of computer graphics in engineering is largely due to the progress of *Computer Aided Geometric Design* (CAGD) which provides the mathematical basis for describing and processing geometric shapes and data. The term *geometric modeling* (which includes *curve modeling*, *surface modeling*, and *solid modeling*) is often used as the synonym of computer aided geometric design, although some authors argue that geometric modeling means the building up of computer representations of complex shapes from representations of similar components.

The most promising description method of geometric shape is the parametric curves and surfaces. The theory of parametric curves and surfaces are well understood in algebraic geometry and differential geometry. However, their possibilities and advantages for representing geometric entities in a CAD environment were not known until the late 1950s. The major breakthroughs in CAGD were undoubtedly the theory of Ferguson curves and patches, Coons patches, Bézier curves and surfaces, later combined with B-spline methods. Today, Bézier and B-spline representations of curves and surfaces have been the industrial standard. In this chapter, we shall briefly review different curve representation methods and tell you why the parametric representation of curves is of most importance. Then, we shall discuss the mathematics on Bézier and B-spline curves, which is the foundation of surface and solid modeling. Curves are one of the most essential objects to create high-resolution graphics. While using many small polylines allows creating graphics that appear smooth at fixed resolutions, they do not preserve smoothness when scaled and also require a tremendous amount of storage for any high-resolution image. Curves can be stored much easier, can be scaled to any resolution without losing smoothness, and most importantly provide a much easier way to specify real-world objects.

All of the popular curves used in graphics are specified by parametric equations. Instead of specifying a function of the form $y = f(x)$, the equations $y = fy(u)$ and $x = fx(u)$ are used. Using parametric equations allows curves that can double back and cross themselves, which are impossible to specify in a single equation in the $y = f(x)$ case. Parametric equations are also easier to evaluate: changing u results in moving a fixed distance along the curve, while in the traditional equation form much work is needed to determine whether to step through x or y, and determining how large a step to take based on the slope.

5.12 AFFINE INVARIANCE

A curve is said to be Affine Invariant if the coordinate system it is represented in can change without affecting the relative geometry of the curve. This can be seen by the fact that the geometry of the curve remains consistant when the curve is rotated, scaled, or translated.

5.13 CONVEX COMBINATION

Given a set of points $P_0, P_1, \ldots, P_n$, we can form an affine combination of these points by selecting constants $\alpha_0, \alpha_1, \ldots, \alpha_n$, $\alpha_1, \ldots, \alpha_n$, (where $\alpha_0, \alpha_1, \ldots, \alpha_n = 1$) and write

$$P = \alpha_0 P_0 + \alpha_1 P_1 + \ldots\ldots\ldots + \alpha_n P_n$$

If each α_i is such that $0 <= \alpha_i <= 1$, then the point P is called a *convex combination* of the points $P_0, P_1, \ldots\ldots P_n$.

Example : Point on a Line Segment

To give a simple example of this, consider two points P_0 and P_1. Any point P on the line passing through these two points can be written as follows:

$$P = (1 - t)P_0 + tP_1$$

NOTES

or equivalently

$$P = \alpha_0 P_0 + \alpha_1 P_1$$

Where $\alpha_0 + \alpha_1 = 1$. The points $Q = 1/3P_0 + 2/3P_1$ and $R = -2/5P_0 + 7/5P_1$ in the following figure are affine combinations of P_0 and P_1.

Graphically it is represented as follows:

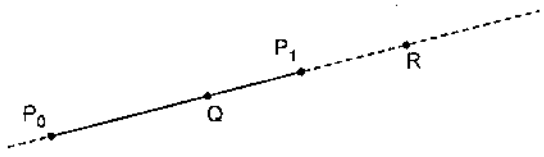


Fig. 5.7

The point Q is a convex combination since $0 <= \alpha_0, \alpha_1 <= 1$. Any point on the line segment joining P_0 and P_1 and can be written as a convex combination.

5.14 CONVEX SET

Given any set of points, we say that the set is a *convex set* if given any points $P_0, P_1, \ldots\ldots P_n$ in the set, any convex combination of these points is also in the set.

The following figure illustrates both a convex set (on the left) and a non-convex set (on the right).

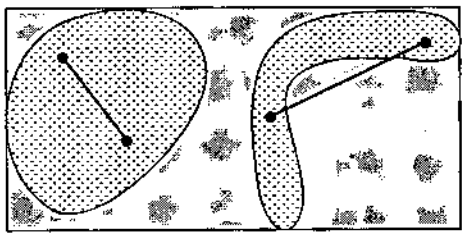


Fig. 5.8

Since any convex combination of points from a convex set must lie in the set, then certainly the straight line joining any two points of the set must also be completely in the set. This concept is actually quite intuitive, in that if one can draw a straight line between two points of the set *that is not completely contained within the set*, the set is not convex.

5.15 CONVEX HULL

Given a set of points $P_0, P_1, \ldots\ldots P_n$. The set of all points P that can be written as convex combinations of $P_0, P_1, \ldots\ldots P_n$ is called the *convex hull* of the set. It is easy to see that this convex hull is necessarily a convex set-but, it turns out that it is the smallest convex set that contains $P_0, P_1, \ldots\ldots P_n$. (If there were a convex set C smaller than the convex hull that contained the points, then we could find a point Q in the convex hull, but not in the set C. But since each of the P_i is in both sets, and the point Q is a convex combination of the P_i, it must also be in the convex hull as well as in C.) The following figure illustrates the convex hull of a set of six points:

We note that one of the six points does not contribute to the boundary of the convex hull. If one looked closely at the coordinates of P_2, one would find that this point could be written as a convex combination of the other five.

Preliminary Definition: A subset S of the plane is called *convex* if and only if for every pair of points *p*, *q* in S, the line segment *pq* is completely contained in S.

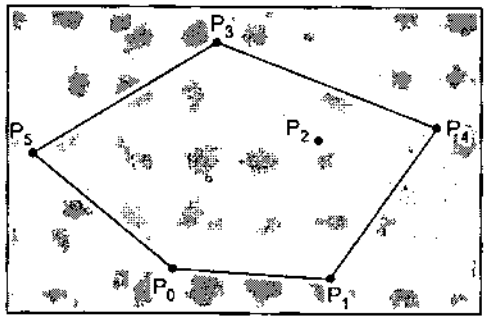


Fig. 5.9

Convex Hull Definition #1: The convex hull CH(S) of a set S is the smallest convex set that contains S.

Convex Hull Definition #2: The convex hull CH(S) of a set S is the intersection of all convex sets that contain S.

Convex Hull Definition #3: The convex hull CH(S) of a set S is the unique convex polygon which contains S and all of whose vertices are points from S.

5.16 ORDER OF A CURVE

The order of a curve deals with the number of degrees of freedom that are necessary to uniquely define that curve. In general the order of a curve equals the degrees of freedom of that curve. The "Degree" of a curve (dealing with the polynomial degree of the functions that define the curve) is NOT related to the "degrees of freedom" of the curve.

We can say in our language that the degree = (Order-1)'

The following table shows the relationship between order, degree and the name.

Order	Degree	Name
1	0	Constant
2	1	Linear
3	2	Quadratic
4	3	Cubic
5	4	Quartic

Example:

For B-Splines, the order of the curve specifies the number of control points that affect the parametric point on the curve. This can be seen by looking at the convex hull that surrounds that point. Notice that for an order 2 curve, the convex hull is actually a straight line. For an order 3 curve, the convex hull is a triangle, and for an order 4 curve, the convex hull can be a quadrilateral.

5.17 DEGREES OF FREEDOM

A "degree of freedom" is one piece of information used to describe a curve. The more degrees of freedom a curve has, the more complex it can be.

Example:

How much information is needed to describe a straight line (lines are simple curves) ?

The answer is two pieces of information, *i.e.*,: two points. Therefore a line has two degrees of freedom.

5.18 LEGRANGE INTERPOLATED CURVES

NOTES

The goal of LeGrange interpolation is to create a smooth curve that passes through an ordered group of points. When used in this fashion, these points are called the control points. The general idea, as with all curve algorithms, is to use the control points to generate parametric equations. To draw the curve, all that is then needed is to step through u at some small amount, drawing straight lines between the calculated points. The smaller the step size the more smooth the curve will appear. The step size can be calculated based on the amount of pixels available in the output image.

To do LeGrange interpolation, we wish to specify a curve that will pass through any number of control points. The curve's function can be constructed as a sum of terms, one for each control point.

```
fx(u) = sum from i = 1 to n of x[i] B[i] (u)
fy(u) = sum from i = 1 to n of y[i] B[i] (u)
```

Each function B|i|(u) specifies how much the ith control point effects the position of the curve. This can be thought of as a function specifying how much the ith control point draws the curve towards it. The name for these functions are blending functions. If for some control point i the value of this function is 1 and for every other control point the value of this function is 0, the curve will go through i at that value.

To achieve the goal of LeGrange interpolation, getting a curve that passes through all the specified control points, the blending functions must be defined so that the curve will go through each control point in the proper order. This can be done by creating blending functions where the first control point is passed through (has total control) at $u = -1$, the second control point is passed through at $u = 0$, the third at $u = 1$, and so on. A mathematical expression that does so is:

$u(u-1)(u-2)\ldots[u-(n-2)]$

At $u = -1$ this expression is:

$-1(-2)(-3)\ldots(1-n)$

So dividing the first by the second gives us 1 when $u = -1$, and gives us 0 when u is a whole number greater then -1 and less then or equal to n. Using this method for the other blending functions gives us the same type of behavior for the other control points. This function defines the LeGrange interpolation blending functions.

$$B|i|(u) = \frac{(u+1)(u)(u-1)\ldots[u-(i-3)][u-(i-1)]\ldots[u-(i-2)]}{(i-1)(i-2)(i-3)\ldots(1)(-1)\ldots(i-n)}$$

To draw a curve using this, it is necessary to decide how many control points we will use for the basis of the curve, and then calculate the blending functions B|1|(u) to B|n|(u), where n is the number of control points. The final functions for x, y, and z are then:

```
x = x[1] B[1](u) + x[2] B[2](u) + x[3] B[3](u) + . . . + x[n]
B[n](u)
y = y[1] B[1](u) + y[2] B[2](u) + y[3] B[3](u) + . . . + y[n]
B[n](u)
```

Where x[i] is the x coordinate of the ith control point, and y[i] is the y coordinate of the ith control point.

The number of control points used is most commonly 4. In this case, the calculated curve is particularly good between the second and third control points ($0 <= u <= 1$). Thus to draw a full LeGrange interpolated curve using many control points, it is common to work with sets of four points. First start with the first through fourth control points, and step through u from 0 to 1 in small steps, drawing straight lines between each. When $u = 1$ is reached, switch to using the second through fifth control points. Now step through u from 0 to 1 in small steps again. Repeat this, each time shifting down along the control points in the curve by one point until the end of the curve is reached. At this point all of the curve will have been drawn, except for the first and last segments (the curve between the first and second control points and the curve between the second to last and last control points). These can be dealt with as special cases, calculating from $-1 <= u <= 0$ using the first four control points and $1 <= u <= 2$ using the last four control points.

Some sample LeGrange interpolated curves appear below :

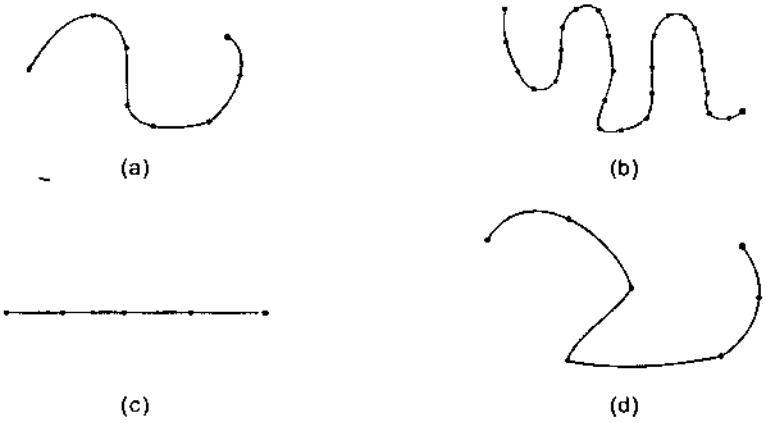


Fig. 5.10

The last two of the above images shows why LeGrange interpolated curves are not usually used in graphics. Although the control points form a straight line, the curve wiggles between the control points. This results because the blending functions sum to 1 at the control points, but do not necessarily do so at fractional values. To eliminate the wiggle we would need to dividing the value of each blending function by the sum of the blending function values before using them. However, there is also another problem. While each section of the curve connects to the next section at a control point, the slopes for the connected curves at the control point may be different. This results in the ability to have corners at control points. This results because while at control points only that control point has control, at any other part of the curve all points have some control. The third problem is that the LeGrange curve goes outside the convex hull of the control points. This means that clipping LeGrange curves must be done with every single line segment drawn, instead of clipping the control points and then just drawing the curve. This adds much computation.

5.19 BEZIER CURVE

Our first question is: why do we need new forms of parametric curves ? An immediate answer is that those parametric curves discussed in the previous unit are not very *geometric*. More precisely, given such a parametric form it is difficult to know the underlying geometry it represents without some further analysis. The coefficients of the

NOTES

equations do not have any geometric meaning, and it is almost impossible to predict the change of shape if one or more coefficients are modified. As a result, designing a curve that follows certain outline is very difficult.

