Second Edition

TU, PU, PoU, KU

Insights on COMPUTER GRAPHICS

Written by:

Er. Shree Krishna Sulu

Edited by:

Er. Sujan Shrestha

Insights on

COMPUTER GRAPHICS

Written by
Er. Shree Krishna Sulu
(Associate Prof., Kathmandu Engineering College)

Edited by
Er. Sujan Shrestha
(Lecturer, Kathmandu Engineering College)

SYSTEM INCEPTION
Koteshwor, Kathmandu

Insights on

COMPUTER GRAPHICS

Published by : **SYSTEM INCEPTION**

Written by : *Er. Shree Krishna Sulu*

Edited by : *Er. Sujan Shrestha*

First edition : 2019 AD

Second edition : 2020 AD

Computer : Creation Graphics
Bagbazar, Kathmandu

PREFACE TO THE SECOND EDITION

"Insights on Computer Graphics" is a textbook of computer graphics for the students of Bachelor level in Electronics and Communication, and Computer Engineering. Computer graphics is widely used in almost all aspects of the life; entertainment to medical treatment, art and commerce to office automation, business visualization to scientific research, computer aided design to virtual reality. And, the daily use of smart mobile, tab, laptop, digital display, smart TV, etc. is tremendously increasing. So, the study and field of computer graphics is widening day by day. The concept and principle of computer graphics systems have been explained step by step in this book in order to make easier to understand.

A large number of numerical of previous IOE exam questions have been solved to develop an efficient understanding of the related topics. The Code in C of some algorithms is also depicted to make clear about the programming in Computer Graphics.

I would like to thank those teachers and students (especially Shyam Dahal and Shiva Ram Sulu) whose inspiration assisted me in bringing this book.

With the hope, students and concerned person will be benefitted, I pray for a prosperous and peaceful Nepal.

Er. Shree Krishna Sulu
Feb. 2020

CONTENTS

Chapter – 1

Introduction and Application

1.1 Introduction

Graphics is an image or a visual representation of an object and the visual representation or image displayed on a computer screen is known as computer graphics. More precisely, computer graphics is the field or branch of science and technology related to generation (creation), storage and manipulation of graphics (images or pictures) of objects using computer i.e. using hardware and software. Objects may be the concrete real world objects or the abstract and synthetic objects such as mathematical surface, engineering structure, architectural design, survey results, etc. In other word, graphics means to plot some points on graph to make an image. Computer graphics means to plot some pixels (points) on a computer screen to make an image. Pixel or picture element is the elementary part of the computer screen. We see every day the images created by using computer in books, magazines, movies, TV, etc.

Computer graphics is the rendering (servicing or making) tools for the generation and manipulation of images. These tools include both hardware and software.

Hardware comprises monitor, printer, plotter (that display graphics) and input devices includes mouse, light pen, keyboard, scanner, etc. Software tools refer to the collection of graphics routine.

Computer graphics = Data structure + Graphics algorithm + Language

Data structure means those data structure that are suitable for computer graphics. Graphics algorithm refers to algorithm for picture generation and transformation. Language means high level language for generation of graphics or pictures of objects.

Computer graphics can be either two-dimensional or three-dimensional. Digital graphic files are divided into two categories:

i. Raster graphics
ii. Vector graphics

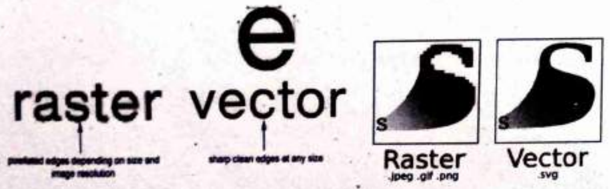
1.2 Raster Graphics

A raster graphic or image is made up of pixels (screen point). Raster graphics are composed of a simple grid of pixels. The pixel can be of different color. Raster graphics are rendered images on a pixel-by-pixel basis and they are well fit when handling shading and gradients. A raster graphic, such as a gif or jpeg, is an array of pixels of various colors, which together form an image. Raster graphics are the most common and are used for digital photos, web graphics, icons, and other types of images.

When a raster image is scaled up, it usually loses quality. A raster image can be enlarged by either adding more pixels or enlarging the size of the pixel. In either way original data is spread over a larger area at the risk to losing clarity.

Raster graphics based file format:

.jpg	Joint Photographic Experts Group (JPEG)
.png	Portable Network Graphics (PNG)
.gif	Graphics Interchange Format (GIF)
.tiff	Tagged Image File Format (TIFF)
.psd	Adobe Photoshop File
.pat	Corel Paint File
.pdf	Portable Document Format (PDF)


raster
vector
pixelated edges depending on size and image resolution
sharp clean edges at any size
Raster
.jpeg .gif .png
Vector
.svg

(a) *(b)*

***Figure 1.1:** Raster and vector graphics*

Advantages of raster graphics

Every pixel in a raster image can be of different color therefore we can create a complex image with any kind of color changes and variations. Raster graphics are useful for creating rich and detailed images.

Almost any program can work with a simple raster file. The most recognized application that handles raster graphics is Adobe Photoshop however there are also several other image editing software options out there to choose from.

Disadvantages of raster graphics

It looks grainy, distorted, and blurred when raster images are scaled up. This is because raster images are created with a finite number of pixels. When we increase the size of a raster image, the image will increase in size. However, because there are no longer enough pixels to fill in this larger space, gaps are created between the pixels in the image.

Raster images hold more data and may be slower to edit. Raster files are often quite large. Raster files contain all the information for every single pixel of the image. Each of these pixels has an X and Y coordinates as well as color information associated with it. Therefore, raster graphics files tend to be very large.

Raster graphics are not great for embroidery. Because raster images are based on square pixels, the embroidery may look like it has jagged edges. If we want to embroider an image with smoother edges, it is best to use vector graphics instead of raster graphics.

1.3 Vector Graphics

Vector graphics (also called geometric modeling or object oriented graphics) is made up of geometrical primitives such as points, lines, curves, and polygons which are all based upon mathematical equations to represent images in computer graphics. A vector graphic can be scaled to any size without losing quality.

Vector graphics are composed of paths which may be lines, shapes, letters, or other scalable objects. A vector graphic, such as an .eps file or Adobe Illustrator file, is composed of paths, or lines, that are either straight or curved. The data file for a vector image contains the points where the paths start and end, how much the paths curve, and the colors that either border or fill the paths. Vector graphics are not made of pixels, the images can be scaled to be very large without losing quality. They are often used for creating logos, signs, and other types of drawings. Unlike raster graphics, vector graphics can be scaled to a larger size without losing quality. Raster image's dimensions are measured in pixels (pixel per inch-ppi). Vector graphics are resolution independent.

The most recognized applications which handle vector based graphics are Adobe illustrator, Macro media freehand and Corel draw. Vector graphics are generally used for line art, illustrations, and embroidery.

All modern current computer video displays translate vector representation of an image to a raster format. The raster image, containing a value for every pixel on the screen, is stored in memory.

Probably the most common example of vector-based files that we use daily without even realizing is font files.

Vector graphics based file format:

.eps	Encapsulated PostScript File (EPS)
.svg	Scalable Vector Graphics (SVG)
.ai	Adobe Illustrator File
.cdr	Corel Draw File
.pdf	Portable Document Format (PDF)

Advantages of vector graphics:

Vector files are small because they contain a lot less data than raster files. Vector graphics are more flexible than raster graphics because they can be easily scaled up and down without any loss to the quality of the image. Vector graphics have smoother lines in comparison to raster graphics.

Disadvantages of vector graphics:

If there are small errors or faults in a vector graphic, these will be seen when the vector image is enlarged significantly. Vector graphics can't display the abundant color depth of a raster graphic.

1.4 Computer Graphics and Image Processing

The difference between computer graphics and image processing can be studied with the help of following table.

Computer graphics	Image processing
1. It is the field related to the generation of pictures using computers.	1. It applies technique to modify or interpret existing pictures.
2. It synthesizes pictures from mathematical or geometrical models.	2. It analyzes picture to derive description in mathematical or geometrical forms.
3. It includes the creation, storage, and manipulation of images of objects	3. It is the part of computer graphics that handles image manipulation or interpretation
4. E.g., drawing a picture	4. E.g., making blurred image visible.

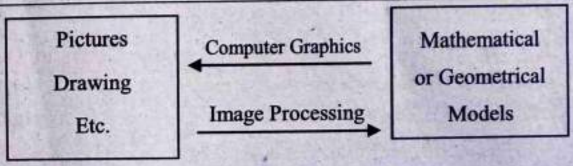


Figure 1.2: Computer graphics and image processing

1.5 History of Computer Graphics

- In 1950's outputs were via teletypes, line printer and Cathode Ray Tube (CRT). Using dark and light character, pictures were reproduced.
- In 1950, Been Laposky created the first graphic images on oscilloscope generated by an electronic (analog) machine. The image was produced by manipulating electronics beams and recording them onto high speed film.
- In 1951, UNIVAC-I, the first general purpose commercial computer; crude hardcopy devices, and line printer were invented. MIT-whirlwind computer, the first to display real time video and capable of displaying real time text and graphic on a large oscilloscope screen was developed.

- In 1960's, modern interactive graphics was begun. Outputs were vector graphics and interactive graphics, but the problems were the cost and inaccessibility of machine. In middle 1950's SAGE (Semi-Automatic Ground Environment) air defenses system was developed. It was the first to use command and control CRT display consoles on which operator identifies target, with light pen (hand –held pointing devices that senses light emitted by objects on screen)
- In 1960, William Fetter coined the computer graphics to describe new design methods.
- In 1961, Stove Russel made space wars, first video/computer game
- In 1963, Douglas Englebart developed first mouse. Ivan Sutherland made sketch pad, interactive C.G. System, a man-machine graphical communication system, it features: pop-up menus, constrained –based drawing, hierarchal modeling, utilized light pen for interaction
- He developed the dragging, rubber banding and transforming algorithms, He introduced data structures for storing. He is considered data founded of computer graphics.
- In 1964, William Felter developed first computer model of a human figure.
- In 1965, Jack Bresenham developed line drawing algorithm.
- In 1968, Tekronix made a special CRT, the direct view storage tube, with keyboard and mouse, a simple computer interface for $15,000 which made graphics affordable.
- In 1969, John Warnock developed area subdivision algorithm, hidden –surface algorithms. Bell labs developed first frame buffer containing 3 bits per pixel. CAD (computer aided design), CAM (computer aided manufacturing) with enormous potentials for automating drafting and other drawing –intensive activities, were developed. The general motors' DAC system for automobile design and the Itek Digitek system for lens design were pioneering efforts that showed the utility of graphical interaction in the interactive design cycles common in engineering.

  A number of commercial products using these systems were appeared. But hardware was expensive.
- In the early 1970's, Output start using raster displays, graphics capability was still fairly chunky.
- In 1972, Nolan kay Bushnell Pong made video arcade game.
- In 1973, John whitiney Jr. and Gary Demos made "Westworld", first film with computer graphics.
- In 1974, Edwin catmuff developed texture mapping and z-buffer hidden surface algorithm. James Blim developed curved surfaces, refinement of texture mapping.
- In 1971, Rendering model Gouraud shading was developed.
- In 1974 – 77, Phong shading (rendering model) was developed.
- In 1977, Steve Wozmak made Apple II, color graphics personal computer.
- In the 1980's, outputs were built in raster graphic, bitmap image and pixel, personal computers cost decrease drastically, track ball and mouse became the standard interactive devices.

  1980's, Artists and graphic designers preferred to use Macintosh and PC's.
- In late 1980's, Artists and graphic designers preferred to use Macintosh and PC.
- In 1982, Ray tracing (Illumination based rendering method) was developed. In 1982, Steven Lisberger made 'Tron', first Disney movie which made extensive use of 3-D computer graphics. John Walkner and Dan Drake developed Auto CAD
- In 1983, Jaron Lanier made 'Data Glove", a virtual reality film features a glove installed with switches and sensors of detection hand motion.

- In 1984, Wave from tech developed Polhemus, first 3D graphics software
- In 1987, IBM introduced VGA(Video Graphics Array)
- In 1989, Video Electronics Standard Association (VESA) formed SVGA (Super VGA)
- In 1990's, since the introduction of VGA and SVGA personal computer could easily display photo realistic images and movies 3D image rendering were become the main advances and it stimulated cinematic graphics application.
- In 1990's, since the introduction of VGA and SVGA, personal computer could easily display photo realistic images and movies. 3D image rendering were become the main advances and it stimulated cinematic graphics applications.
- In 1990, Render man system that provides fast accumulate and high quality digital computer effects was developed.
- In 1992, Silicon graphic developed open GL specification.
- In 1993, Mosaic, first graphic web browser and Jurassic park, a successful CG fiction film was made.
- In 1995, 'Toy story', first full length computer generated feature film was made.
- In 2003, ID software developed Doom graphics engine.

1.6 Uses of Computer Graphics

The main application areas of computer graphics are display of information (simple data to scientific visualization), design (graphic to industrial design), user interfaces (GUI to virtual reality), simulation (entertainment to academic purpose), etc. which can be synthesized as follows:

1. User interfaces
2. Plotting or visualization of measurement data
3. Office automation and electronic publishing
4. Computer aided design and drafting
5. Scientific and business visualization
6. Simulation and virtual reality
7. Entertainment
8. Art and commerce
9. Civil Engineering applications
10. Medical applications
11. Internet

1. User Interfaces

The interface between the human and the computer has been radically changed by the use of computer graphics. Most applications have a Graphical User Interface (GUI) for user friendly and interactive operation. Today's software has user interfaces that rely on the desktop window systems to manage multiple simultaneous activities and on point and click facilities to select items menu, icon, and objects on the screen.

2. Plotting or visualization of measurement data

Graphics is extensively used in plotting 2D and 3D graphs such as the histograms, bar and pie charts, the task scheduling charts of mathematical, physical, and economic functions. These plotting or visualization of measurement data are useful to analyze meaningfully and concisely the trend and pattern of complex data.

3. Office automation and electronic publishing

Nowadays, the necessary document that contains text, tables, graphs, drawings, pictures can be easily printed or saved as electronic (softcopy) document. The office automation and electronic publishing became possible by the development of computer graphics.

4. Computer aided design and drafting

A major use of computer graphics is in design process. Computer graphics is used to design components and systems (including building and other structural design) of architectural systems, mechanical, electrical, electrochemical and electronic devices, auto mobile bodies, aircrafts, watercrafts, spacecrafts, very large scale integrated (VLSI) chips, optical systems, and telephone and computer networks.

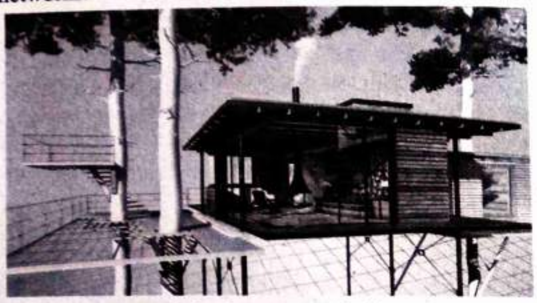


Figure 1.3: Computer aided design

5. Scientific and business visualization

Scientific visualization means generating computer graphics for scientific works and medical data sets. Business visualization is generating computer graphics for nonscientific data sets such as economic data set. Visualization makes easier to understand the trends and pattern inherent in huge amount of data sets. It would otherwise be almost impossible to analyze those data numerically.

6. Simulation and virtual reality

One of the most impressive and familiar uses of computer graphics is simulation and virtual reality. Simulation is the imitation of the conditions like those, which is encountered in real life. Virtual reality is an interactive computer-generated experience taking place within a simulated environment, that incorporates mainly auditory and visual, but also other types of sensory feedback like haptic. Simulation helps to learn or feel the condition one might have to face in near future without being in danger at the beginning of the course. Astronauts' simulator, flight simulator, military simulator, naval simulator, driving

simulator, air traffic control simulator, and heavy duty vehicle simulator are some of the mostly used simulators in practice.

7. Entertainment

Computer graphics is used in making games, special effects in movies, music videos, television shows, etc. Sometimes, the graphics scenes are displayed totally using computer graphics and sometimes, graphics objects are combined with the real actors and live scenes. Computer and video games such as FIFA, Formula-1, Doom and Pools are few to name where computer graphics is used extensively. Disney movies such as Lion King, The Beauty and the Beast, and other scientific movies like Star Trek are the best examples of application of computer graphics in the field of entertainment.

8. Art and commerce

Computer graphics are used in both fine art and commercial art. The ability to create any shape and play with any color with the help of computer graphics opened the new realm of art and commerce. Computer graphics is used to produce a picture that expresses a message and attract attentions such as a new model of a car moving along the ring of the Saturn.

These pictures are frequently seen at transportation terminal, super markets, hotel, etc. The slide production for commercial, scientific, or education presentation is another cost effective use of computer graphics. One of such graphics package is Power Point.

9. Civil engineering applications

Civil engineering applications include cartography, GIS (geographical information system), etc. Cartography is a subject which deals with the marketing of the maps and chart. Computer graphics is used to produce both accurate and schematically representation of geographical and other natural phenomenon from measurement data. It includes geographic map, oceanographic chart, weather map, color map, and population density map. Surfer is one of such graphic packages which in extensively used for cartography.

10. **Medical applications**

Computer graphics has become a powerful tool for diagnosis and treatment in medical fields. X-ray, video x-ray, complex operation, etc. are done using computer graphics method and techniques in medical field.

11. **Internet**

There is a large amount of multimedia content available on net. Internet became famous because of the development of computer graphics.

1.7 General Term and Terminologies

1. **Pixel (or pel)**

Pixel (picture element) is defined as the smallest screen element. It is the smallest piece of the display screen which can be controlled. The screen point is controlled by setting the intensity and color of the pixel.

2. **Aliasing**

Real objects or lines, polygon, edges are continuous but a raster device is discrete. The digitization of continuous signal produces jaggies i.e., a staircase problem. The sampling process digitizes the coordinate points and it produces staircase appearance. This process of distortion of information due to sampling is called aliasing.

3. **Antialiasing**

It is defined as the process which compensates the consequences of under sampling process.

4. **Pixel phasing**

It is a hardware based antialiasing technique in which the graphics system shifts individual pixels from their normal positions in the pixel grid by a fraction (typically 0.25 and 0.5) of the unit distance between the pixels. By moving pixels closer to the time line, this technique is very effective in smoothing out the stair steps without reducing the sharpness of the edges.

5. **Interlacing**

It is used when the perpetual threshold is greater than the frequency of standard line voltage. If refresh rate is greater than phosphor's persistence, then moving objects become blurred. If refresh rate is lesser than phosphor's persistence, then it creates flickering. Interlacing is used to break the raster line into two sweep patterns consisting of half the number of raster lines in original patterns.

6. **Bit depth (or color depth)**

It is defined as the number of bits needed (assigned) to a pixel in the image. It specifies the number of colors that a monitor can display. For example, if a pixel is denoted by a byte (8 bits), then the total number of color that can be displayed per pixel is $2^8 = 256$.