In practice, designers or users usually do not care about the underlying mathematics (and equations, of course). They are more or less focusing on getting their jobs done. To do so, a system that supports users to design curves must be

1. **Intuitive**: We expect that every step and every algorithm will have an intuitive and geometric interpretation.
2. **Flexible**: The system should provide the users with more control for designing and editing the shape of a curve. The way of creating and editing a curve should be easy, intuitive and geometric rather than by manipulating equations.
3. **Unified**: The way of representing, creating and editing different types of curves (*e.g.*, lines, conic sections and cubic curves) must be the same. That is, it does not require different techniques for manipulating different curves (*i.e.*, conics and cubics).
4. **Invariant**: The represented curve will not change its geometry under geometric transformations such as translation, rotation and affine transformations.
5. **Efficiency and Numerical Stability**: A user of a curve design system may not care about the beauty of the underlying geometry; but, he/she expects the system to deliver the curve he/she wants *fast* and *accurately*. Moreover, a large amount of computations will not "distort" the shape of the curve (*i.e.*, numerical stability).

This unit focuses on some techniques for curve design that can fulfill the above criteria. We shall discuss Bézier curves here, and B-spline and NURBS curves in the next two units. The unified theme of these techniques consists of the following advantages:

1. A user layouts a set of *control points* for the system to come up with a curve that more or less follows the trend of the set of control points.
2. A user can change the positions of some control points and some other characteristics for modifying the shape of the curve. No equation is required, because the equation of a curve is usually not stored.
3. If necessary, a user can add control points and other vital information *without* changing the shape of the curve. In this way, a user has more freedom of editing a curve because adding control points and other information increases the degree of freedom of the curve.
4. A user can even break a curve into two pieces for "micro" editing and then join them back into one piece.
5. There are very geometric, intuitive and numerically stable algorithms for finding points on the curve *without* knowing the equation of the curve.
6. Once you know curves, the surface counterpart is a few steps away. More precisely, the transition from curve to surface will not cause much difficulty, since what you learn for curves applies *directly* to surfaces.

We shall start with the most fundamental one in this unit: the Bézier curves. Bézier curves were discovered simultaneously by Paul de Casteljau at Citroen and Pierre E. Bézier at Renault around late 50s and early 60s. Basis splines, or B-splines for short, were known and studied by N. Lobachevsky whose major contribution to mathematics is perhaps the so-called hyperbolic (non-Euclidean) geometry in late eighteenth century. However, we shall adopt a modern version developed by C. de Boor, M. Cox and L. Mansfield in late 70s. Note that Bézier curves are special cases of B-splines.

Both Bézier curves and B-splines are polynomial parametric curves. As discussed in the previous unit, polynomial parametric forms cannot represent some simple curves such as circles. As a result, Bézier curves and B-splines can only represent what polynomial

parametric forms can. By introducing homogeneous coordinates making them rational, Bézier curves and B-splines are generalized to rational Bézier curves and Non-Uniform Rational B-splines, or NURBS for short. Obviously, rational Bézier curves are more powerful than Bézier curves since the former now can represent circles and ellipses. Similarly, NURBS are more powerful than B-splines. The relationship among these four types of curve representations is shown as follows :

NOTES

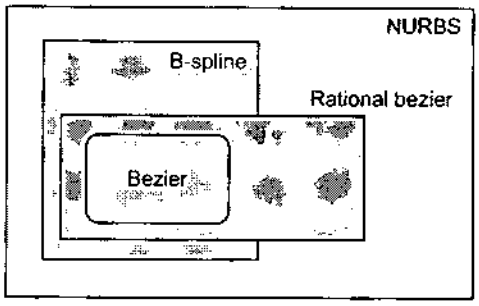


Fig. 5.11

Construction of Bézier Curves

Given $n+1$ points P_0, P_1, P_2, ... and P_n in space, the *control points*, the *Bézier curve* defined by these control points is

$$C(u) = \sum_{i=0}^{n} B_{n,i}(u) P_i$$

where the coefficients are defined as follows:

$$B_{n,i}(u) = \frac{n!}{i!(n-i)} u^i (1-u)^{n-i}$$

Therefore, the point that corresponds to u on the Bézier curve is the "weighted" average of all control points, where the weights are the coefficients $B_{n,i}(u)$. The line segments P_0P_1, P_1P_2, ..., $P_{n-1}P_n$, called *legs*, joining in this order form a *control polyline*. Many authors prefer to call this control polyline as *control polygon*. Functions $B_{n,i}(u)$, $0 <= i <= n$, are referred to as the *Bézier basis functions* or *Bernstein polynomials*.

Note that the domain of u is [0,1]. As a result, all basis functions are non-negative. In the above, since u and i can both be zero and so do $1 - u$ and $n - i$, we adopt the convention that 0^0 is 1. The following shows a Bézier curve defined by 11 control points, where the blue dot is a point on the curve that corresponds to u=0.4. As you can see in the figure, the curve more or less follows the polyline.

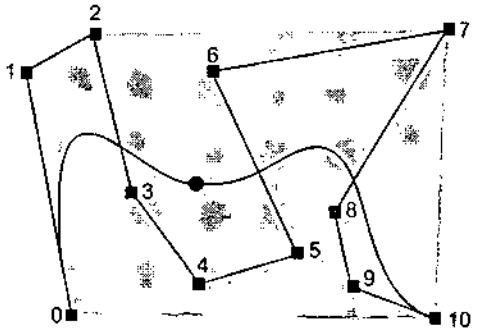


Fig. 5.12

Properties of Bézier Curve

The following properties of a Bézier curve are important:

NOTES

1. **The degree of a Bézier curve defined by $n + 1$ control points is n**: In each basis function, the exponent of u is $i + (n - i) = n$. Therefore, the degree of the curve is n.
2. **C(u) passes through P_0 and P_n**: This is shown in the above figure. The curve passes though the first and the last control point. Please verify this yourself with some simple algebraic manipulation.
3. **Non-negativity**: All basis functions are non-negative. We have mentioned this earlier.
4. **Partition of Unity**: The sum of the basis functions at a fixed u is 1. It is not difficult to verify that the basis functions are the coefficients in the binomial expansion of the expression $1 = (u + (1 - u))^n$. Hence, their sum is one. Moreover, since they are non-negative, we conclude that the value of any basis function is in the range of 0 and 1.

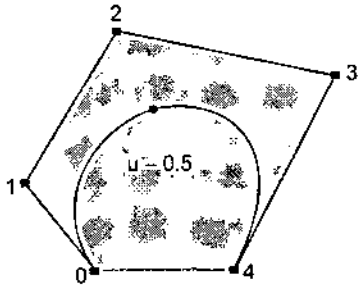


Fig. 5.13(a)

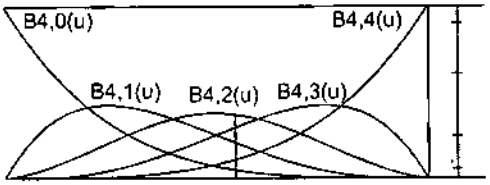


Fig. 5.13(b)

In the left figure above, we have a Bézier curve defined by five control points. Its five basis functions are functions in u as shown in the right figure above. The figures show $u = 0.5$ and all five basis functions. The right-most vertical bar shows the way of partitioning 1 into five intervals and hence the name of *partition of unity*. Note that the color of a partition is identical to the color used to draw its corresponding basis functions.

Since all basis functions are in the range of 0 an 1 and sum to one, they can be considered as weights in the computation of a weighted average. More precisely, we could say "to compute C(u), one takes the weight $B_{n,i}(u)$ for control point P_i and sum them together."

5. **Convex Hull Property**: This means the Bézier curve defined by the given $n + 1$ control points lies completely in the convex hull of the given control points. The convex hull of a set of points is the smallest convex set that contains all points. In the following figure, the convex hull of the 11 control points is shown in color gray. Note that not all control points are on the boundary of the convex hull. For example, control points 3, 4, 5, 6, 8 and 9 are in the interior. The curve, except for the first two endpoints, lies completely in the convex hull.

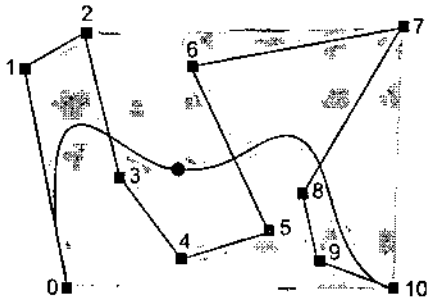


Fig. 5.13(c)

This property is important because we are guaranteed that the generated curve will be in an understood and computable region and will not go outside of it.

6. **Variation Diminishing Property**: If the curve is in a plane, this means *no straight line intersects a Bézier curve more times than it intersects the curve's control polyline.* Take a look at the above figure. The yellow line intersects the curve 3 times and the polyline 7 times; the magenta line intersects the curve 5 times and the polyline 7 times; and the cyan line intersects the curve and its polyline twice. You could draw other straight lines to verify this property.

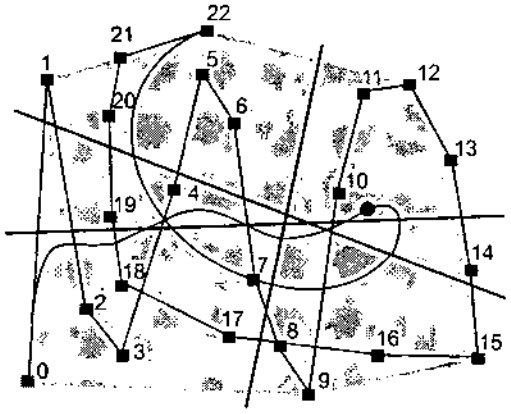


Fig. 5.13(d)

If the curve is a space curve, replacing "line" with "plane" will suffice. Special cases may occur; however, it will not be a difficult task to come up with a proper counting scheme that takes these special cases into consideration.

So, what is the meaning of this property? It tells us that the complexity (*i.e.*, turning and twisting) of the curve is no more complex than the control polyline. In other words, the control polyline twists and turns more frequently than the Bézier curve does, because an arbitrary line hits the control polyline more often than it hits the curve. Take a look at the above figure,the control polyline is more complex than the curve it defines.

7. **Affine Invariance**: If an affine transformation is applied to a Bézier curve, the result can be constructed from the affine images of its control points. This is a nice property. When we want to apply a geometric or even affine transformation to a Bézier curve, this property states that we can apply the transformation to control points, which is quite easy, and once the transformed control points are obtained the transformed Bézier curve is the one defined by these new points. Therefore, we do not have to transform the curve.

What If the Domain of u Is Not [0,1]?

Sometimes the domain of a Bézier curve is $[a, b]$ rather than $[0, 1]$. Thus, a change of variable is required. What we should do is simply converting a u in $[a, b]$ to a new $\underline{u}$ in $[0, 1]$ and using this $\underline{u}$ in the basis functions. Converting $\underline{u}$ to $[0, 1]$ can be done as follows:

$$\underline{u} = \frac{u-a}{b-a}$$

Plugging this $\underline{u}$ into the basis function $B_{n,i}(\underline{u})$ gives the following:

$$B_{n,i}(u) = \frac{n!}{i!(n-i)!}\left(\frac{u-a}{b-a}\right)^{i}\left(1-\frac{u-a}{b-a}\right)^{n-i}$$

These new basis functions define a Bézier curve on the domain of $[a, b]$.

NOTES

NOTES

Moving Control Points

Changing the position of a control point will change the shape of the defined Bézier curve. Our question is:

How does the shape of the curve change if a control point is moved to a new position?

Suppose a control point P_k is moved to a new position $P_k + v$, where vector v gives both the direction and length of this move. This is shown as follows:

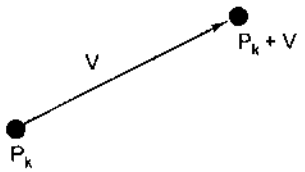


Fig. 5.14(a)

Let the original Bézier curve be as follows:

$$C(u) = \sum_{i=0}^{n} B_{n,i}(u) P_i$$

Since the new Bézier curve is defined by $P_0, P_1, ..., P_k + v, ..., P_n$, its equation $D(u)$ is

$$D(u) = \sum_{i=0}^{k-1} B_{n,i}(u) P_i + B_{n,k}(u)(P_k + v) + \sum_{i=k+1}^{n} B_{n,i}(u) P_i$$

$$= \sum_{i=0}^{n} B_{n,i}(u) P_i + B_{n,k}(u) v$$

$$= C(u) + B_{n,k}(u) v$$

In the above, since only the k-th term uses a different control point $P_k + v$, after regrouping we know that the new curve is the sum of the original curve and an extra term $B_{n,k}(u)v$. This means:

The corresponding point of u on the new curve is obtained by translating the corresponding point of u on the original curve in the direction of v with a distance of $|B_{n,k}(u)v|$.

More precisely, given a u, we have point $C(u)$ on the original curve and $D(u)$ on the new curve and $D(u) = C(u) + B_{n,k}(u)v$. Since v gives the direction of movement, $D(u)$ is the result of moving $C(u)$ in the same direction. The length of this translation is, of course, the length of vector $B_{n,k}(u)v$. Therefore, when $B_{n,k}(u)$ reaches its maximum, the change from $C(u)$ to $D(u)$ is the largest.

The following figure illustrates this effect. Both the black and red curves are Bézier curves of degree 8 defined by 9 control points. The black one is the original curve. If its control point 3 is moved to a new position as indicated by the blue vector, the black curve changes to the red one. On each of these two curves there is the point corresponding to $u = 0.5$. It is clear that $C(0.5)$ moves in the same direction to $D(0.5)$. The distance between $C(0.5)$ and $D(0.5)$ is the length of vector $B_{8,3}(0.5)v = 8!/(3!(8-3)!) \times 0.5^3(1-0.5)^{8-3}v = 0.22v$. Hence, the distance is about 22% of the distance between the original control point 3 and the new control point 3 as shown in the figure.