7. **Fluorescence and phosphorescence**

When the electron beam strikes the phosphor-coated screen of the CRT, some of this energy is dissipated as heat but the rest of energy is used to make the electron of the phosphor atoms to jump to higher quantum energy level. Fluorescence is the light emitted by very unstable electrons while the phosphor is being struck by electrons. Phosphorescence is the light given off by stable excited electron to their unexcited state once the electron beam excitation is removed. Fluorescence usually last for a fraction of microsecond. Most of the light emitted is phosphorescence.

8. **Persistence**

Persistence is how long phosphors continue to emit light (that is to have excited electron returning to ground state) after the CRT beam is removed. More precisely, persistence is the time to decay to 1/10th of its original intensity of the emitted light. That is, how long phosphorescence persists is the persistence. Lower persistence phosphors require higher refresh rates to maintain a picture on the screen without flicker. The phosphor with low persistence is useful for animation. A high persistence phosphor is useful to highly complex static pictures. Graphics monitors are usually constructed with persistence in the range from 10 to 60 micro second.

9. Refresh rate

The light emitted by the phosphor fades very rapidly. So, some method is needed for maintaining the screen picture. One way to keep the phosphor glowing is to redraw the picture repeatedly by quickly directing the electron beam back over the same points. Refresh rate is the number of times the image is redrawn per second to give a feeling of picture without flick. It is the frequency at which the content of the frame buffer is sent to the display monitor. Refresh rate is usually 50 frames per second. Refresh rate above which flickering stops is called critical fusion frequency (CFF). The factors affecting CFF are persistence, image intensity, ambient room light, and wave length of emitted light.

10. Horizontal scan rate

The horizontal scan rate is the number of scan lines per second. The rate is approximately the product of the refresh rate and the number of scan lines.

11. Resolution

Resolution is the maximum number of pixels (points) that can be displayed horizontally and vertically without overlap on a display device. More precisely, it is the number of pixels per unit length that can be placed horizontally and vertically. It is the number of pixels in horizontal direction × number of pixels in vertical direction.

Factors affecting the resolution:

i) **Spot profile:** The spot intensity has a gaussian distribution as depicted in figure. So, two adjacent spots on the display device appear distinct as long as their separation D_2 is greater than the diameter of the spot D_1 at which each spot has an intensity of about 60 percent of that at the center of the spot.

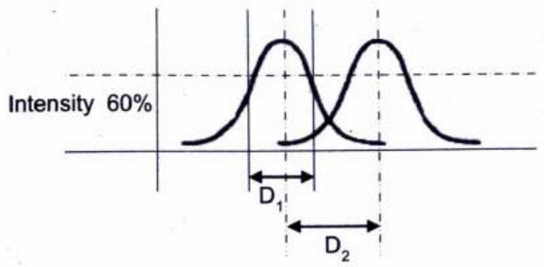


Figure 1.4: *Gaussian distribution of spot intensity*

ii) **Intensity:** As the intensity of electron beam increases, the spot size on the display tends to increase because of spreading of energy beyond the point of bombardment. This phenomenon is called blooming, and consequently, the resolution decreases.

12. Aspect ratio

It is defined as the ratio of vertical points to horizontal points necessary to produce equal length lines in both direction on the screen. Aspect ratio 3/4 means that a vertical line plotted with three points has the same length as a horizontal line plotted with four points. It is the ratio of image's height to its width.

600×800 pixels in display has the aspect ratio

$$\text{A.R.} = \frac{600}{800} = \frac{3}{4} = 3:4$$

13. Horizontal and vertical retrace

Horizontal retrace means at the end of each scan line, the returning of the electron beam to the left side of the screen to begin displaying the next scan line. The return to the left of the screen, after refresh each scan line is called the horizontal retrace of the electron beam. At the end of each frame ($\frac{1}{50}$ th of a second), the electron beam returns to the top left corner of the screen to begin the next frame. It is called vertical retrace.

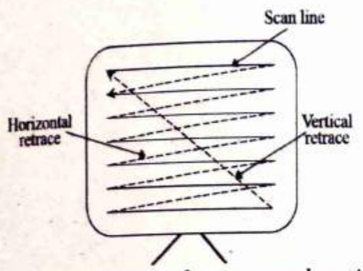


Figure 1.5: *Scan line, horizontal retrace, and vertical retrace*

14. Refresh buffer/ frame buffer/ bit map/ pix map:

In 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 the beam intensity turned on and off to create a pattern of illuminated spots. Picture definition is stored in a memory. The memory is 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 or pel (picture element). On black and white system with one bit per pixel, the frame buffer is commonly called a bit map. For system with multiple bits per pixel, the frame buffer is commonly called a pixmap.

Color depth	Number of color displayed	Byte of storage per pixel	Common name for color depth
4bit	16	0.5	Standard
8bit	256	1	256 color mode
16 bit	65,536	2	High color
24 bit	16,717,216	3	True color

1.8 Hardware Concepts

1. Tablet

Tablet, a digitizer, is a device used to scan an object. A tablet digitizes an object detecting the position of a movable stylus (a pencil shaped device) or a puck (a mouse like device with cross hairs for sighting positions) held in the user's hand. These discrete co-ordinate positions can be then joined with straight line segments to approximate the shape of original object. A tablet is a flat surface and its size varies from 6 by 6 inches up to 48 by 72 inches or more. The accuracy of the tablet usually varies from about 0.2 mm on desktop models to about 0.05 mm or less on larger models.

Types of tablet:

i. Electrical tablet
ii. Sonic (acoustic) tablet
iii. Resistive tablet

i) Electrical tablet

In electrical tablet, a rectangular grid of wires is embedded in the tablet surface. Electromagnetic pulses are generated along the wires and electric signal is induced in a wire coil in the stylus or puck. The strength of the signal induced by each pulse is used to determine the position of the stylus. A signal is sent to the computer when the tip of the stylus is pressed against the tablet or when any button on the puck is pressed. The information provided by the tablet repeats 30 to 60 times per second.

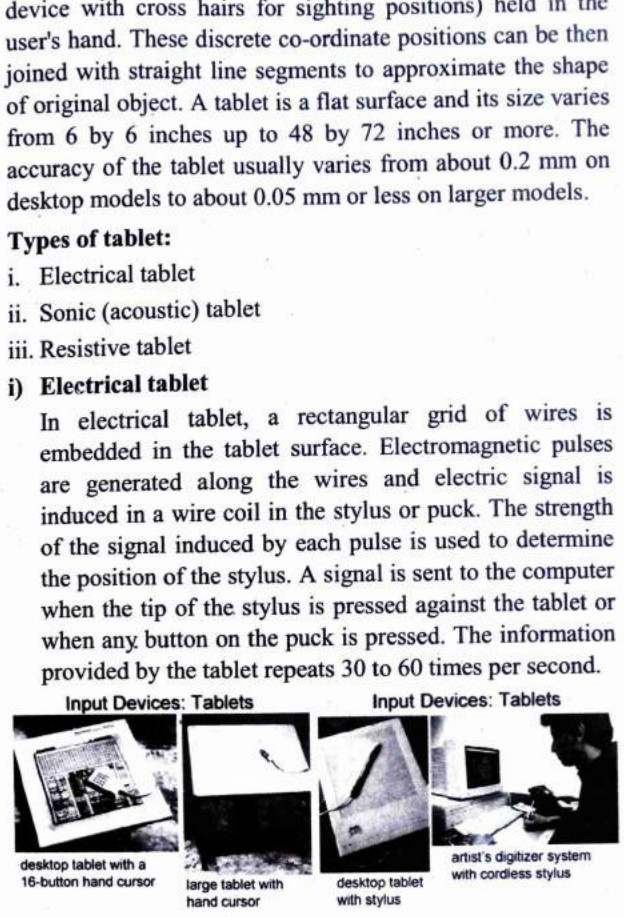

Input Devices: Tablets
Input Devices: Tablets
desktop tablet with a 16-button hand cursor
large tablet with hand cursor
desktop tablet with stylus
artist's digitizer system with cordless stylus

Figure 1.6: *Tablets*

ii) Sonic (acoustic) tablet

Sonic (acoustic) tablet uses sound waves to detect the stylus position. Microphone is used to detect the sound emitted by an electrical spark from a stylus tip. The position of the stylus or the co-ordinate values is calculated using the delay between when the spark occurs and when its sound arrives at each microphone. The main advantage of sonic tablet is that it doesn't require a dedicated working area as the microphones can be placed on any surface to form the tablet work area. This facilitates digitizing drawing on thick books.

iii) Resistive tablet

Resistive tablet is a piece of glass coated with a thin layer of conducting material. When a battery is powered, stylus is activated at certain position. The device emits high frequency radio signals which induces the radio signals on its conducting layer. The strength of the signal received at the edges of the tablet is used to calculate the position of the stylus. Several types of tablets are transparent, and thus can be backlit for digitizing x-ray films and photographic negatives. The resistive tablet can be used to digitize the objects on CRT because it can be curved to the shape of the CRT. The mechanism used in the electrical or sonic tablets can also be used to digitize the 3D objects.

2. Touch panels

The touch panel allows the user to select the screen positions directly with the touch of a finger to move the cursor around the screen or to interact with the icons. There are three types of touch panels:

i. Optical touch panel
ii. Electric touch panel
iii. Sonic (acoustic) touch panel

i. Optical touch panel

Optical touch panel employs a series of infrared light emitting diodes (LED) along one vertical edge and along one horizontal edge of the panel. The opposite vertical and horizontal edges contain light detectors which are used to record beams that are interrupted when the panel is touched. Touching the screen breaks one or two vertical and horizontal light beams. The interrupted beams identify the coordinate positions.

ii. Electrical touch panel

Electrical touch panel consists of slightly separated two transparent panels one coated with a thin layer of conducting material and the other with resistive material. When the outer plate is touched with a finger, it is forced into contact with the inner plates by creating the voltage drop across the resistive plate which is then used to calculate the co-ordinate of the touched position.

iii. Sonic (acoustic) touch panel

In sonic (acoustic) touch panel, high frequency sound waves traveling alternately horizontally and vertically are generated at the edge of the panel (glass plate).Touching the screen causes part of each wave to be reflected back to its source. The screen position at the point of contact is then calculated using the time interval between the transmission of each wave and its reflection to the emitter.

3. Light Pen

It is a pencil shaped device used to select the co-ordinates of a screen point by detecting the light coming from the points on the CRT screen. In raster display 'Y' is set at Y_{max} and 'X' changes from 0 to X_{max} in the first scan line. For the second line, 'Y' decreases by one and 'X' again changes from 0 to X_{max} and so on. When activated light pen sees a burst of light at certain position as the electron beam hits the phosphor coating at that position, it generates an electric pulse. This is used to save the video controller's 'X' and 'Y' registers and interrupt the computer. By reading the saved value, the graphics package can determine the co-ordinates of the position seen by the light pen.

4. Keyboard

Keyboard is used for entering text. It consists of alphanumeric key, function keys, cursor-control keys, and separate numeric pad. It is used to move the cursor, to select the menu, item, predefined functions. In computer graphics, keyboard is mainly used for entering screen co-ordinate and text to invoke certain functions. Nowadays, ergonomically designed keyboard (ergonomic keyboard) with removable palm rests is available. The slope of each half of the keyboard can be adjusted separately.

5. **Mouse**

Mouse is a small handheld device used to position the cursor on the screen. It can be picked up, moved in space, and then put down again without any change in the reported position. For this, the computer maintains the current mouse position, which is incremented or decremented by the mouse movements. Following are the mice, which are mostly used in computer graphics:

i. **Mechanical mouse**

When roller in the box of this mechanical mouse is moved, a pair of orthogonally arranged toothed wheels, each place in between LED and a photo detector, interrupts the light path. The numbers it interrupts, so generated, are used to report the mouse movements to the computer.

ii. **Optical mouse**

The optical mouse is used on a special pad having grid of an alternating light and dark lines. A LED in the bottom of the mouse directs a beam of light down onto the pad from which it is reflected and sensed by the detectors on the bottom of the mouse. As the mouse is moved the reflected light beam is broken each time a dark line crossed. The number of pulses so generated, which is equal to the number of lines crossed are used to report mouse movements to the computer.

6. **Barcode reader**

A barcode reader (or barcode scanner) is an electronic device for reading printed barcodes. Like a flatbed scanner, it consists of a light source, a lens, and a light sensor. Barcode reader translates optical impulses into electrical ones. Additionally, nearly all barcode readers contain decoder circuitry analyzing the barcode's image data provided by the sensor and sending the barcode's content to the scanner's output port.

7. **Data glove**

A data glove is an interactive device resembling a glove worn on the hand, which facilitates tactile sensing and fine-motion control in robotics and virtual reality. Data gloves are one of several types of electromechanical devices used in haptics applications. Tactile sensing involves simulation of the sense of human touch and includes the ability to perceive pressure, linear force, temperature, and surface texture. Fine-motion control involves the use of sensors to detect the movements of the user's hand and fingers and the translation of these motions into signals that can be used by a virtual hand (for example, in gaming) or a robotic hand (for example, in remote-control surgery).

1.9 Refresh Cathode Ray Tube

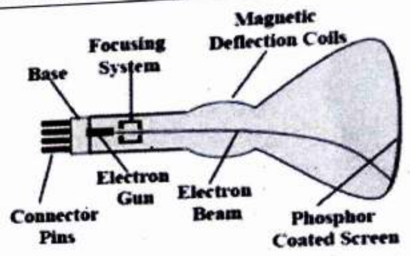


Figure 1.7: *Basic design of a magnetic-deflection CRT*

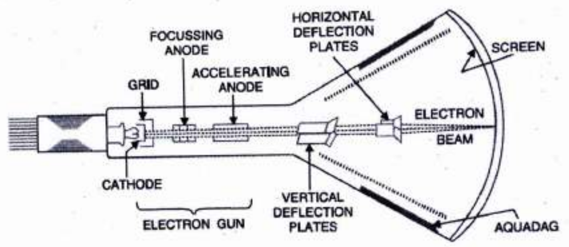


Figure 1.8: *Operation of an electron gun with an accelerating anode.*

It consists of a CRT along with control circuits. CRT is a vacuum glass tube with the display screen at one end and connectors to the control circuits at the other. Inside of display screen is a special material called phosphor which emits light for a period of time when hit by a beam of electrons. The color of light and the time period vary from one type of phosphor to another.

The major parts of refresh CRT are:

1. **Electron gun**

   Electron gun is made up of heated metal cathode and a control grid.

   i. **Heated metal cathode**

   Heat is supplied to the cathode by directing a current through a coil by wire, called heating filament, inside the cylindrical cathode structure. This causes electron to be "boiled off" the cathode surface.

   ii. **Control grid**

   Control grid is responsible for controlling the brightness of a display. By setting voltage level on control grid, the brightness emitted by phosphor coating depends on the number of electrons that strike the phosphor coat.

2. **Accelerating anode**

   In the vacuum inside the CRT envelope, the freely negatively charged electrons are accelerated towards the phosphor coating by a high positive voltage (15000-20000).This high positive voltage can be generated by using accelerating anode.

3. **Focusing system/focusing anode**

   It is required to force the electron beam to converge into small spot when it strikes the phosphor coat. Otherwise, the electrons would repel each other and the beam would spread out as it approaches the screen. There are two types of focusing system: electrostatic focusing and magnetic field focusing. Additional focusing hardware is used in high precision systems to keep the beam in focus at all screen positions. Because of radius of curvature for CRT monitor as beam moves to the outer edges of the screen displayed image may be blurred. To adjust this, the focusing is necessary.

4. **Deflection system**

   Deflection system is needed to direct the electron beam towards a particular point on the screen. It is done in two ways: electrostatic deflection system and magnetic deflection system. When electron beam passes through the horizontal and vertical deflection plates, it is bent or deflected by the electric fields between the plates. The horizontal plates control the beam to scan from left to right and retrace from right to left (horizontal retrace).The vertical plates control the beam to go from the first scan line at the top to the last scan line at the bottom and retrace from the bottom back to the top (known as vertical retrace).

1.10 Raster and Random (Vector) Scan Display

1.10.1 Raster Scan Display

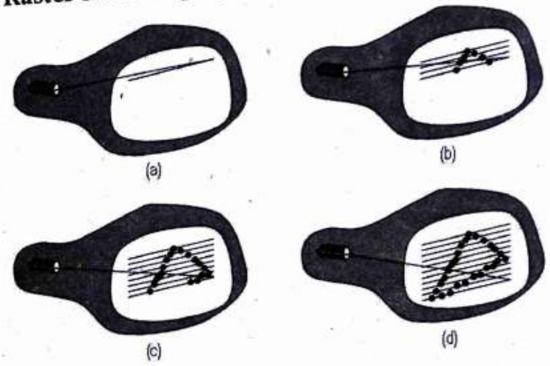


Figure 1.9: Raster scan display

Most common type of graphics monitor are employing a CRT (based on television technology). The electron beam is swept across the screen one row at a time from top to bottom. Picture definition is stored in memory area called "refresh buffer" or "frame buffer". The set of intensity values is retrieved from refresh buffer and "painted" on the screen one row at a time. Each screen point is referred as a "pixel" or "pel". For black and white system, each screen point (pixel) is represented by one bit either "on" or "off". For color system, additional bit is needed to represent single pixel (24 bit per pixel in high quality systems).For black and white system with one bit per pixel, the frame buffer is called "bitmap" whereas for system with multiple bits per pixel, the frame buffer is called "pixmap". Normal refresh rate is 50-60 frames per second.

Interlacing

Some monitors use a technique called "interlacing" to double the refresh rate. In this case, only half of the scan lines in a frame are refreshed at a time, first the odd numbered lines, and then the even numbered lines. Thus, the screen is refreshed from top to bottom in half time it would have taken to sweep across all the scan line. This effect is quite effective in reducing flicker.

1.10.2 Random (Vector) Scan Display

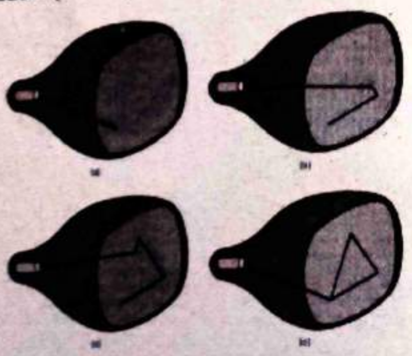


***Figure 1.10:** Vector scan display*

In random or vector scan display, electron beam is directed only to the points of the screen where a picture is to be drawn e.g., pen plotter (hard copy device). In this system, refresh rate depends on the number of lines to be displayed. Picture definition is stored as a set of line drawing commands in system memory called display file (or display program or display list).To display a specified picture, the system cycles through the set of commands in 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 components of a picture 30 to 60 times each second.

1.10.3 Difference between Raster Scan Display and Random (Vector) Scan Display

Base of difference	Raster scan display	Random (vector) scan display
1. Electron beam	The electron beam is swept across the screen one row at a time from top to bottom.	The electron beam is swept to the parts of the screen where a picture is to be drawn.
2. Resolution	It has lower or poor resolution because picture definition is stored as an intensity value.	It has high resolution because it stores picture definition as a set of line commands.
3. Picture definition	Picture definition is stored as a set of intensity values for all screen points (pixels) in a refresh buffer.	Picture definition is stored as a set of line in a display list or file.
4. Realistic display	The capacity of the system to store intensity values for pixels make it well suited for realistic display with shadow and color pattern.	These systems are designed for line-drawing and can't display realistic shaded scenes.
5. Image drawing	Screen points or pixels are used to draw an image.	Mathematical functions are used to draw an image.
6. Cost	They are cheaper than random display.	It is more expensive than raster-scan display.