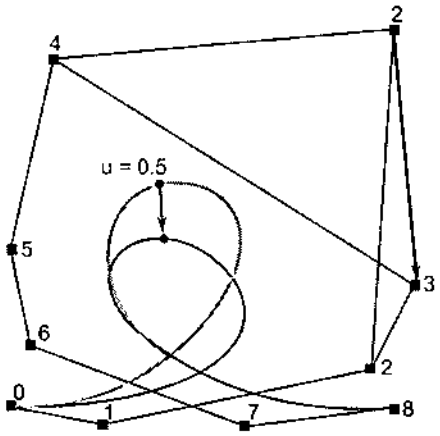


Fig. 5.14(b)

We can obtain one more important conclusion from the above discussion. Since $B_{n,k}(u)$ is non-zero in the open interval (0,1), $B_{n,k}(u)\mathbf{v}$ is not a zero vector in (0, 1). This means that except for the two endpoints C(0) and C(1) ***all*** points on the original curve are moved to new locations. Therefore, we have

Changing the position of a control point causes the shape of a Bézier curve to change globally.

Derivatives of a Bézier Curve

To compute tangent and normal vectors at a point on a Bézier curve, we must compute the first and second derivatives at that point. Fortunately, computing the derivatives at a point on a Bézier curve is easy.

Recall that the Bézier curve defined by $n + 1$ control points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$ has the following equation:

$$\mathrm{C}(u) = \sum_{i=0}^{n} \mathrm{B}_{n,i}(u)\mathrm{P}_i$$

where $B_{n,i}(u)$ is defined as follows:

$$\mathrm{B}_{n,i}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i}$$

Since the control points are constants and independent of the variable u, computing the derivative curve $\mathrm{C}'(u)$ reduces to the computation of the derivatives of $B_{n,i}(u)$'s. With some simple algebraic manipulations, we have the following result for $B'_{n,i}(u)$:

$$\frac{d}{du}\mathrm{B}_{n,i}(u) = \mathrm{B}'_{n,i}(u) = n\left(B_{n-1,i-1}(u) - B_{n-1,i}(u)\right)$$

Then, computing the derivative of the curve C(u) yields:

$$\frac{d}{du}\mathrm{C}(u) = \mathrm{C}'(u) = \sum_{i=0}^{n-1} \mathrm{B}_{n-1,i}(u)\left\{n\left(\mathrm{P}_{i+1} - \mathrm{P}_i\right)\right\}$$

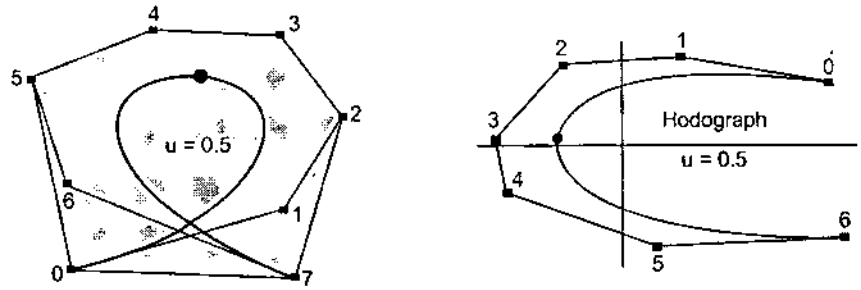
Let $\mathbf{Q}_0 = n(\mathbf{P}_1 - \mathbf{P}_0)$, $\mathbf{Q}_1 = n(\mathbf{P}_2 - \mathbf{P}_1)$, $\mathbf{Q}_2 = n(\mathbf{P}_3 - \mathbf{P}_2)$, ... $\mathbf{Q}_{n-1} = n(\mathbf{P}_n - \mathbf{P}_{n-1})$. The above equation reduces to the following:

$$\mathrm{C}'(u) = \sum_{i=0}^{n-1} \mathrm{B}_{n-1,i}(u)\mathrm{Q}_i$$

Therefore, the derivative of C(u) is a Bézier curve of degree $n - 1$ defined by n control points $n(\mathbf{P}_1 - \mathbf{P}_0)$, $n(\mathbf{P}_2 - \mathbf{P}_1)$, $n(\mathbf{P}_3 - \mathbf{P}_2)$, ... $n(\mathbf{P}_n - \mathbf{P}_{n-1})$. This derivative curve is usually

NOTES

referred to as the *hodograph* of the original Bézier curve. Note that $P_{i+1} - P_i$ is the direction vector from P_i to P_{i+1} and $n(P_{i+1} - P_i)$ is n times longer than the direction vector. Once the control points are known, the control points of its derivative curve can be obtained immediately. The left figure below shows a Bézier curve of degree 7 and the right figure shows its derivative which is a degree 6 Bézier curve.



(a) (b)

Fig. 5.15

Degree Elevation of a Bézier Curve

Many applications that involve two or more Bézier curves require all involved curves to have the same degree. Moreover, although higher degree Bézier curves require longer time to process, they do have higher flexibility for designing shapes. Therefore, it would be very helpful to increase the degree of a Bézier curve *without* changing its shape. Note that "*without changing the curve's shape*" is the key point; otherwise, just increasing the degree of a Bézier curve does not make any practical sense. Increasing the degree of a Bézier curve *without* changing its shape is referred to as *degree elevation*. In what follows, only an algorithm will be discussed.

Suppose we have a Bézier curve of degree n defined by $n+1$ control points $P_0, P_1, P_2, \ldots, P_n$ and we want to increase the degree of this curve to $n+1$ *without* changing its shape. Since a degree $n+1$ Bézier curve is defined by $n+2$ control points, we need to find such a new set of control points. Obviously, P_0 and P_n must be in the new set because the new curve also passes through them. Therefore, what we need is only n new control points. Let the new set of control points be $Q_0, Q_1, Q_2, \ldots, Q_{n+1}$. As mentioned above, $Q_0 = P_0$ and $Q_{n+1} = P_n$. The other control points are computed as follows:

$$Q_i = \frac{i}{n+1} P_{i-1} + \left(1 - \frac{i}{n+1}\right) P_i \quad 1 \le i \le n$$

If you are not comfortable with the above general formula, the following are formulae for each control points from Q_1 to Q_n.

$$Q_1 = \frac{1}{n+1} P_0 + \left(1 - \frac{1}{n+1}\right) P_1$$

$$Q_2 = \frac{2}{n+1} P_1 + \left(1 - \frac{2}{n+1}\right) P_2$$

$$Q_3 = \frac{3}{n+1} P_2 + \left(1 - \frac{3}{n+1}\right) P_3$$

$$\vdots$$

$$Q_{n-1} = \frac{n-1}{n+1} P_{n-2} + \left(1 - \frac{n-1}{n+1}\right) P_{n-1}$$

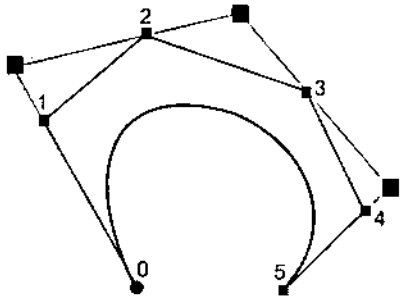
$$Q_n = \frac{n}{n+1} P_{n-1} + \left(1 - \frac{n}{n+1}\right) P_n$$

NOTES

Each leg of the original polyline contains exactly one new control point. More precisely, leg $P_{i-1}P_i$ contains new control point Q_i. Recall from the discussion of de Casteljau's algorithm that a point C on a line segment AB that divides AB in a ratio of $u : 1 - u$ can be written as $C = (1 - u)A + uB$. From the formulae for the new control points, we see that Q_i divides the segment $P_{i-1}P_i$ in a ratio of $1 - i/(n + 1):i/(n + 1)$. However, unlike de Casteljau's algorithm, this ratio is *not* a constant but varying with the value of i. This computation is very similar to that of de Casteljau's algorithm, though.

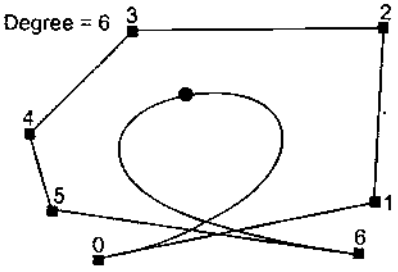
Once the new set of control points is obtained, the original set can be discarded. Since each leg of the original control polyline contains a new control point, the process of replacing the old set with the new one can be viewed as cutting off the corners at the original control points. The following figure illustrates this corner-cutting effect. The figure shows a Bézier curve of degree 4 whose control points are shown in red rectangles and control polyline in blue dashed line segments. After increasing its degree to 5, the new control polyline is shown in solid line segments. It is clear that all corners are cut. The left table gives the ratio on each leg of the original control polyline.

i	$1 - i/(n + 1)$
1	0.8
2	0.6
3	0.4
4	0.2

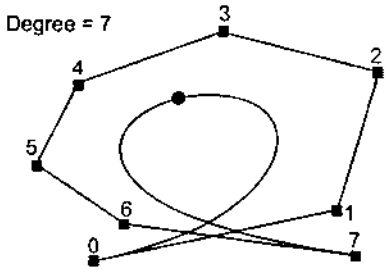


(a)

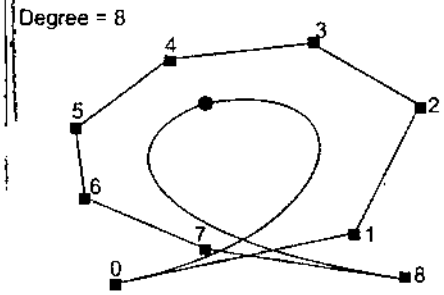
Note that degree elevation can be used repeatedly as long as your system permits. Note also that as the degree increases the number of control points increases. Moreover, the new control polyline moves toward the curve. In the following figures, we start with a Bézier curve of degree 6 with 7 control points. Then, its degree is increased to 7, 8, 10, 15 and 29. As you can see from the figures, the shape of the curve is not changed as its degree increases and the control polyline moves closer and closer to the curve. Eventually, as the degree keeps increasing to infinity, the control polyline approaches to the curve and has it as a limiting position.



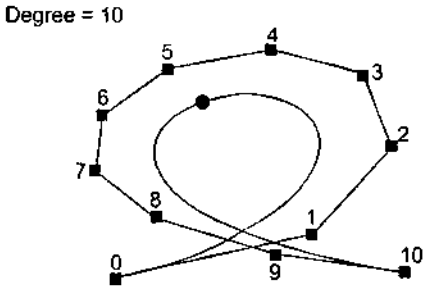
(b)



(c)

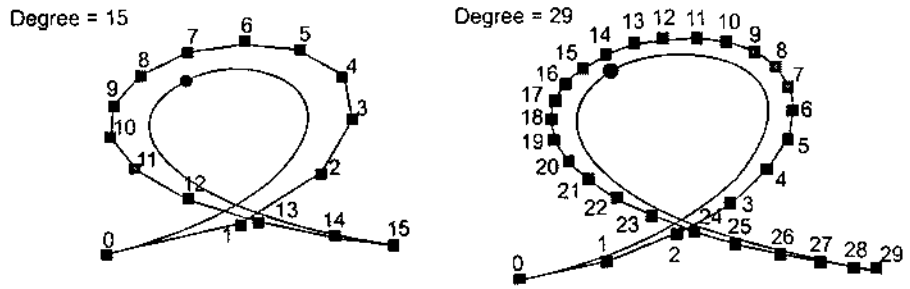


(d)



(e)

NOTES



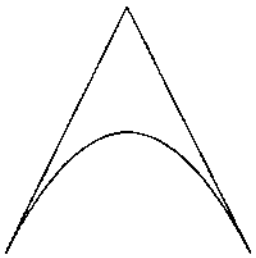
(f) (g)

Fig. 5.16

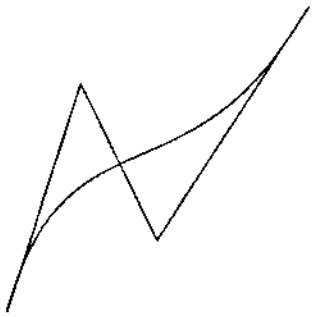
Examples:

The pink lines show the control point polygon, the grey lines the Bézier curve.

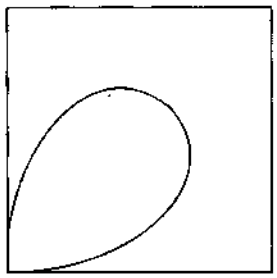
The degree of the curve is one less than the number of control points, so it is a quadratic for 3 control points. It will always be symmetric for a symmetric control point arrangement.



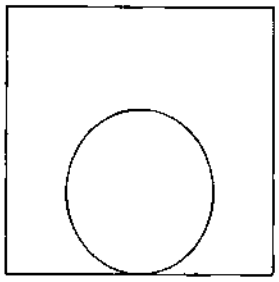
The curve always passes through the end points and is tangent to the line between the last two and first two control points. This permits ready piecing of multiple Bézier curves together with first order continuity.



The curve always lies within the convex hull of the control points. Thus the curve is always "well behaved" and does not oscillating erratically.



Closed curves are generated by specifying the first point the same as the last point. If the tangents at the first and last points match then the curve will be closed with first order continuity. In addition, the curve may be pulled towards a control point by specifying it multiple times.