Base of difference	Raster scan display	Random (vector) scan display
7. Refresh rate	Refresh rate is 60-80 fps.	All components are drawn 30 to 60 times per second.
8. Interlacing	It uses interlacing.	It doesn't use interlacing.
9. Editing	Editing is difficult.	Editing is easy.
10. Refresh area	Refresh area is independent of picture complexity.	Refresh area depends on complexity of picture.
11. Smoothness	Produce jagged line.	Produce smooth line.
Example:	CRT, TV, Printer	Pen Plotter

1.11 Color CRT Monitors

Color CRT monitor displays color pictures using a combination of phosphors that emits different colored light.

Two basic techniques are available:

i) Beam penetration method

ii) Shadow mask method

i) Beam penetration method

In this method, two different layers of phosphor coating used red (outer) and green (inner).It displays color depending on the depth of penetration of electron beam into the phosphor layer.

i. A beam of slow electron excites only the outer red layer.

ii. A beam of very fast electrons penetrates through the red phosphor and excites the inner green layer.

iii. When quantity of red is more than green, then color appears as orange.

iv. When quantity of green is more than red, then color appears as yellow.

Screen colors as controlled by the beam acceleration voltage. Only four colors are possible and picture quality is poor.

ii) Shadow mask method

The inner side of the viewing surface of a color CRT consists of closely spaced groups of red, green, and blue phosphor dots. Each group is called a triad. A thin metal plate perforated with many small holes is mounted close to the inner side of the viewing surface. This plate is called shadow mask. The shadow mask is mounted in such a way that each hole is correctly aligned with a triad in color CRT. There are three electron guns one for each dot in triad. The electron beam from each gun therefore hits only the corresponding dot of a triad as the three electron beams deflect. A triad is so small that light emanatory from the individual dots is perceived by the viewer as a mixture of the three colors.

Two types of shadow mask method:

a) Delta-delta CRT

b) Precision inline CRT

a) Delta-delta CRT

A triad has a triangular or delta pattern as are three electron guns. Main drawback of this method is that a high precision display is very difficult to achieve because of technical difficulties involves in the alignment to shadow mask holes and the triad on one to one basis.

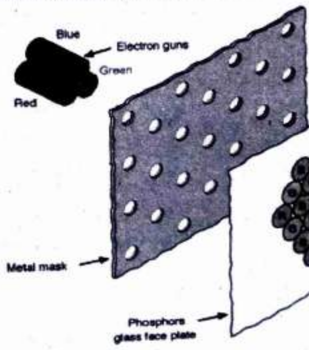


Figure 1.11: Delta-delta CRT

b) Precision inline CRT

A triad has an *in-line pattern* as are the three electron guns. The introduction of this type of CRT has eliminated the main drawback of a delta-delta CRT. Normally, 1000 scan lines can be achieved by this method.

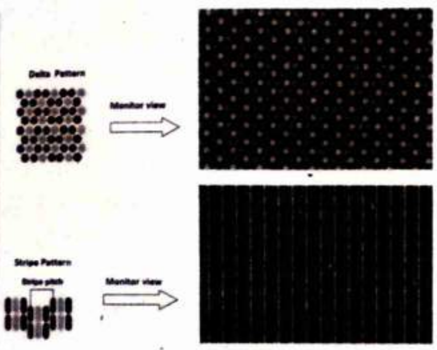


Figure 1.12: *Delta-delta and precision inline CRT*

But in this method, a slight reduction of image sharpness at the edges of the tube has been noticed. Normally, 1000 scan lines can be achieved. The necessity of triad has reduced the resolution of a color CRT. The distance between the centers of adjacent triads is called a *pitch.* In very high resolution tubes, pitch measures 0.21 mm (0.61 mm for home TV tubes). The diameter of each electron beam is set at 1.75 times the pitch. For example, if a color CRT is 15.5 inches wide and 11.6 inches high and has a pitch of 0.01 inches. The beam diameter is therefore 0.01 ×1.75 = 0.018 inches. Thus, the resolution per inch is about $\frac{1}{0.018} = 55$ lines. Hence, the resolution achievable for the given CRT is 850 (=15.5×55) by 638 (=11.6×55). The resolution of a CRT can therefore be increased by decreasing the pitch. But small pitch CRT is difficult to manufacture because it is difficult to set small triads and the shadow mask is more fragile owing to too many holes on it. Beside the shadow is more likely to warp from heating by the electrons.

1.12 Raster Scan Display System/ Architecture/ Technology

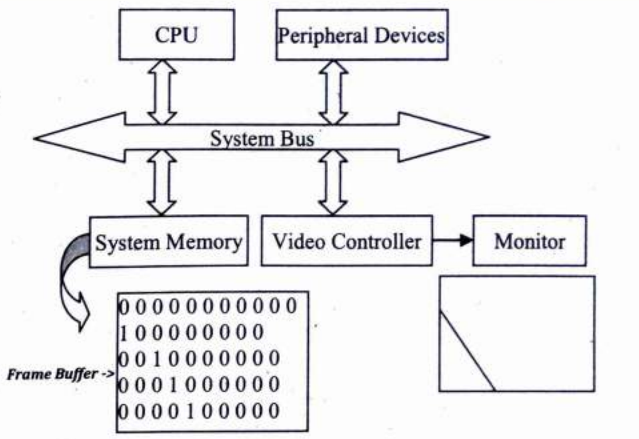


Figure 1.13: *Raster Scan System*

Raster scan system consists of CPU, a video controller (special-purpose processor), a monitor, system memory, and peripheral devices.

Advantages:

It has an ability to fill the areas with solid colors or patterns. The time required for refreshing is independent of the complexity of the image. It has low cost.

Disadvantages:

For real-time dynamics, not only the end points are required to move but all the pixels in between the moved end points have to be scan converted with appropriate algorithms which might slow down the dynamic process. Due to scan conversion, "*jaggies*" or "*stair-casing*" are unavoidable.

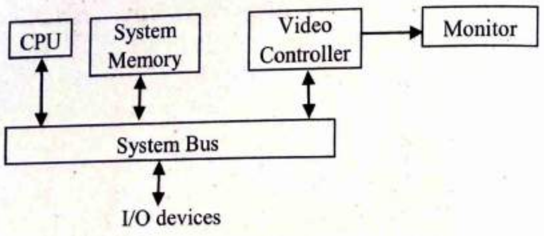


***Figure 1.15:** Architecture of a simple raster graphics system.*

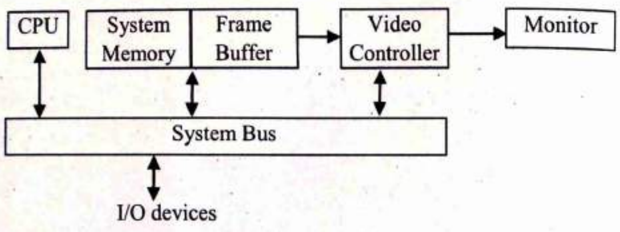


***Figure 1.14:** Architecture of RS with a fixed portion of the system memory reserved for the FB.*

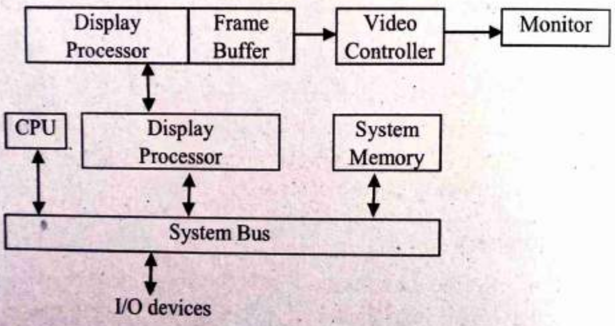


***Figure1.15:** Architecture of a raster-graphics system with a display processor.*

Application program and graphics subroutine package both reside in system memory and execute in CPU. When particular command (e.g., line (x_1, y_1, x_2, y_2)) is called by application program, the graphics subroutine package sets the appropriate pixels in the frame buffer. The video controller then cycles through the frame buffer, one scan line at the time (50 fps).It brings a value of each pixel contained in the buffer.

Video controller

A fixed area of the system memory is reserved for frame buffer and video controller is given direct access to the frame-buffer memory to refresh the screen.

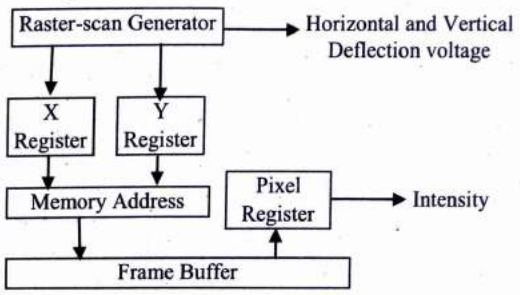


***Figure 1.16:** Basic video-controller refresh operation.*

The screen positions are referenced in Cartesian co-ordinate. Origin is defined as the lower left screen corner. The screen surface is then represented as the first quadrant of 2D system. Two registers are used to store co-ordinates of screen pixel.

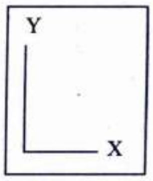


***Fig. 1.17:** Screen position*

Initially, Y-register is set to Y_{max} and X to 0.

The value stored in frame buffer for this pixel is retrieved and used to set intensity of CRT beam. Then X-register is incremented by 1 and same process is repeated for each pixel along scan line as before. After cycling through all pixels along bottom scan-line ($y = 0$), video controller resets register to first position on top scan line and refresh process starts again. Since the refreshing per pixel is slow process, the video controller retrieves the intensity values for a group of adjacent pixels from the frame buffer. This block of pixel intensity is stored in separate registers and used to control the CRT beam intensity for a group of adjacent pixels on the screen. Video controller can retrieve pixel intensities from different memory areas on different refresh cycles.

Display processing unit (DPU)

Display processing unit (DPU) is also called graphics controller or display co-processor. The purpose of display processor is to free the CPU from graphic chores (manipulation).It has its own memory. The major task is digitizing a picture definition given in an application program into a set of pixel intensity values for storage in frame buffer. The digitization process is called scan conversion.

DPU also performs generating various line style (dashed, dotted, solid lines), displaying color areas, and performing various transformation.

1.13 Random Scan System/Architecture/Technology

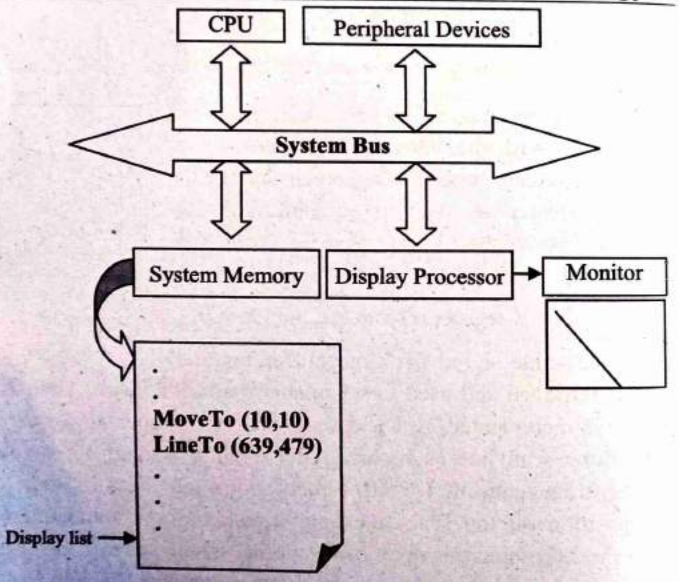


***Figure 1.18:** Architecture of a simple random scan system*

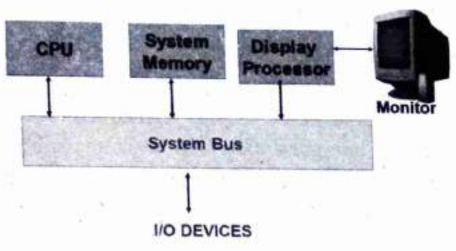


***Figure 1.19:** Architecture of a simple random scan system.*

Random scan system was developed in 60's and used as common display device until 80's. The system consists of CPU, a display processor (DPU or graphics controller), a CRT monitor, system memory, peripheral devices. An application program is input and stored in the system memory along with a graphics package. Graphics commands in the application program are translated by graphics package into a display file stored in the system memory. The display list or file is then accessed by the display processor to refresh the screen. The display processor cycle through each command in the display file program once during every refresh cycle.

1.14 Flat Panel Displays

Flat panel displays have reduced volume, weight and power requirements compared to CRT. Current uses of flat panel displays are like, in TV monitor, calculator, pocket video games, laptop, and armrest viewing of movies on airlines.

Two types:

i. Emissive displays

ii. Non-emissive displays

1.14.1 Emissive Displays (Emitter)

Emissive displays are devices that convert electrical energy into light energy.

E.g., plasma panel, LED, thin-film electroluminescent displays.

Plasma Panel

Plasma panel is also known as gas-discharge display. Region between two glasses plates is filled with mixture of gases usually neon. A series of vertical conducting ribbons is placed on one glass plate and horizontal ribbons in another. Firing voltage is applied the pair of conductor to break down into glowing plasma of electrons and ions. Picture definition is stored to refresh buffer and firing voltage is applied to refresh the pixel position 60 times per second. Separation between pixels is provided by electric field of conductors.

Thin-Film Electroluminescent Displays

Thin film electroluminescent displays are similar in construction to a plasma panel filled with phosphor such as zinc sulfide doped with manganese instead of gas. When 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 manganese atoms which release the energy as spot of light to glowing plasma affect in plasma panel.

LED

In LED, a matrix of diodes is arranged to form the pixel positions in the display and picture definition is stored in a refresh buffer. Information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.

1.14.2 Non-Emissive Displays (Non-Emitter)

It uses optical effects to convert sun light or light from some other source into graphic pattern.

E.g., LCD (Liquid crystal device)

LCD (Liquid Crystal Display)

LCD is 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 source through a liquid crystal material that can be aligned to either block or transmit the light.

SOLVED NUMERICAL

1. ***Calculate the frame buffer size (in KB) for a raster system recording a video for 1 min with resolution of 1280 × 1024, and storing 24 bits per pixel with a refresh rate of 25 fps.***
[2076 Ashwin Back]

Solution:

Screen resolution = 1280 × 1024

Refresh rate = 25 fps

Bit required to represent a pixel = 24 bits

Memory required just for a frame = 1280 × 1024 × 24 bits

Memory required for 1 second = 1280 × 1024 × 24 × 25 bits

Memory required for recording a video for 1 min is

= 1280 × 1024 × 25 × 24 × 60 bits = 5760,0000 KB

2. ***If pixels are accessed from the frame buffer with an average access the 300ns. Then will this rate produce the flickering effects? (screen resolution = 640×480)***

Solution:

Access time for 1 pixel = 300ns

Access time for 640×480 pixels = 640 × 480 × 300ns

$$\text{Frequency} = \frac{1}{t} = \frac{1}{640 \times 480 \times 300 \times 10^{-9}}$$

= 10.85 frame per second (fps)

This value is lesser than 50fps, so flicker occurs.

2. ***If the total intensity available for a pixel is 256 and the screen resolution is 640×480. What will be the size of the frame buffer?***

Solution:

Size in frame buffer for 1 pixel = 8 bit

For 640×480 pixels, size in frame buffer = 640 × 480 ×8 bits

= 300 KBytes

4. ***Consider 256 pixel × 256 scan lines image with 24-bit true color. If 10 minutes video is required to capture, calculate the total memory required?***

Solution:

Memory required for 1 sec = 256×256×3×50 Bytes

For 10 minutes, total memory required

$$= \frac{(256\times256\times3\times50\times10\times60)}{(1024\times1024\times1024)} = 5.49 \text{ GB}$$

5. ***If we want to resize at 1024×768 image to one that is 640 pixels wide with the same aspect ratio, what would be the height of the resized image?*** *[2070 Ashadh]*

Solution:

$$\text{Aspect ratio} = \frac{H}{W} = \frac{768}{1024}$$

Even after the image is resized, the aspect ratio remains same. So,

$$\frac{H}{640} = \frac{768}{1024}$$

$$\therefore H = 480$$

6. ***How much time is spent scanning across each row of pixels during screen refresh on a raster system with resolution 1024×768 and refresh rate 60 frames per second?*** *[2070 Chaitra]*

Solution:

Resolution = 1024×768

Refresh rate = 60fps

For 60 frames, it takes 1 second to scan

For 1 frame, it takes $\frac{1}{60}$ second

1 frame means total 1024×768 pixels

For 1024×768 pixels, it takes $\frac{1}{60} = 0.016667$ second

For scanning 1 pixel, it takes $\frac{0.016667}{1024\times768}$ second

For scanning 1 row of pixels i.e., 1024 pixels, it takes $\frac{0.016667}{1024\times768}\times1024 = 0.0000217013$ second.

7. ***Consider a raster scan system having 12 inch by 10 inch screen with a resolution of 100 pixels per inch in each direction. If the display controller of this system refreshes the screen at the rate of 50 frames per second, how many pixels could be accessed per second and what is the access time per pixel of the system?*** *[2071 Shrawan]*

Solution:

Total pixels = 12×100×10×100

Refresh rate= 50 frames per second.

Pixels accessed per second(f) = 12×100×10×100×50

= 60000000

Access time per pixel $= \frac{1}{f} = 1.667\times10^{-8}$ second

Chapter – 2

Scan Conversion

2.1 Output Primitives

The basic geometric structures such as points, straight line segments, circles, conical sections, quadric surfaces, spline curves and surfaces, polygon color areas, which are used to describe a scène, are called output primitives.

The graphic programming packages describe the scene in terms of output primitives and to group sets of output primitives into more complex structures.

2.2 Line-Drawing Algorithm

The slope-intercept equation of a straight line is:

$y = mx + b$.............(1)

where

m = slope of line

b = y-intercept

For any two given points (x_1, y_1) and (x_2, y_2),

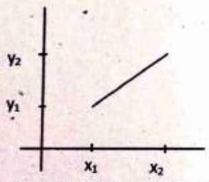


Fig. 2.1: Line path between two endpoint positions

$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{dy}{dx}$ ....(2)

$b = y_1 - m x_1$..................(3)

At any point (x_k, y_k)

$y_k = mx_k + b$..................(4)

At point (x_{k+1}, y_{k+1})

$y_{k+1} = mx_{k+1} + b$..............(5)

Subtracting equation (4) from (5)

$y_{k+1} - y_k = m(x_{k+1} - x_k)$

$(y_{k+1} - y_k)$ is increment in y as correspondingly increment in x.

For given x interval Δx along a line, we can compute the corresponding y interval Δy from the following equation.