NOTES

Implementation of Bézier Curves Using C Language C Source

```
/*
   Three control point Bézier interpolation
   mu ranges from 0 to 1, start to end of the curve
*/
XYZ Bezier3(XYZ p1,XYZ p2,XYZ p3,double mu)
{
   double mum1,mum12,mu2;
   XYZ p;
   mu2 = mu * mu;
   mum1 = 1 - mu;
   mum12 = mum1 * mum1;
   p.x = p1.x * mum12 + 2 * p2.x * mum1 * mu + p3.x * mu2;
   p.y = p1.y * mum12 + 2 * p2.y * mum1 * mu + p3.y * mu2;
   p.z = p1.z * mum12 + 2 * p2.z * mum1 * mu + p3.z * mu2;
   return(p);
}
/*
   Four control point Bezier interpolation
   mu ranges from 0 to 1, start to end of curve
*/
XYZ Bezier4(XYZ p1,XYZ p2,XYZ p3,XYZ p4,double mu)
{
   double mum1,mum13,mu3;
   XYZ p;
   mum1 = 1 - mu;
   mum13 = mum1 * mum1 * mum1;
   mu3 = mu * mu * mu;
   p.x = mum13*p1.x + 3*mu*mum1*mum1*p2.x + 3*mu*mu*mum1*p3.x + mu3*p4.x;
   p.y = mum13*p1.y + 3*mu*mum1*mum1*p2.y + 3*mu*mu*mum1*p3.y + mu3*p4.y;
   p.z = mum13*p1.z + 3*mu*mum1*mum1*p2.z + 3*mu*mu*mum1*p3.z + mu3*p4.z;
   return(p);
}
/*
   General Bezier curve
   Number of control points is n+1
   0 <= mu < 1 IMPORTANT, the last point is not computed
*/
XYZ Bezier(XYZ *p,int n,double mu)
{
   int k,kn,nn,nkn;
   double blend,muk,munk;
   XYZ b = {0.0,0.0,0.0};
```

NOTES

```
    muk = 1;
    munk = pow(1-mu,(double)n);
    for (k=0;k<=n;k++) {
      nn = n;
      kn = k;
      nkn = n - k;
      blend = muk * munk;
      muk *= mu;
      munk /= (1-mu);
      while (nn >= 1) {
         blend *= nn;
         nn--;
         if (kn > 1) {
            blend /= (double)kn;
            kn--;
}
if (nkn > 1) {
          blend /= (double)nkn;
          nkn--;
  }
}
b.x += p[k].x * blend;
b.y += p[k].y * blend;
b.z += p[k].z * blend;
}
return(b);
}
```

5.20 B-SPLINE CURVES

Given $n+1$ control points $P_0, P_1, ..., P_n$ and a knot vector $U = \{ u_0, u_1, ..., u_m \}$, the B-spline curve of degree p defined by these control points and knot vector U is

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i$$

where $N_{i,p}(u)$'s are B-spline basis functions of degree p. The form of a B-spline curve is very similar to that of a Bézier curve. Unlike a Bézier curve, a B-spline curve involves more information, namely: a set of $n+1$ control points, a knot vector of $m+1$ knots, and a degree p. Note that n, m and p must satisfy $m = n + p + 1$. More precisely, if we want to define a B-spline curve of degree p with $n+1$ control points, we have to supply $n+p+2$ knots $u_0, u_1, ..., u_{n+p+1}$. On the other hand, if a knot vector of $m+1$ knots and $n+1$ control points are given, the degree of the B-spline curve is $p = m - n - 1$. The point on the curve that corresponds to a knot u_i, $C(u_i)$, is referred to as a *knot point*. Hence, the knot points divide a B-spline curve into curve segments, each of which is defined on a knot span. We shall show that these curve segments are all Bézier curve of degree p on the curve subdivision page.

Although $N_{i,p}(u)$ looks like $B_{n,i}(u)$, the degree of a B-spline basis function is an input, while the degree of a Bézier basis function depends on the number of control points. To change the shape of a B-spline curve, one can modify one or more of these control parameters: the positions of control points, the positions of knots, and the degree of the curve.

NOTES

If the knot vector does not have any particular structure, the generated curve will not touch the first and last legs of the control polyline as shown in the left figure below. This type of B-spline curves is called *open* B-spline curves. We may want to clamp the curve so that it is tangent to the first and the last legs at the first and last control points, respectively, as a Bézier curve does. To do so, the first knot and the last knot must be of multiplicity $p+1$. This will generate the so-called *clamped* B-spline curves. See the middle figure below. By repeating some knots and control points, the generated curve can be a *closed* one. In this case, the start and the end of the generated curve join together forming a closed loop as shown in the right figure below. In this note, we shall use clamped curve.

The above figures have $n+1$ control points ($n=9$) and $p=3$. Then, m must be 13 so that the knot vector has 14 knots. To have the clamped effect, the first $p+1=4$ and the last 4 knots must be identical. The remaining $14-(4+4)=6$ knots can be anywhere in the domain. In fact, the curve is generated with knot vector $U=\{0, 0, 0, 0, 0.14, 0.28, 0.42, 0.57, 0.71, 0.85, 1, 1, 1, 1\}$. Note that except for the first four and last four knots, the middle ones are almost uniformly spaced. The figures also show the corresponding curve segment on each knot span. In fact, the little triangles are the knot points.

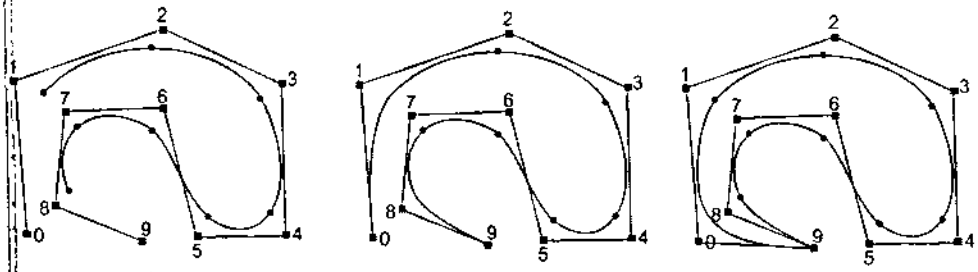


Fig.5.17

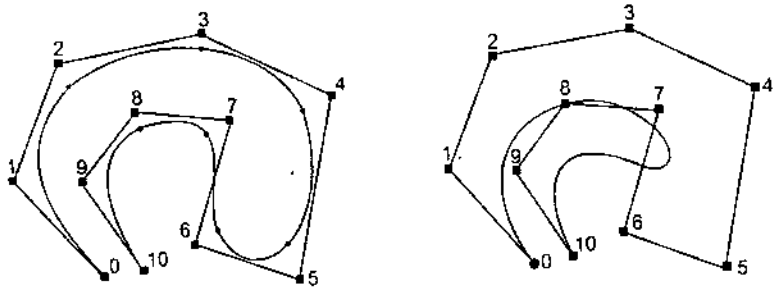
Properties of B-spline Curves

B-spline curves share many important properties with Bézier curves, because the former is a generalization of the later. Moreover, B-spline curves have more desired properties than Bézier curves. The list below shows some of the most important properties of B-spline curves.

In the following we shall assume a B-spline curve $C(u)$ of degree p is defined by $n+1$ control points and a knot vector $U=\{u_0, u_1, \ldots, u_m\}$ with the first $p+1$ and last $p+1$ knots "clamped" (*i.e.*, $u_0=u_1=\ldots=u_p$ and $u_{m-p}=u_{m-p+1}=\ldots=u_m$).

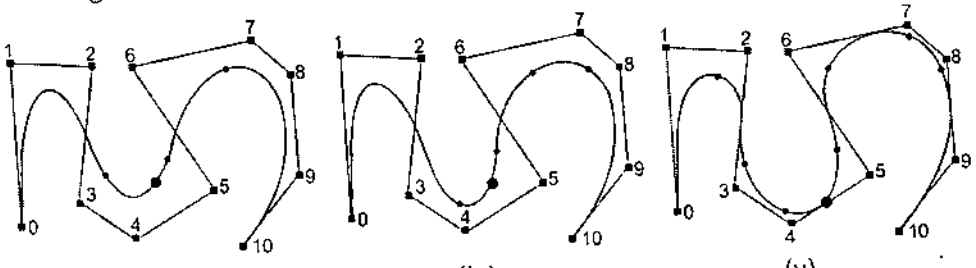
1. **B-spline curve $C(u)$ is a piecewise curve with each component a curve of degree p.** We know that, $C(u)$ can be viewed as the union of curve segments defined on each knot span. In the figure below, where $n=10$, $m=14$ and $p=3$, the first four knots and last four knots are clamped and the 7 internal knots are uniformly spaced. There are eight knot spans, each of which corresponds to a curve segment. In the left figure below, these knot points are shown as triangles. This nice property allows us to design complex shapes with lower degree polynomials. For example, the right figure below shows a Bézier curve with the same set of control points. It still cannot follow the control polyline nicely even though its degree is 10!

NOTES



(i) (ii)

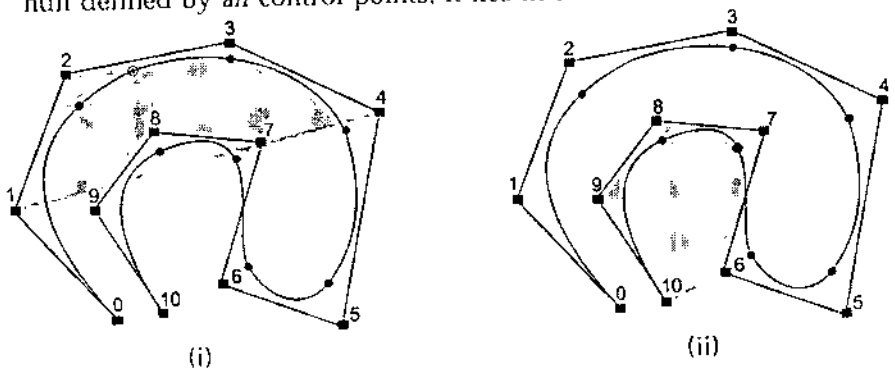
In general, the lower the degree, the closer a B-spline curve follows its control polyline. The following figures all use the same control polyline and knots are clamped and uniformly spaced. The first figure has degree 7, the middle one has degree 5 and the right figure has degree 3. Therefore, as the degree decreases, the generated B-spline curve moves closer to its control polyline.



(iii) (iv) (v)

Fig. 5.18(a)

2. **Equality $m = n + p + 1$ must be satisfied.** Since each control point needs a basis function and the number of basis functions satisfies $m = n + p + 1$.
3. **Clamped B-spline curve C(u) passes through the two end control points P_0 and P_n.** Note that basis function $N_{0,p}(u)$ is the coefficient of control point P_0 and is non-zero on $[u_0, u_{p+1}]$. Since $u_0 = u_1 = ... = u_p = 0$ for a clamped B-spline curve, $N_{0,0}(u)$, $N_{1,0}(u)$, ..., $N_{p-1,0}(u)$ are zero and only $N_{p,0}(u)$ is non-zero (recall from the triangular computation scheme). Conse quently, if $u = 0$, then $N_{0,p}(0)$ is 1 and $C(0) = P_0$. A similar discussion can show $C(1) = P_n$.
4. **Strong Convex Hull Property: A B-spline curve is contained in the convex hull of its control polyline. More specifically, if u is in knot span $[u_i, u_{i+1}]$, then C(u) is in the convex hull of control points P_{i-p}, P_{i-p+1}, ... P_i.** If u is in knot span (u_i, u_{i+1}), there are only $p + 1$ basis functions (*i.e.*, $N_{i,p}(u)$, ... , $N_{i-p+1,p}(u)$, $N_{i-p,p}(u)$) non-zero on this knot span. Since $N_{k,p}(u)$ is the coefficient of control point P_k, only $p + 1$ control points P_i, P_{i-1}, P_{i-2}, ... P_{i-p} have non-zero coefficients. Since on this knot span the basis functions are non-zero and sum to 1, their "weighted" average, C(u), must lie in the convex hull defined by control points P_i, P_{i-1}, P_{i-2}, ..., P_{i-p}. The meaning of "strong" is that while C(u) still lies in the convex hull defined by *all* control points, it lies in a much smaller one.



(i) (ii)

Fig. 5.18(b)

The above two B-spline curves have 11 control points (*i.e.*, $n = 10$), degree 3 (*i.e.*, $p = 3$) and 15 knots ($m = 14$) with first four and last four knots clamped. Therefore, the number of knot spans is equal to the number curve segments. The knot vector is

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}
0	0	0	0	0.12	0.25	0.37	0.5	0.62	0.75	0.87	1	1	1	1

The left figure has u in knot span $(u_4, u_5) = (0.12, 0.25)$ and the corresponding point (*i.e.* $C(u)$) in the second curve segment. Therefore, there are $p + 1 = 4$ basis functions non-zero on this knot span (*i.e.*, $N_{4,3}(u)$, $N_{3,3}(u)$, $N_{2,3}(u)$ and $N_{1,3}(u)$) and the corresponding control points are P_4, P_3, P_2 and P_1. The shaded area is the convex hull defined by these four points. It is clear that $C(u)$ lies in this convex hull.

The B-spline curve in the right figure is defined the same way. However, u is in $(u_9, u_{10}) = (0.75.0.87)$ and the non-zero basis functions are $N_{9,3}(u)$, $N_{8,3}(u)$, $N_{7,3}(u)$ and $N_{6,3}(u)$. The corresponding control points are P_9, P_8, P_7 and P_6.

Consequently, as u, moves from 0 to 1 and crosses a knot, a basis functions becomes zero and a new non-zero basis function becomes effective. As a result, one control point whose coefficient becomes zero will leave the definition of the current convex hull and is replaced with a new control point whose coefficient becomes non-zero.