$\Delta y = m\,\Delta x$............(6)

Similarly, we can obtain the x interval Δx corresponding to specified Δy as

$\Delta x = \frac{\Delta y}{m}$ ....(7)

Case I: For lines with slope magnitudes $|m|<1$, Δx can be set proportional to a unit horizontal deflection voltage and the corresponding vertical deflection is then set proportional to Δy as calculated from equation 6.

Case II: For lines whose slopes have magnitude $|m|>1$, Δy can be set proportional to a unit vertical deflection voltage and the corresponding horizontal deflection voltage set proportional to Δx, calculated from equation 7.

Case III: For lines with $m=1$, $\Delta x = \Delta y$ and the horizontal and vertical deflection voltages are equal.

On raster systems, we must sample a line at discrete positions and determine the nearest pixel to the line at each sampled position.

2.2.1 Digital Differential Analyzer Algorithm

Digital differential analyzer (DDA) algorithm is scan conversion line algorithm based on calculating either Δx or Δy using the relation.

$\Delta y = m\Delta x$......... (i)

$\Delta x = \frac{\Delta y}{m}$.......... (ii)

We sample the line at unit interval in one co-ordinate and determine corresponding integer values nearest the path for the other co-ordinate.

Different cases:

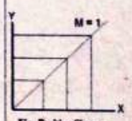
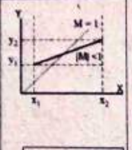
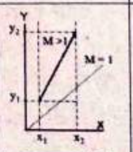
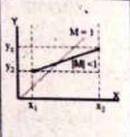
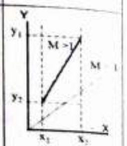
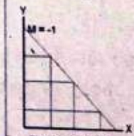
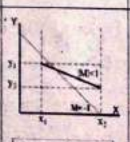
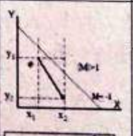
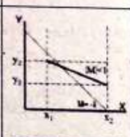
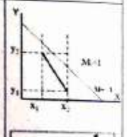
Case 1: Line with positive slope and magnitude less than or equal to 1 ($|m| \leq 1$)

Case 2: Line with positive slope and magnitude more than 1 (|m|>1)

Case 3: Line with negative slope and magnitude less than or equal to 1 ($|m| \le 1$)

Case 4: Line with negative slope and magnitude more than 1 (|m|>1)

The possible combinations can be shown as follows:

	Moving Left to Right		Moving Right to Left	
	Slope (m) < 1	Slope (m) > 1	Slope (m) < 1	Slope (m) > 1
Positive Slope  Fig. Positive Slope	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k + 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k - 1$
Negative Slope  Fig. Negative Slope	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k - 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k + 1$

Line with a positive slope

Case I: If slope (m) ≤ 1, then sample at unit x intervals ($\Delta x = 1$) and compute each successive y value because the increment in x is more than increment in y.

So, set $\Delta x = 1$.

Now, from equation (1), we get

$\Delta y = m$

i.e., $x_{k+1} = x_k + \Delta x = x_k + 1$...............(3)

$y_{k+1} = y_k + \Delta y = y_k + m$...............(4)

where subscript k takes integer values starting from 1, for the first point, and increases by 1 unit until the final end point is reached. Since m can be any real number between 0 to 1, the calculated y values must be rounded to the nearest integer.

Case II: If m > 1, then the increment in x (Δx) is smaller than increment in y (Δy),

So, set $\Delta y = 1$.

Then,

$$\Delta x = \frac{1}{m}$$

That is,

$$x_{k+1} = x_k + \frac{1}{m} \text{..........(5)}$$

$$y_{k+1} = y_k + 1 \text{..........(6)}$$

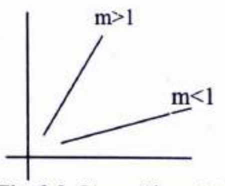


Fig. 2.2: Line with positive slope |m|<1 and |m|>1

Here, in the both case m≤1 and m>1, consider the algorithm (i.e., equations 3, 4, 5, 6) based on the assumption that lines are to be processed from the left end point to the right end point.

If this process is reversed (that is, lines are to be processed from **right to left**), relation required to change in both case.

Case I: If m≤1, $\Delta x = -1$, so, $\Delta y = -m$

i.e., $x_{k+1} = x_k - 1$

$y_{k+1} = y_k - m$

Case II: If m > 1, $\Delta y = -1$ and $\Delta x = -\frac{1}{m}$

i.e., $x_{k+1} = x_k - \frac{1}{m}$ ....(7)

$y_{k+1} = y_k - 1$ ....(8)

Line with negative slope

Case I: If |m|≤1, then assume start end point is the left.

$\Delta x = 1$ and $\Delta y = m$ (m is negative)

That is,

$x_{k+1} = x_k + 1$.............(9)

$y_{k+1} = y_k + m$.............(10)

If algorithm is required to proceed right to left, then set on

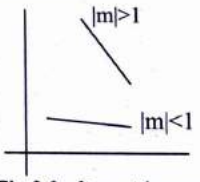


Fig 2.3: line with negative slope |m|<1 and |m|>1

$\Delta x = -1$ and $\Delta y = -m$

i.e., $x_{k+1} = x_k - 1$..............(11)

$y_{k+1} = y_k - m$.............(12)

Case II: If $|m| > 1$, then assume start end is at left and set $\Delta y = -1$

and $\Delta x = -\frac{1}{m}$

i.e., $x_{k+1} = x_k - \frac{1}{m}$..............(13)

$y_{k+1} = y_k - 1$..................(14)

If the algorithm is required to proceed **right to left**, then set

$\Delta y = 1$ and $\Delta x = \frac{1}{m}$.

i.e., $x_{k+1} = x_k + \frac{1}{m}$ ....(15)

$y_{k+1} = y_k + 1$..................(16)

DDA Algorithm

```
Step 1:  Start Algorithm.
Step 2:  Declare $x_1$, $y_1$, $x_2$, $y_2$, dx, step as integer variable and x, y,
         $x_{inc}$, $y_{inc}$ as floating point.
Step 3:  Enter value of $x_1$, $y_1$, $x_2$, $y_2$.
Step 4:  Calculate dx = $x_2 - x_1$.
Step 5:  Calculate dy = $y_2 - y_1$.
Step 6:  If absolute (dx) > absolute (dy)
         Then step = absolute (dx)
         Else step = absolute (dy)
Step 7:  $x_{inc} = \frac{dx}{step}$
         $y_{inc} = \frac{dy}{step}$
         assign x = $x_1$
         assign y = $y_1$
Step 8:  Set pixel (x, y)
Step 9:  $x = x + x_{inc}$
         $y = y + y_{inc}$
         Set pixels (Round (x), Round (y))
Step 10: Repeat step 9 until x = $x_2$
Step 11: End Algorithm
```

Advantages:

- It is faster method than direct line drawing equation y = mx+c for calculating pixel position as it eliminates multiplication.
- It avoids multiplication operation. It is simple to understand, and it doesn't require special knowledge to implement.

Disadvantages:

- The floating point addition is still needed in determining each successive point which is time consuming. The value of slope 'm' is usually stored in floating point number. So, there could be round off error.
- The line will move away from the true line path, especially when it is long due to successive round of error.
- Accumulation of round off error in successive additions of floating point increment.

Code in C to draw a line using DDA Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
        int gd=DETECT,gm;
        int x1,y1,x2,y2,stepsize,dx,dy,i;
        float x,y, xinc,yinc;
        initgraph(&gd,&gm,"c:\\tc\\bgi");
        printf("Digital Differntial Line Drawing Algorithm\n");
```

```
    printf("put the values of x1 and y1\n");
    scanf("%d %d",&x1,&y1);
    printf("put the values of x2 and y2\n");
    scanf("%d %d",&x2,&y2);
    dx = x2-x1;
    dy = y2-y1;
    x = x1;
    y = y1;
    if (abs(dy)>abs(dx))
    {
    stepsize=abs(dy);
    }
    else
    {
    stepsize=abs(dx);
    }
    xinc=dx/(float)stepsize;
    yinc=dy/(float)stepsize;
    putpixel(x,y,RED);
    for(i=0;i<stepsize;i++)
    {
     x=x+xinc;
     y=y+yinc;
     putpixel((int)(x+0.5),(int)(y+0.5),RED);
     }
    getch();
    closegraph();
}
```

Code in C to draw a checker box using DDA Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
```

```
#include<dos.h>
#include<math.h>
void dda (int, int, int, int);
void main()
{
      int gd = DETECT, gm;
      initgraph (&gd, &gm, "c:\\tc\\bgi");
      dda (100,100,200,100);
      dda (200,100,200,200);
      dda (200,200,100,200);
      dda (100,200,100,100);
      dda (100,100,200,200);
      dda (100,200,200,100);
      dda (100,125,200,125);
      dda (100,150,200,150);
      dda (100,175,200,175);
      dda (175,100,175,200);
      dda (175,100,150,200);
      dda (125,100,125,200);
      dda (100,150,150,100);
      dda (150,100,200,150);
      dda (200,150,150,200);
      dda (150,200,100,150);
       getch();
       closegraph();
       }
   void dda (int x1, int y1, int x2, int y2)
   {
     int i, stepsize, dx, dy;
     float x, y, xinc, yinc;
     dx = x2-x1;
```

```
    dy = y2-y1;
    x = x1;
    y = y1;
    if (abs(dy) > abs(dx))
    {
    stepsize = abs(dy);
    }
    else
    {
    stepsize = abs(dx);
    }
    xinc =  dx / (float) stepsize;
    yinc=dy/ (float) stepsize;
    putpixel (x, y, RED);
    for (i=0; i < stepsize; i++)
    {
      x = x + xinc;
      y = y + yinc;
      putpixel ((int) (x+0.5), (int) (y+0.5), RED);
      delay (10);
    }
    }
```

2.2.2 Bresenham's Line Algorithm

Bresenham's line algorithm is an accurate and efficient line drawing algorithm. It uses only integer arithmetic to find the next position to be plotted. It avoids incremental error. The major concept of Bresenham's algorithm is to determine the nearest pixel position. Great advantage of this algorithm is that it can be used to display circles and other curves.