5. **Local Modification Scheme: Changing the position of control point P_i only affects the curve $C(u)$ on interval $[u_i, u_{i+p+1}]$.** This follows from another important property of B-spline basis functions. Recall that $N_{i,p}(u)$ is non-zero on interval (u_i, u_{i+p+1}). If u is not in this interval, $N_{i,p}(u)P_i$ has no effect in computing $C(u)$ since $N_{i,p}(u)$ is zero. On the other hand, if u is in the indicated interval, $N_{i,p}(u)$ is non-zero. If P_i changes its position, $N_{i,p}(u)P_i$ is changed and consequently $C(u)$ is changed.

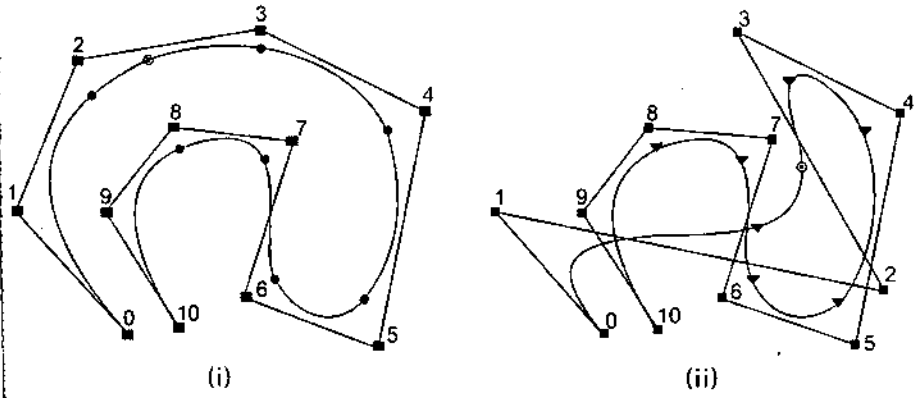


Fig. 5.18(c)

The above B-spline curves are defined with the same parameters as in the previous convex hull example. We intent to move control point P_2. The coefficient of this control point is $N_{2,3}(u)$ and the interval on which this coefficient is non-zero is $(u_2, u_{2+3+1}) = (u_2, u_6) = (0,0.37)$. Since $u_2 = u_3 = 0$, only three segments that correspond to (u_3, u_4) (the domain of the first curve segment), (u_4, u_5) (the domain of the second curve segment) and (u_5, u_6) (the domain of the third curve segment) will be affected. The right figure shows the result of moving P_2 to the lower right corner. As you can see, only the first, second and third curve segments change their shapes and all remaining curve segments stay in their original place without any change.

NOTES

NOTES

This local modification scheme is very important to curve design, because we can modify a curve locally without changing the shape in a global way. This will be elaborated on the moeing control point page. Moreover, if fine-tuning curve shape is required, one can insert more knots (and therefore more control points) so that the affected area could be restricted to a very narrow region. We shall talk about knot insertion later.

6. **C(u) is C^{p-k} continuous at a knot of multiplicity k.** If u is not a knot, C(u) is in the middle of a curve segment of degree p and is therefore infinitely differentiable. If u is a knot in the non-zero domain of $N_{i,p}(u)$, since the latter is only C^{p-k} continuous, so does C(u).

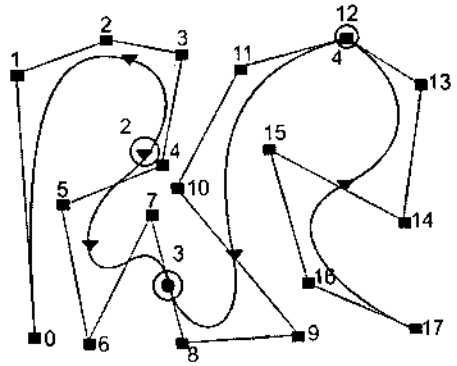


Fig. 5.18(d)

The above B-spline curve has 18 control points (*i.e.*, $n = 17$), degree 4, and the following clamped knot vector

u_0 to u_4	u_5	u_6 and u_7	u_8	u_9 to u_{11}	u_{12}	u_{13} to u_{16}	u_{17}	u_{18} to u_{22}
0	0.125	0.25	0.375	0.5	0.625	0.75	0.875	1

Thus, u_6 is a double knot, u_9 is a triple knot and u_{13} is a quadruple knot. Consequently, C(u) is of C^1 continuous at any point that is not a knot, C^3 continuous at all simple knots, C^2 continuous at u_6, C^1 continuous at u_9, C^0 continuous at u_{13}.

All points on the curve that correspond to knots are marked with little triangles. Those corresponding to multiple knots are further marked with circles and their multiplicities. It is very difficult to visualize the difference between C^4, C^3 and even C^2 continuity. For the C^1 case, the corresponding point lies on a leg, while the C^0 case forces the curve to pass through a control point. We shall return to this issue later when discussing modifying knots.

7. **Variation Diminishing Property**:

The variation diminishing property also holds for B-spline curves. If the curve is in a plane (*resp.*, space), this means *no straight line (resp., plane) intersects a B-spline curve more times than it intersects the curve's control polyline.*

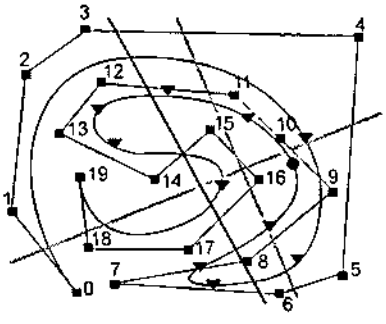


Fig. 5.18(e)

In the above figure, the blue line intersects both the control polyline and the B-spline curve 6 times, while the yellow line also intersects the control polyline and the B-spline curve 5 times. However, the orange line intersects the control polyline 6 times and the curve 4 times.

8. **Bézier Curves Are Special Cases of B-spline Curves.**

   If $n = p$ (*i.e.*, the degree of a B-spline curve is equal to n, the number of control points minus 1), and there are $2(p + 1) = 2(n + 1)$ knots with $p + 1$ of them clamped at each end, this B-spline curve reduces to a Bézier curve.

9. **Affine Invariance**

   The affine invariance property also holds for B-spline curves. If an affine transformation is applied to a B-spline curve, the result can be constructed from the affine images of its control points. This is a nice property. When we want to apply a geometric or even affine transformation to a B-spline curve, this property states that we can apply the transformation to control points, which is quite easy, and once the transformed control points are obtained the transformed B-spline curve is the one defined by these new points. Therefore, we do not have to transform the curve.

NOTES

Advantage of Using B-spline Curves

B-spline curves require more information (*i.e.*, the degree of the curve and a knot vector) and a more complex theory than Bézier curves. But, it has more advantages to offset this shortcoming. First, a B-spline curve can be a Bézier curve. Second, B-spline curves satisfy all important properties that Bézier curves have. Third, B-spline curves provide more control flexibility than Bézier curves can do. For example, the degree of a B-spline curve is separated from the number of control points. More precisely, we can use lower degree curves and still maintain a large number of control points. We can change the position of a control point without globally changing the shape of the whole curve (local modification property). Since B-spline curves satisfy the strong convex hull property, they have a finer shape control. Moreover, there are other techniques for designing and editing the shape of a curve such as changing knots.

However, keep in mind that B-spline curves are still polynomial curves and polynomial curves cannot represent many useful simple curves such as circles and ellipses. Thus, a generalization of B-spline, NURBS, is required.

5.21 HERMITE SPLINE

In the mathematical subfield of numerical analysis, a **Hermite spline** is a spline curve where each polynomial of the spline is in Hermite form.

Hermite curves are very easy to calculate but also very powerful. They are used to smoothly interpolate between key-points (like object movement in keyframe animation or camera control). Understanding the mathematical background of hermite curves will help you to understand the entire family of splines. Maybe you have some experience with 3-D programming and have already used them without knowing that (the so called kb-splines, curves with control over tension, continuity and bias are just a special form of the hermite curves).

To keep it simple we first start with some simple stuff. We also only talk about two-dimensional curves here. If you need a 3-D curve just do with the z-coordinate what you do with y or x. Hermite curves work in any number of dimensions.

NOTES

To calculate a hermite curve you need the following vectors:

- P_1: the startpoint of the curve
- T_1: the tangent (*e.g.*, direction and speed) to how the curve leaves the startpoint
- P_2: the endpoint of the curve
- T_2: the tangent (*e.g.*, direction and speed) to how the curves meets the endpoint

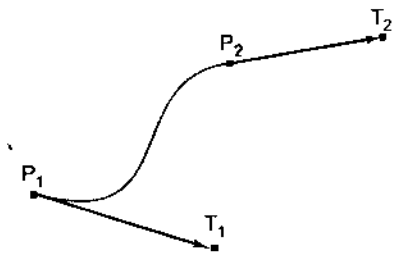


Fig. 5.19(a)

These 4 vectors are simply multiplied with 4 hermite basis functions and added together.

$h_1(s) = 2s^3 - 3s^2 + 1$
$h_2(s) = -2s^3 + 3s^2$
$h_3(s) = s^3 - 2s^2 + s$
$h_4(s) = s^3 - s^2$

Below are the 4 graphs of the 4 functions (from left to right: h_1, h_2, h_3, h_4).

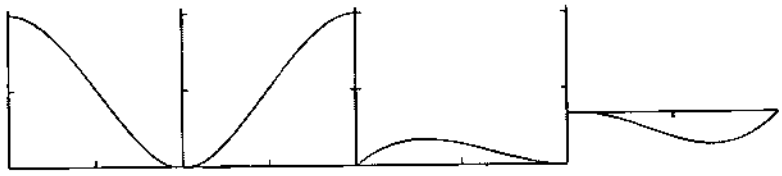


Fig. 5.19(b)

(all graphs except the 4th have been plotted from 0.0 to 1.1)

Take a closer look at functions h_1 and h_2:

- h_1 starts at 1 and goes slowly to 0.
- h_2 starts at 0 and goes slowly to 1.

Now multiply the startpoint with h_1 and the endpoint with h_2. Let s go from 0 to 1 to interpolate between start and endpoint. h_3 and h_4 are applied to the tangents in the same manner. They make sure that the curve bends in the desired direction at the start and endpoint.

Matrix Form

All this stuff can be expessed with some vector and matrix algebra. I think the matrix-form is much easier to understand.

```
Vector S: The interpolation-point and it's powers up to 3:
Vector C: The parameters of our hermite curve:
Matrix h: The matrix form of the 4 hermite polynomials:
```

$$s = \begin{vmatrix} s^3 \\ s^2 \\ s^1 \\ 1 \end{vmatrix} \quad c = \begin{vmatrix} P_1 \\ P_2 \\ T_1 \\ T_2 \end{vmatrix} \quad h = \begin{vmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix}$$

To calculate a point on the curve you build the Vector S, multiply it with the matrix h and then multiply with C.

P = S • h • C

5.22 THE FRACTAL MAGIC

NOTES

What is a Fractal?

According to B. Mandelbrot: "A rough or fragmented geometric shape that can be subdivided in parts, each of which is (at least approximately) a reduced/size copy of the whole."

Fractals are geometric figures, just like rectangles, circles and squares, but fractals have special properties that those figures do not have.

Fractal Properties

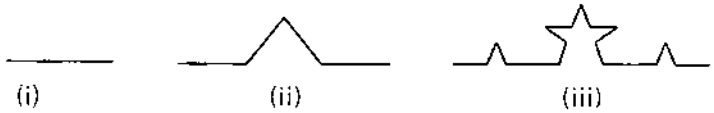


Fig. 5.20(a)

A **point** has no dimensions - no length, no width, no height. That dot is obviously way too big to really represent a point. But we'll live with it, if we all just agree what a point really is.

A line has one dimension - length. It has no width and no height, but infinite length.

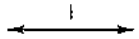


Fig. 5.20(b)

Again, this model of a line is really not very good, but until we learn how to draw a line with 0 width and infinite length, it'll have to do.

A **plane** has two dimensions - length and width, no depth.

Fig. 5.20(c)

It's an absolutely flat tabletop extending out both ways to infinity.

Space, a huge empty box, has three dimensions, length, width, and depth, extending to infinity in all three directions.

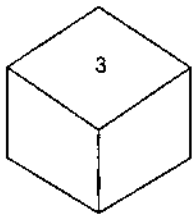


Fig. 5.20(d)

Obviously this isn't a good representation of 3-D. Besides its size, it's just a hexagon drawn to fool you into thinking it's a box.

NOTES

Fractals can have ***fractional (or fractal) dimension.*** A fractal might have dimension of 1.6 or 2.4. How could that be? Let's investigate below.

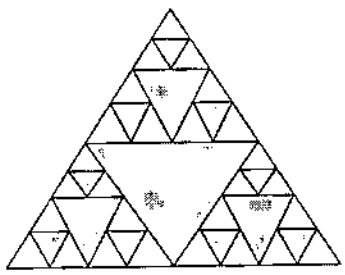


Fig. 5.20(e)

Just as the images above weren't very good pictures of a point, line, plane, or space, the drawing meant to be the Sierpinski Triangle has limitations. Remember as we continue that fractals are really formed by infinitely many steps. So there are infinitely many smaller and smaller triangles inside the figure, and infinitely many holes (the black triangles).

Let's look further at what we mean by dimension. Take a self-similar figure like a line segment, and double its length. Doubling the length gives two copies of the original segment.

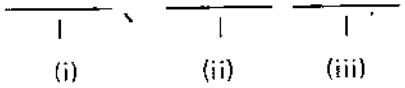


Fig. 5.20(f)

Take another self-similar figure, this time a square 1 unit by 1 unit. Now multiply the length and width by 2. How many copies of the original size square do you get? Doubling the sides gives four copies.

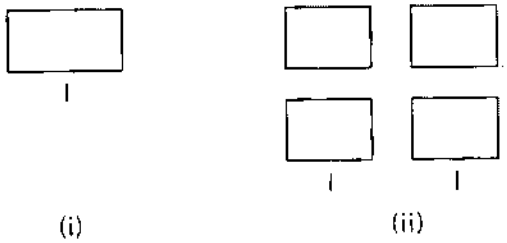


Fig. 5.20(g)

Take a 1 by 1 by 1 cube and double its length, width, and height. How many copies of the original size cube do you get? Doubling the side gives eight copies.

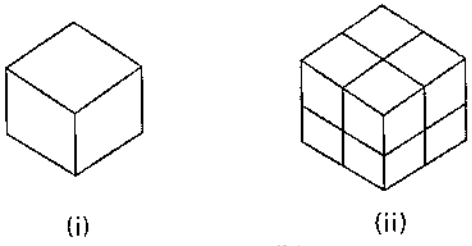


Fig. 5.20(h)

Let's organize our information into a table.

Figure	*Dimension*	*No. of Copies*
Line	1	$2 = 2^1$
Square	2	$4 = 2^2$
Cube	3	$8 = 2^3$

Do you see a pattern? It appears that the dimension is the exponent - and it is! So when we double the sides and get a similar figure, we write the number of copies as a power of 2 and the exponent will be the dimension.

Let's add that as a row to the table.

Figure	*Dimension*	*No. of Copies*
Line	1	$2 = 2^1$
Square	2	$4 = 2^2$
Cube	3	$8 = 2^3$
Doubling Similarity	d	$n = 2^d$

NOTES

We can use this to figure out the dimension of the Sierpinski Triangle because when you double the length of the sides, you get another Sierpinski Triangle similar to the first.

Start with a Sierpinski triangle of 1-inch sides. Double the length of the sides. Now how many copies of the original triangle do you have? Remember that the black triangles are holes, so we can't count them.

Doubling the sides gives us three copies, so $3 = 2^d$, where d = the dimension.

But wait. $2 = 2^1$, and $4 = 2^2$, so what number could this be? It has to be somewhere between 1 and 2, right? Let's add this to our table

Figure	*Dimension*	*No. of Copies*
Line	1	$2 = 2^1$
Sierpinski's Triangle	?	$3 = 2^?$
Square	2	$4 = 2^2$
Cube	3	$8 = 2^3$
Doubling Similarity	d	$n = 2^d$

So the dimension of Sierpinski's Triangle is between 1 and 2. Do you think you could find a better answer? Use a calculator with an exponent key (the key usually looks like this ^). Use 2 as a base and experiment with different exponents between 1 and 2 to see how close you can come. For example, try 1.1. Type 2^1.1 and you get 2.143547. I'll bet you can get closer to 3 than that. Try 2^1.2 and you get 2.2974. That's closer to 3, but you can do better.

That is how fractals can have fractional dimension

5.23 FRACTAL GEOMETRY

Almost all geomtric forms used for building man made objects belong to Euclidean geometry, they are comprised of lines, planes, rectangular volumes, arcs, cylinders, spheres, etc. These elements can be classified as belonging to an integer dimension, either 1, 2, or 3. This concept of dimension can be described both intuitively and mathematically. Intuitively we say that a line is one dimensional because it only takes 1 number to uniquely define any point on it. That one number could be the distance from the start of the line. This applies equally well to the circumference of a circle, a curve, or the boundary of any object.

A plane is two dimensional since in order to uniquely define any point on its surface we require two numbers. There are many ways to arrange the definition of these two numbers but we normally create an orthogonal coordinate system. Other examples of two dimensional objects are the surface of a sphere or an arbitrary twisted plane.

The volume of some solid object is 3 dimensional on the same basis as above, it takes three numbers to uniquely define any point within the object.

A more mathematical description of dimension is based on how the "size" of an object behaves as the linear dimension increases. In one dimension consider a line segment. If

NOTES

the linear dimension of the line segment is doubled then obviously the length (characteristic size) of the line has doubled. In two dimensions, if the linear dimensions of a rectangle for example is doubled then the characteristic size, the area, increases by a factor of 4. In three dimensions if the linear dimension of a box are doubled then the volume increases by a factor of 8. This relationship between dimension D, linear scaling L and the resulting increase in size S can be generalised and written as

$$S = L^D$$

This is just telling us mathematically what we know from everyday experience. If we scale a two dimensional object for example then the area increases by the square of the scaling. If we scale a three dimensional object the volume increases by the cube of the scale factor. Rearranging the above gives an expression for dimension depending on how the size changes as a function of linear scaling, namely

$$D = \log(S)/\log(L)$$

In the examples above the value of D is an integer, either 1, 2, or 3, depending on the dimension of the geometry. This relationship holds for all Euclidean shapes. There are however many shapes which do not conform to the integer based idea of dimension given above in both the intuitive and mathematical descriptions. That is, there are objects which appear to be curves for example but which a point on the curve cannot be uniquely described with just one number. If the earlier scaling formulation for dimension is applied the formula does not yield an integer. There are shapes that lie in a plane but if they are linearly scaled by a factor L, the area does not increase by L squared but by some non integer amount. These geometries are called fractals! One of the simpler fractal shapes is the von Koch snowflake. The method of creating this shape is to repeatedly replace each line segment with the following 4 line segments.

The process starts with a single line segment and continues for ever. The first few iterations of this procedure are shown below.

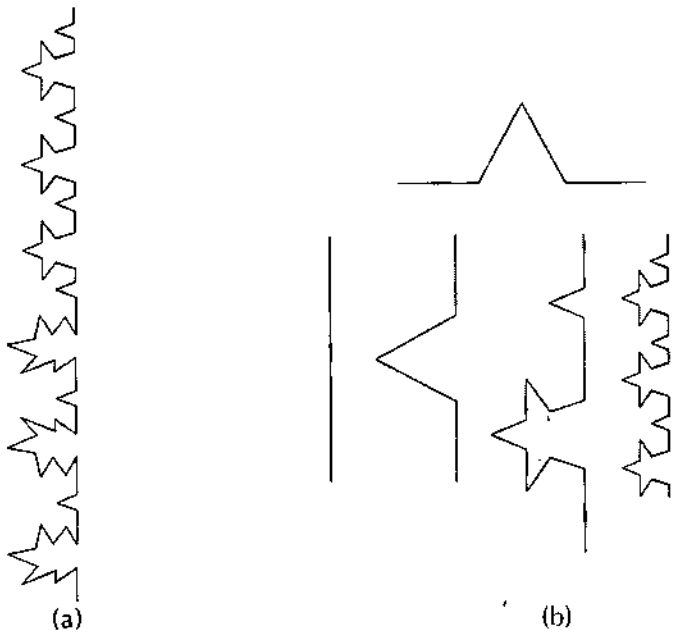


Fig. 5.21

This demonstrates how a very simple generation rule for this shape can generate some unusual (fractal) properties. Unlike Euclidean shapes this object has detail at all levels. If one magnifies an Euclidean shape such as the circumference of a circle it becomes a different shape, namely a striaght line. If we magnify this fractal more and more detail is

uncovered, the detail is self similar or rather it is exactly self similar. Put another way, any magnified portion is identical to any other magnified portion.

5.24 ITERATIVE FORMATION

Fractals are often formed by what is called an iterative process

To make a fractal: Take a familiar geometric figure (a triangle or line segment, for example) and operate on it so that the new figure is more "complicated" in a special way.

NOTES

Then in the same way, operate on that resulting figure, and get an even more complicated figure.

Now operate on that resulting figure in the same way and get an even more complicated figure.

Do it again and again...and again. In fact, you have to think of doing it infinitely many times.

You can observe this iterative process in all the fractals that we make

- Sierpinski's Triangle
- Koch Snowflake.

5.25 INTRODUCTION TO COMPUTER ANIMATION

Animation is the creation of the illusion of movement by assembling a sequence of still images.

To 'animate' is literally 'to give life to'. 'Animating' is moving something which can't move itself. These pictures do not really move — they are composed of a series of static images that affect the eyes at the rate of 12 to 24 images per second. The illusion of movement is caused by a physiological affect known as 'persistence of vision'.

Animation adds to graphics the dimension of time which vastly increases the amount of information which can be transmitted. In order to animate something, the animator has to be able to specify, either directly or indirectly, how the 'thing' is to move through time and space. The basic problem is to select or design animation tools which are expressive enough for the animator to specify what he wants to specify while at the same time are powerful or automatic enough that the animator doesn't have to specify the details that he is not interested in. Obviously, there is no one tool that is going to be right for every animator, for every animation, or even for every scene in a single animation. The appropriateness of a particular animation tool depends on the effect desired by the animator. An artistic piece of animation will probably require different tools that an animation intended to simulate reality.

There are two main categories of computer animation: ***computer-assisted animation and computer generated animation.*** This book is mainly concerned with computer-generated animation. For discussion purposes, motion specification for computer-generated animation is divided into two categories: low level techniques (techniques that aid the animator in precisely specifying motion), and high level techniques (techniques used to describe general motion behavior) .

Low level techniques consist of techniques, such as shape interpolation algorithms, which help the animator fill in the details of the motion once enough information about the motion has been specified by the animator. When using low level techniques, the animator usually has a fairly specific idea of the exact motion that he or she wants.

NOTES

High level techniques are typically algorithms or models used to generate a motion using a set of rules or constraints. The animator sets up the rules of the model, or chooses an appropriate algorithm, and selects initial values or boundary values. The system is then set into motion, so to speak, and the motion of the objects is controlled by the algorithm or model. The model-based/algorithmic approaches often rely on fairly sophisticated computation, such as physically based motion control.

Any technique requires a certain amount of effort from the animator and a certain amount of effort from the computer. One of the things which distinguishes animation techniques is whether its the animator or the computer which bears most of the burden. Motion specification aids are those techniques which seem to require more input from the user and fairly straightforward computation. Model-based approaches, on the other hand, require less from the animator and more computation. But the categories are really artificial.

Another way to characterize the difference between techniques is to look at the level of abstraction at which the animator is working. In one extreme, at a very low level of abstraction, the animator could color in every pixel individually in every frame. At the other extreme, at a very high level of abstraction, the animator could tell a computer to 'make a movie about a dog'. Presumably, the computer would whirl away while it computes such a thing. A high level of abstraction frees the animator from dealing with all of the details. A low level of abstraction allows the animator to be very precise in specifying exactly what is displayed when. In reality, animators want to be able to switch back and forth and work at various levels of abstraction. The challenge to developing animation tools is designing the tools so that animators are allowed to work at high levels of abstraction when desired, while providing them the ability to work at low levels when needed.

5.26 PERCEPTION

Images convey a lot of information because the human visual system is a sophisticated information processor. It follows, then, that moving images have the potential to convey much more information.

When animation is recorded for later viewing, it is typcially presented in film or video formats by recording a series of still images. This is possible because the eye-brain assembles a sequence of images and interprets them as a continuous movement. Persistence of motion is created by presenting a sequence of still images at a fast enough rate to induce the sensation of continuous motion.

The receptors in the eye continually sample light in the environment. The only limitation on motion detection is the reaction time of those sensors and on certain mechanical limitations such as blinking and tracking. If an object moves fast enough, then the receptors in the eye will not be able to respond fast enough for the brain to distinguish a sharply defined, individual detail; motion blur results.

In either film or video, a sequence of images is recorded which can be played back at rates fast enough to fool the eye into interpreting them as continuous motion. Of course, in order to save resources, this rate is kept as low as possible while still maintaining the persistence of motion. Under some viewing conditions such as room lighting and viewing distance, the rate at which single images must be played back in order to maintain the perception of motion varies. The image is said to flicker when the perception of continuous motion fails to be created. The object appears as a rapid sequence of still images to the eye-brain.

There are actually two rates that are of concern. One is the number of images per second that are displayed in the viewing process. The other, is the number of different images that occur per second. The former is the *playback rate*: the latter is the *sampling rate* or *update rate*. For example, images are always played back at 30 images per second on a TV; but in some Saturday morning cartoons there may be only six different images per second with each image repeated five times.

5.27 THE EARLY DAYS OF ANIMATION

NOTES

Persistence of vision was discovered in the 1800s. This led to such devices as the zoetrope, or "wheel of life." The zoetrope has a short, fat cylinder which rotated on its axis of symmetry. Around the inside of the cylinder were a sequence of drawings, each one slightly different from the one next to it. The cylinder had long slits cut into its side in between each of the images so that when the cylinder was spun a slit would allow the eye to see the image on the oppose wall of the cylinder. As the cylinder was spun on its axis, the sequence of slits passing in front of the eye would present a sequence of images to the eye, creating the illusion of motion.

Another low-tech animation piece of equipment was the flipbook. The flipbook was a tablet of paper with an individual drawing on each page so the viewer could flip through them. This was also popular in the 1800s.

The earliest hint of using a camera to make lifeless things appear to move was by Meleis in 1890 using simple tricks. The earliest pioneers in film animation were Emile Cohl, a Frenchman who produced several vignettes, J. Stuart Blackton, an American, who actually animated 'smoke' in a scene in 1900 and who is credited with the first animated cartoon in 1906, and the first celebrated animator, Winsor McCay, an American best known for his works Little Nemo and Gertie the Dinosaur.

The first major technical developments in the animation process can be traced to the work (and patents) of John Bray starting in 1910. His work laid the groundwork for the use of translucent cels (short for celluloid) in compositing multiple layers of drawings into a final image as well as the use of grey scale (as opposed to black and white) drawings. Later developments by Bray and others enhanced the overlay idea to include multiple translucent pieces of celluloid (cels), added a peg system for registration, and the drawing of the background on long sheets of paper so that panning (translating the camera parallel to the plane of the background) could be performed more easily. Fleischer patented rotoscoping in 1915. Rotoscoping is drawing images on cells by tracing over previously recorded live action. Bray experimented with color in one 1920 short.

During this time, animation as an art form was still struggling. The first animated character with an identifiable personality is Felix the Cat by Otto Messmer which appeared in the early 1920s in Pat Sullivan productions. In the late 1920s however, new forces had to be reckoned with: sound and Walt Disney.

5.28 DISNEY

Walt Disney was, of course, the overpowering force in the history of animation. Not only did his studio contribute several technical innovations, but the Disney studio, more than anyone else, advanced animation as an art form.

Some of Disney's innovations in animation technology were the use of a storyboard to review the story, the use of pencil sketches to review motion, and the multi-plane camera

NOTES

stand In addition, Disney pioneered the use of sound and color in animation (although not the first to use color). Disney also studied live action sequences to create more realistic motion in his films. When he used sound for the first time in Steamboat Willie (1928), he gained an advantage over his competitors.

Camera stand animation is more powerful than you might think. A camera stand allows the parallax effect - moving of backgrounds at different rates as the observer pans across an environment to create the illusion of depth - and zooming. Each of the planes can move six directions (right, left, up, down, in, out) as well as the camera moving in and out. By keeping the camera lens open during movement, figures can be made to appear extruded into shapes of higher dimension, simulate motion blur, exhibit depth attenuation.

With regard to the art form of animation, Disney perfected the ability to impart unique, endearing personalities in his characters including Mickey Mouse, Pluto, Goofy, the three little Pigs, and the seven Dwarfs. He also developed mood pieces of animation including Skeleton Dance and Fantasia. He also promoted the idea that the mind of the character was the driving force of the action and that a key to believable animated motion was the analysis of real life motion.

5.29 OTHER MEDIA FOR ANIMATION

Computer animation is considered by many to be actually closer to other animation techniques rather than traditional hand-drawn animation. Often it is compared to stop motion animation, such as puppet animation, that builds and manipulates identifiable objects. Other stop motion techniques are claymation, pinhead animation and sand animation.

In this type of animation, a physical object is manipulated, the camera takes an image of it, the object is manipulated again, another image is taken of it, and the process repeats to produce the animated sequence.

5.30 ANIMATION PRODUCTION

Computer animation production has borrowed most of the ideas from conventional animation production including the use of a story board, test shots, and pencil testing. The use of key frames and in-betweening have also been adopted in certain computer animation systems.

Story boards have pretty much translated directly over to computer animation production although they may be kept on a computer. They still hold the same functional place in the animation process.

In computer animation there is usually a strict distinction between the creation of the models, the specification of their motion, and the rendering process which is applied to those models. In conventional animation, the model building, motion specification, and rendering are really all the same thing. In computer animation, speed-quality tradeoffs can be made in each of the three stages during the trial and error process that characterizes much of animation.

A test shot in computer animation is usually a high quality rendering of a highly detailed model to see a single frame of the final product. Pencil testing can be performed either by simplifying the sophistication of the models used, or by using low quality and/or low resolution renderings, or by using simple motion control algorithms. Place

holder cubes can be rendered in wire frame to present the gross motion of rigid bodies in space and to see spacial and temporal relationships among objects. This may also provide real-time calculation and playback of the animation. Similarly, high-quality rendering of low complexity models or low-quality renderings of highly detailed models, or some intermediate levels of both the model and the renderer can be used to give the animator clues to the finished product's quality without committing to the final computation. For example, a hardware z-buffer display can provide good turn-around time with decent quality images before going to a ray traced image. Solids of revolution objects lend themselves quite well to allowing for three, four or five levels of detail for a given model. Also, smooth shading, texture mapping, specular reflection and solid texturing can all be options presented to the animator for a given run. To simplify motion control, for example, simple interpolation between poses may be used instead of inverse dynamics.

NOTES

5.31 TIME

When choreographing action in an animation or other production, the precise specification of time and temporal relationships is important. In everyday life, time is usually considered to be a continuous axis. In animation, however, because of the ultimate breakdown into frames, the time axis is discrete. It is important to keep this in mind when discussing temporal relationships of events in the animation production.

Because of the discrete nature of frames, there may be some ambiguity when saying, for example, a span ends exactly when another span begins. In the discrete world, we will take that to mean that a span ends on the frame before the frame that the other span begins on.

5.32 COMPUTER ANIMATION: FILMS AND VIDEOS

Early computer animation companies included Mathematical Applications Group, Inc. (MAGI), Information International Inc. (III, or Triple-I), Digital Productions, Digital Effects, Image West, Robert Abel and Associates, and Cranston-Csuri.

Current computer animation companies (who also contribute significantly to research in the area are: Pixar, Industrial Light and Magic (ILM), Pacific Data Images (PDI), Disney, Xaos, Rhythm & Hues, Digital Domain, Lamb & Company, Metrolight Studios, Boss Film Studios, deGraf/Wahrman, R/Greenberg Associates, Blue Sky Productions, Sony Pictures, Cinesite, Imageworks, and Apple. (One might also include Silicon Graphics, in that list since their equipment is used so extensively for computer animation).

Early on, Computer Graphics (CG) appeared in a variety of movies in which it was used as computer graphics (that is, the CC was not intended to fool the audience into thinking it was anything other than CG). For example, *Future World* (1976) and *Star Wars* (1977, Image West) fall into this category. More recently, *Lawnmower Man* (1992, Xaos, Angel Studios) which has a segment of Hollywood's view of Virtual Reality in it, used CG in the same role, although a more sophisticated example.

Tron (1982, MAGI) was a little different. The CG was still supposed to be computer-like because the action takes place inside a computer. But in this case, it was an integral part of the environment that the actions (and actors) were taking place in. The CG was used throughout the movie. It integrated computer animation with live action, but, since the action took place in a computer, the CG didn't have to look realistic (and didn't). This was the first time CG was used as an integral part of a movie.

NOTES

Along the same lines of *Tron* in using CG to create an 'inside the computer' environment is *Reboot* (1995, Limelight Ltd./BLT Productions). *Reboot* deserves special mention as the first Saturday morning cartoon that is full three-dimensional computer-generated animation. The action takes place inside a computer so they don't have to go for 'realism'. Still, there are several human-like main characters and, overall, The *Reboot* series is very impressive.

One main use of CG has been to replace physical models. In this case, CG is used to create realistic elements which are intermixed with the live action. *The Last Star Fighter* (1984, Gray Demos' Digital Productions, even the Cray X-MP was in the credits) used computer animation instead of building models for special effects. The action takes place in space as well as on planets; CG was used for the scenes in space and physical models were used for the scenes on a planet. It's not hard to tell when the movie switched between CG models and physical models. There were probably 20 minutes of CG used in the movie. This was the first time CG was used as part of the live action in which it wasn't supposed to look computer generated. More recently, *Apollo* 13 (1995, Digital Domain) used CG models of the return vehicle from the mission. In TV-land, special note should go to *Babylon 5* (1995, Newtek). *Babylon 5* is the first TV show to routinely use CG models as regular features of it's sci-fi show - and it's Amiga-based.

CG is also used to create 'alien' creatures. Creatures which are supposed to be realistic, but don't have to match anything that the audience is familiar with. *The Abyss* (1989, ILM) is one such movie in which CG is used to effect an alien creature which is integrated with the rest of the live action. Some of the CG in *Terminator II* served a similar purpose as well as *Casper* (1995, ILM), *Species* (1995, Boss Film Studios), *Mighty Morphin' Power Rangers* (VIFX), *Sexy Robot* (TV Commercial, Abel).

More challenging is the use of CG to create realistic models of creatures that are familiar to the audience. *Jurassic Park* (1993, ILM) was the first to completely integrate use of CG character animation in which the graphics were designed so as to blend in with the live action so that it was difficult to tell what was computer generated and what wasn't. *Jumanji* (1995, ILM) does the same thing with it's incredible modeling of animals. To a lesser extent, *Batman Returns* (1995, Digital Domain) also does the same thing by providing 'stunt doubles' of Batman in a few scenes. CG was used to create the face of *RoboCop 2* (1990, deGraf/Wahrman) and animated skeletons in *Total Recall* (Metrolight) as well.

Another popular CG technique for special effects is the use of particle systems. One of the best examples is in *Star Trek II: The Wrath of Khan* (LucasFilm computer division, later ILM) in which a wall of fire sweeps over the surface of a planet. Another example is *Lawnmower Man* in which a character disintegrates into a swirl of small balls. A more recent example from television is in the opening sequence of *Star Trek: Deep Space Nine* (1995) to model a comet's tail. *Twister* also uses particle systems to simulate a tornado.

Of course, one use of computer animation is simply to 'do animation.' By that I mean computer animation is used to produce animated pieces which would otherwise be done by more traditional means - essentially 3-D cartoons (although the term cheapens the idea somewhat). Although I'd like to restrict my topic to 3-D computer animation, I suppose that morphing should be mentioned. This is essentially a 2-D procedure which warps control points (or feature lines) of one image into the control points (feature lines) of another image while the images themselves are blended. In Star Trek IV, one of the first commercial morphs was provided by ILM in the back in time dream sequence In *Willow* (1988, ILM), ILM provided the morph of several animals. This technique was also used by ILM in *Indiana Jones and the Last Crusade* (1989) and *Terminator 2*. PDI is known for its use of morphing in various commercials including a Plymouth Voyager commercial and an Exxon commercial in which a car changes into a tiger. Of course,

NOTES

morphing has gone on to become another Energizer Bunny of TV commercials - it keeps going and going and going... Full 3-D morphing has yet to make it out of the research labs and into any production environment.

There is another class of movies in which CG plays a role - that of 'hidden special effect' (for lack of a better term). CG can be used to cover up mechanical special effects or to enhance the scene to integrate a mechanical special effect more completely. For the most part, this resides in the 2-D realm and, as such, will not be the focus of this document. However, with the onset of digital techniques for 2-D compositing, sequences will be routinely digitally available making them susceptible to a variety of digital post-processing techniques. The first digital blue screen matte extraction was in Willow (ILM). The first wire removal was in Howard the Duck (ILM). In *True Lies* (1994, Digital Domain), CG was used to erase support wires from suspended actors. In *Forest Gump* (1994, Digital Domain), CG was used to insert a ping pong ball in a sequence showing an extremely fast action game. In *Babe* (1995, Rythm & Hues), CG was used to move the mouths of animals and fill in the background uncovered by the movement. In *Interview with a Vampire* (1994, Digital Domain), CG was used to curl the hair of a woman during the transformation into a vampire. In this case, some of the effect was created using 3-D graphics and then integrated into the scene by 2-D compositing.

5.33 COMPUTER ANIMATION SOFTWARE

Here is a short list of some animation software.

GIFF Animator
Macromedia Flash
Softimage (Microsoft)
Alias/Wavefront (SGI)
3-D Studio MAX (Autodesk)
Lightwave 3-D (Newtek)
Prisms 3-D Animation Software (Side Effects Software)
HOUDINI (Side Effects Software)
Apple's toolkit for game developers
Digimation

Here we are going to discuss **Macromedia Flash** and **GIF Animator**

5.34 MACROMEDIA FLASH

Macromedia Flash is quickly becoming the standard for multimedia development for the web and the desktop. Flash is an extremely powerful drawing or illustration program, featuring all the creation and manipulation features you could want. You can introduce objects, move objects, and even morph objects over time. Flash allows you to easily incorporate text, graphics, sound, and animation into a compressed package that can be sent over the web or played from the desktop.

The core elements of the interface can be described in terms that lend themselves to a movie analogy. The desktop is the "stage," and the unfolding of your creation takes place along a timeline that is divided into "frames." You can set an object to exist for a specific number of frames, and you can assign actions (start/stop) to a frame. Larger subdivisions of the file are called "scenes." Flash provides tools for sketching and layout in a process called "storyboarding."

Objects are controlled using a series of "panels" that group the functions you might want to perform on an object you've created. The **Transform** panel, for example, lets you rotate, skew, scale, and duplicate elements. The **Align** panel controls the alignment, distribution, sizing, and spacing of objects. With the **Mixer**, you can define colors. Using the **Character** panel, you can edit fonts characteristics such as font size, font color, and kerning, as well as set links to text. Similar panels are available to create and control color and sound.

NOTES

Some key concepts behind the features:

- The range of options for object manipulations is due to the program's ability to treat objects as either bitmap or vector, depending on the operation you want to perform. Imported bitmap art can be converted to a collection of vector drawings for extraction and further manipulation to suit the purposes of the presentation.
- A "key frame" is a space in the timeline where you can define the properties of an object (shape, color, transparency, position, etc.) for a chosen time duration. Another function of key frames is to introduce a layer in a timeline relationship to active layers. Different key frames can exist in different layers, but you need to line up key frames to give the appearance of motion and movement.
- A major consideration in developing web-based communication vehicles is the time required for downloading. This becomes crucial when using motion. The library feature of Flash serves to reduce the file size of the final product, because objects can be reused many times in the movie, but are downloaded only once. You can edit one instance of a library item and apply the change to the library item or leave the library item alone. Objects must be placed in a library before you can apply animation.
- Download time can also be saved by manipulating objects in layers so that part of an object or group is covered by another. Flash stores only the content that's visible at the surface.
- "Tweening" allows you to select two frames and then morph an object from one shape to another between the frames.

5.35 GIFF ANIMATOR

Key Features

Fast and easy animation

1. Intuitive Interface: Get around fast using a tab-based interface
2. Easy composition: Create dynamic multiple object animation with drag and drop precision
3. Dynamic effects: Apply various text effects, video effects, transitions etc.
4. Powerful optimization: Ensure fast loading animation with the latest image compression techniques
5. Flexible output: Export to a wide variety of file formats, including Flash, AVI, MPEG etc.

System Requirements for Giff Animator

- Intel Pentium compatible processors
- Microsoft Windows 98, NT 4.0 SP5 or higher
- 64 MB of RAM
- 20 MB of available hard disk space

- CD-ROM drive
- True color or Hicolor display adapter
- Windows compatible pointing device.

5.36 SOUND ANIMATION

NOTES

Special bonds connect sound with moving images. In animation, these bonds are very special. The visual elements and the audio track seem to share a more intimate and more creative partnership than exists in other motion picture form.

This may be created by :

- **High degree of synchronization**

  The technology of animation encourages a higher degree of synchronization that is practical within other forms of moviemaking. In both visual and audio realms, the animator has total control. An image or a sound can be placed with accuracy down to the tenths of a second.

  [read about the Synchresis phenoma]

- **Strong relationship between animation and music**

  Animation and music have a basic mathematical foundation and move forward at a determined speed. The rhythm of a musical composition is measured, and beats are fitted into bar units of defined time length and are interpreted in time units.

- **Sound metaphors**

  The combination of fast moving animated visuals and unrealistic sound effects create audiovisual metaphors that often have humorous aspects. For example slow moving footsteps synched to cymbal crashes ... or ... fast moving hands digging to the sound of a roaring engine.

- **Kinetic energy**

  In animation synchronized sound effects are used as a source of kinetic energy.

  In the Classic Hollywood Studio Cartoon the sound track are mostly constructed from boinks, petangs, pings, and kerthuds to energize a scene.

  When these sounds are determinants of the scene's pacing, they are recorded prior to shooting and logged like a voice track; the animation is shot to the effects track.

Animation Sound can be...

- Pre-synchronous
- Post-synchronous
- Non-synchronous.

Pre-synchronous

Music and sound effects for key actions is recorded before the images are produced and the animation is "shoot to the track" The sounds are created "presynch".

Most voices are recorded presynch. Presynched is the best way to achieve the precise synchronization between pictures and sound that, for audience, is one of the most entertaining aspects of animation. For the animator, presynching also helps solve one of the most difficult problems encountered during shooting: how to pace action, or more precisely, how to determine the extent of each incremental movement. Once a voice or music track has been recorded, analyzed, and logged, it becomes a concrete determinant of when key actions must occur relative to those points.

Dramatic important sound are logged as voice track prior to shooting.

Post-synchronous

The images are shoot before the sounds.

Sound effects are used to complete the outer orientation (the spatial and temporal settings) established by the visuals. These sound effects are often added in postproduction, since they are rarely the primary determinant of the scene's pacing. If the animator is shooting to the voice track, particular actions in the picture that require an sound effect – such as doors closing, gunshots, telephones ringing, and so forth – are simply cut into the effects track at the appropriate frame.

NOTES

Non-synchronous

Non-synchronous music has not been carefully timed to fit the picture. Because music and film both are "time arts", it is inevitable that any selection will synchronize with the picture at random points, and even non-synchronous music severs as a "bed" for the action throughout.

In Animation Actions are Condensed

Standard libraries of sound effects are more and then useless for animation because the action of animation is for compact - a real car takes ages to go into the distance compared with the animation of the same action.

First Principle of Animation Sound

First Principle of animation sound effects is that there is no connection between the object in the picture and the origin of the sound.

SUMMARY

- A major part of rendering (making images more realistic) is the visible surface problem. (*i.e.*, only display those surfaces which should be visible.
- Hidden Surface Algorithms are usually image space or a combination of object and image space.
- Object Space method is implemented in the physical device coordinated system in which objects are described.
- Image space method is implemented in the screen coordinate system in which the objects are viewed.
- Two basic approaches used for visible surface algorithm detection which are as follows:-
  1. Object Precision Algorithm
  2. Image Precision Algorithm.
- Coherence is based on the principle of locality, whereby "nearby" things do have the same or similar characteristics.
- With scan-line algorithms an image is generated sequentially, one scan line at a time. Spans are portions of a scan line with some constant property, *e.g.*, the same object is visible over a given span.
- The curve in general does not pass through any of the control points except the first and last. From the formula $B(0) = P_0$ and $B(1) = P_N$.
- The curve is always contained within the convex hull of the control points, it never oscillates wildly away from the control points.
- If there is only one control point P_0, *i.e.*,: $N = 0$ then $B(u) = P_0$ for all u.

- If there are only two control points P_0 and P_1, *i.e.*: N = 1 then the formula reduces to a line segment between the two control points.

$$B(u) = \sum_{k=0}^{1} p_k \frac{1}{k!(1-k)!} u^k (1-u)^{1-k} = P_0 + u(P_1 - P_0)$$

- The term

$$\frac{N!}{k!(N-k)!} u^k (1-u)^{N-k}$$

is called a blending function since it blends the control points to form the Bézier curve.

- The blending function is always a polynomial one degree less than the number of control points. Thus 3 control points results in a parabola, 4 control points a cubic curve etc.
- Closed curves can be generated by making the last control point the same as the first control point. First order continuity can be achieved by ensuring the tangent between the first two points and the last two points are the same.
- Except for the redundant cases of 2 control points (straight line), it is generally not possible to derive a Bézier curve that is parallel to another Bézier curve.
- A circle cannot be exactly represented with a Bézier curve.
- It isn't possible to create a Bézier curve that is parallel to another, except in the trivial cases of coincident parallel curves or straight line Bézier curves.
- Special case, 3 control points

$$B(u) = P_0 * (1-u)^2 + P_1 * 2 * u(1-u) + P_2 u^2$$

- Special case, 4 control points

$$B(u) = P_0 * (1-u)^3 + P_1 * 3 * u * (1-u)^2 + P_2 * 3 * u^2 * (1-u) + P_3 * u^3$$

Bézier curves have wide applications because they are easy to compute and very stable. There are similar formulations which are also called Bézier curves which behave differently, in particular it is possible to create a similar curve except that it passes through the control points. See also Spline curves.

- Fractals stands for fractional dimensional and is a term widely associated in graphics with randomly generated curves and surfaces. They are used to provide materialistic shapes for representing objects such as coast lines, rugged mountains, grass etc.
- Animation is the creation of the illuion of movement by assembling a sequence of still images. To animate is 'to give life to'. The illusion of movement is caused by physiological affect known as "PERSISTENCE of version". We are concerned with computer generated animation.
- There are softwares for animation like Giff Animation and Macromedia Flash. They have been discussed in the chapter.

NOTES

REVIEW QUESTIONS

1. Define Hidden Surface Algorithms.
2. What are the Object Space and Image Space methods?
3. Define the techniques for Efficient Visible-Surface Algorithms.
4. Define coherence and types of coherence.
5. Define Spatial Partitioning.
6. Define back face removal.
7. Define Z-Buffer algorithm and Painter's algorithm.
8. Define Warnock Area Subdivision Algorithm.
9. What is Binary Space Partitioning?
10. Define Binary Space Partitioning (BSP) tree.

NOTES

11. Given three control points on the *xy*-plane (– 1,0), (0,1) and (2,0), do the following:
    - Write down its Bézier curve equation.
    - Expand this equation to its equivalent conventional form.
    - Since there are three control points, there are three Bézier coefficients. Write down their equations and sketch their graphs.
    - Use your calculator to find enough number of points using the conventional parametric form and sketch the curve.
    - Find points on the curve that correspond to $u = 0, 0.25, 0.5, 0.75$ and 1 with the conventional form.
    - Use de Casteljau's algorithm to find points on the curve corresponding to $u = 0, 0.25, 0.5, 0.75$ and 1.
    - Subdivide the Bézier curve at $u = 0.4$ and list the control points of the resulting curve segments.
    - Increase the degree of this curve to three and list the new set of control points. Then, increase the degree to four and list the new set of control points.
12. In the variation diminishing property, what if you have a line or a plane that passes through a control point or contains a line segment of the control polyline? Suggest a proper counting of intersection points and verify your claim with examples.
13. A Bézier curve of degree 2 defined by three control points $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$ is a portion of a conic section. What type of this conic section is it? Is it a portion of a parabola, a hyperbola or an ellipse? You can assume the given control points are in the *xy*-coordinate plane.
14. Suppose Bézier curve $C(u)$ (*resp.*, $D(u)$) of degree n is defined by control points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$ (*resp.*, $\mathbf{Q}_0, \mathbf{Q}_1, \ldots, \mathbf{Q}_n$). If the curves are identical (*i.e.*, $C(u) = D(u)$ for every u in $[0,1]$), then the corresponding control points are also identical (*i.e.*, $\mathbf{P}_i = \mathbf{Q}_i$ for all $0 <= i <= n$).

    Hint: First show that if $(1 - u)\mathbf{A} + u\mathbf{B}$ is a zero vector for every u in $[0, 1]$, then $\mathbf{A}$ and $\mathbf{B}$ are both zero vectors. Then, work the de Casteljau's algorithm backward to show that $\mathbf{P}_i - \mathbf{Q}_i$ is a zero vector for all $0 <= i <= n$.
15. Suppose Bézier curve $C(u)$ of degree n is defined by control points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$.
    1. Prove the following:

$$B_{n,i}(u) = \sum_{j=i}^{n} (-1)^{j-i} C(n,i) C(n-i, j-i) u^j$$

    2. Show that curve $C(u)$ can be rewritten to the following *matrix* form:

$$C(u) = \left[\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n\right] \cdot \begin{bmatrix} m_{00} & m_{01} & \cdots & m_{0n} \\ m_{10} & m_{11} & \cdots & m_{1n} \\ \vdots & & \ddots & \vdots \\ m_{n0} & m_{n1} & \cdots & m_{nn} \end{bmatrix} \cdot \begin{bmatrix} u^0 \\ u^1 \\ \vdots \\ u^n \end{bmatrix}$$

    where entry m_{ij} is defined as follows:

$$m_{ij} = \begin{cases} (-1)^{j-i} C(n,i) C(n-i, j-i) & \text{if } j \geq i \\ 0 & \text{otherwise} \end{cases}$$

16. Therefore, a Bézier curve can be rewritten using the traditional polynomial form in $u^0 = 1, u^1, u^2, \ldots, u^n$. This is the so-called *monomial form* and the basis functions are $u^0 = 1, u^1, u^2, \ldots, u^n$. However, the use of this monomial form is computationally unstable.
17. Show that the maximum of $B_{n,i}(u)$ occurs at $u = i/n$ and that the maximum value is

$$\frac{n!}{i!(n-i)!} \frac{i^i (n-i)^{n-i}}{n^n}$$

18. Verify the following results with your calculus knowledge:
    - the derivative of $B_{n,i}(u)$:

$$\frac{d}{du}B_{n,i}(u) = B'_{n,i}(u) = n\left(B_{n-1,i-1}(u) - B_{n-1,i}(u)\right)$$

    - the derivative of Bézier curve $p(u)$:

$$\frac{d}{du}C(u) = C'(u) = \sum_{i=0}^{n-1} B_{n-1,i}(u)\left\{n\left(P_{i+1} - P_i\right)\right\}$$

19. The discussion of joining two Bézier curves with C^1-continuity assumes the domain of the curves is [0, 1]. Suppose the domain of the first curve is [0, s] and the domain of the second curve is [s, 1]. Redo the calculation. What is your conclusion? Is there any modification required?

20. Prove the following:

$$D_i^k = \sum_{j=0}^{k} (-1)^{k-j} C(k, j) P_{i+j}$$

where D_i^k's are the k-th difference points and $C(k, j)$ is the binomial coefficient defined as follows:

$$C(k, j) = \frac{k!}{j!(k-j)!}$$

With this formula, we can express a higher derivative using the original control points rather than using finite difference points.

21. After subdividing a Bézier curve of degree p at s, we have two Bézier curves of degree p, one on interval [0, s] while the other on [s, 1]. Show that these two curves are of C^1 continuous at the joining point.

    Hint: Suppose the last two control points of the curve on [0, s] are P_{p-1} and P_p, and the first two control points of the curve on [s, 1] are Q_0 and Q_1. Then, we have P_{p-1}, $P_p = Q_0$ and Q_1 are on the same line, and the ratio of the distance from P_{p-1} to $P_p = Q_0$ and the distance from $P_p = Q_0$ to Q_1 is equal to s due to subdivision. Now, change the variables of both curves so that they have domain on [0, 1]. A simple calculation will lead to the desired conclusion.

22. Explain the steps used in Animation process.
23. Write a program to generate Beizer curves.
24. Write a program for generation of fractals.
25. Write short notes on :
    (a) Persistence of version
    (b) Animation and Movies
    (c) Animation Hardware
    (d) Multimedia
    (e) Fractals.

FURTHER READINGS

- Computer Graphic: V.K. Pachghare, Laxmi Publications, 2007, Second edition.
- Computer Graphics: Prabhakar Gupta, Vineet Agarwal and Manish Varshney, Laxmi Publications, 2011.
- Computer Graphics: Rajiv Chopra, S. Chand Publisher, 2011.
- Computer Graphics: C.S. Verma, Ane Books, 2011.
- Computer Graphics: Pradeep K. Bhatia, I.K. International, 2009, pbk, Second Edition.
- Computer Graphics: Ruchi Mishra, Global Vision Publisher, 2010.