In Bresenham's algorithm, we calculate the decision parameter which decides which pixel to select and which function is used for next decision parameter.

For positive slope and slope |m| < 1

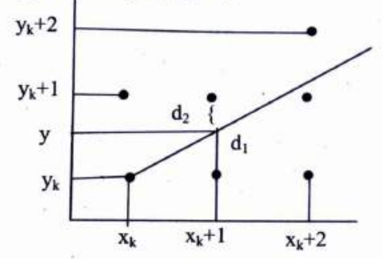


Figure 2.4: *Line with m<1*

- Pixel positions are determined by sampling at unit x intervals.
- Starting from left end position (x_0, y_0) of a given line, we step to each successive column (x-position) and plot the pixel whose scan line y value is closed to the line path.

Assuming the pixel at (x_k, y_k) to be displayed is determined, we next to decide which pixel to plot in column $x_{k+1,}$ our choices are the pixels at positions.

$(x_k + 1, y_k)$ and $(x_k + 1, y_k + 1)$

At sampling position $x_k + 1$, we label vertical pixel separations from the mathematical line path d_1 and d_2.

The y-co-ordinate on the mathematical at pixel column position $x_k + 1$ is calculation.

As,

$y = m (x_k + 1) + b$........(1)

Then,

$d_1 = y - y_k$

$d_1 = m (x_k + 1) + b - y_k$ .... (2)

And,

$d_2 = (y_k + 1) - y$

$d_2 = y_k + 1 - m (x_k+1) - b$ .... (3)

Now,

$d_1 - d_2 = m(x_k + 1) + b - y_k - y_k - 1 + m(x_k + 1) + b$

$= 2m(x_k + 1) - 2y_k + 2b - 1$

$= 2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1$

Defining decision parameter $p_k = \Delta x(d_1 - d_2)$

$p_k = \Delta x(d_1 - d_2) = 2\Delta y(x_k + 1) - 2\Delta x y_k + 2\Delta x b - \Delta x$

$= 2\Delta y x_k + 2\Delta y - 2\Delta x y_k + \Delta x(2b - 1)$

$= 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + \Delta x(2b - 1)$

$= 2\Delta y x_k - 2\Delta x y_k + c$............(4)

Where, $c = 2\Delta y + \Delta x(2b - 1)$

Here, sign of p_k is same as the sign of $d_1 - d_2$ since $\Delta x > 0$.

Case I:

If $p_k \geq 0$, then $d_2 < d_1$, which implies that $y_k + 1$ is nearer than y_k. So, pixel at $(y_k + 1)$ is better to choose which reduce error than pixel at y_k. This determines next pixel co-ordinate to plot is $(x_k + 1, y_k + 1)$.

Case II:

If $p_k < 0$ then $d_1 < d_2$ which implies pixel at y_k is nearer than pixel at $(y_k + 1)$.So, pixel at y_k is better to choose which reduce error than pixel at $(y_k + 1)$.This determines next pixel co-ordinate to plot is $(x_k + 1, y_k)$

Now, similarly, pixel as $(x_k + 2)$ can be determined whether it is $(x_k + 2, y_k + 1)$ or $(x_k + 2, y_k + 2)$ by looking the sign of deciding parameter p_k+1 assuming pixel as (x_k+1) is known.

$p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c$ where c is same as in p_k

Now,

$p_{k+1} - p_k = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c - (2\Delta y x_k - 2\Delta x y_k + c)$

$= 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$ where $x_{k+1} = x_k + 1$

$= 2\Delta y(x_k + 1 - x_k) - 2\Delta x(y_{k+1} - y_k)$

$= 2\Delta y - 2\Delta x(y_{k+1} - y_k)$

This implies that decision parameter for the current column can be determined if the decision parameter of the last column is known.

Here, $(y_{k+1} - y_k)$ could either 0 or 1 which depends on sign of p_k.

If $p_k \geq 0$ (i.e.,$d_2 < d_1$), $y_{k+1} = y_k + 1$ which implies $(y_k + 1 - y_k) = 1$

That is, at $p_k \geq 0$, the pixel to plot is $(x_k + 1, y_k + 1)$ and

$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

If $p_k < 0$ (i.e.,$d_1 < d_2$), $y_{k+1} = y_k$ which implies $(y_{k+1} - y_k = 0)$

i.e., at $p_k < 0$, then pixel to be plotted is (x_k+1, y_k)

and $p_{k+1} = p_k + 2\Delta y$

Initial decision parameter (p_0)

$p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + \Delta x(2b - 1)$

$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2b - 1)$

But $b = y_0 - m x_0 = y_0 - \frac{\Delta y}{\Delta x} x_0$

$= 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$

$= 2\Delta y - \Delta x$

BLA Algorithm

```
Step 1. Start
Step 2. Declare variables $x_1, y_1, x_2, y_2, l_x, l_y, \Delta x, \Delta y, p_0, p_k, p_{k+1}$
Step 3. Read values of $x_1, y_1, x_2, y_2$
Step 4. Calculate $\Delta x$ = absolute $(x_2 - x_1)$
        $\Delta y$ = absolute $(y_2 - y_1)$
Step 5. If $(x_2 > x_1)$
        assign $l_x = 1$
        else
        assign $l_x = -1$
Step 6. if $(y_2 > y_1)$
        assign $l_y = 1$
        else
        assign $l_y = -1$
```

```
Step 7. Plot $(x_1, y_1)$
Step 8. if $\Delta x > \Delta y$ (i.e., $|m| < 1$)
            compute $p_0 = 2\Delta y - \Delta x$
            starting at $k = 0$ to $\Delta x$ times, repeat
            if ($p_k < 0$)
            $x_{k+1} = x_k + 1$
            $y_{k+1} = y_k$
            $p_{k+1} = p_k + 2\Delta y$
            else
            $x_{k+1} = x_k + 1$
            $y_{k+1} = y_k + 1$
            $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
            plot($x_{k+1}, y_{k+1}$)
            else
            calculate $p = 2\Delta x - \Delta y$
            starting at $k = 0$ to $\Delta y$ times, repeat
            if($p_k < 0$)
            $x_{k+1} = x_k$
            $y_{k+1} = y_k + 1$
            $p_{k+1} = p_k + 2\Delta x$
            else
            $x_{k+1} = x_k + 1$
            $y_{k+1} = y_k + 1$
            $p_{k+1} = p_k + 2\Delta x - 2\Delta y$
            Plot($x_{k+1}, y_{k+1}$)
Step 9. Stop
```

Advantage of BLA over DDA:

- In DDA algorithm, each successive point is computed in floating point. So, it requires more time and more memory space. While in BLA, each successive point is calculated in integer value. So, it requires less time and less memory space.

- In DDA, since the calculated point value is in floating point number, it should be rounded at the end of calculation. But in BLA, round off is not necessary. So, there is no accumulation of rounding error.
- Due to rounding error, the line drawn by DDA algorithm is not accurate, while by BLA algorithm, line is accurate.
- DDA algorithm cannot be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse, and other curves.

Comparison between DDA and BLA

Base of comparison	Digital differential analyzer line drawing algorithm	Bresenham's line drawing algorithm
Arithmetic	DDA algorithm uses floating points	BLA uses integer
Operations	DDA algorithm uses multiplication and division in its operations	BLA uses only subtraction and addition in its operation
Speed	DDA algorithm is slower than BLA because it uses floating points	BLA is faster than DDA because it uses subtraction, addition and integer only
Accuracy and Efficiency	DDA algorithm is not as accurate and efficient as BLA as error and error accumulation occurs on it	BLA is more efficient and much accurate than DDA algorithm
Drawing	DDA algorithm can't draw curves and circles as accurate as by BLA	BLA can draw curves and circles much more accurately
Round off	DDA algorithm round off the co-ordinates to integer that is nearest to the line	BLA does not round off but takes the incremental value in its operation

Base of comparison	Digital differential analyzer line drawing algorithm	Bresenham's line drawing algorithm
Expensive	DDA is expensive as it uses floating point	BLA is cheaper than DDA as it uses only addition and subtraction

Code in C to draw a line using BLA

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<math.h>
void main()
{
 int gd = DETECT, gm, x1, y1, x2, y2, lx, ly, dy, dx, pk, i;
 initgraph (&gd, &gm, "c:\\tc\\bgi");
 printf ("put the values of x1 and y1\n");
  scanf ("%d %d",&x1, &y1);
  printf ("put the values of x2 and y2\n");
  scanf ("%d %d",&x2, &y2);
  dx = abs (x2-x1);
  dy = abs (y2-y1);
  if (x2 > x1)
 {
      lx=1;
      }
 else
      {
      lx=-1;
      }
if (y2 > y1)
      {
      ly=1;
```

```
      }
else
      {
      ly=-1;
      }
putpixel (x1,y1,RED);
if (dx > dy)
      {
      pk=2*dy-dx;
      for(i=0; i<dx; i++)
      {
      if (pk < 0)
      {
      x1 = x1 + lx;
      y1 = y1;
      pk = pk + 2*dy;
      }
else
      {
      x = x1 + lx;
      y1 = y1 + ly;
      pk = pk + 2*dy - 2*dx;
      }
putpixel (x1, y1, RED);
      }
}
else
      {
pk=2*dx-dy;
for(i=0; i<dy;i++)
{
```

```
if(pk<0)
{
x1=x1;
y1=y1+ly;
pk=pk+2*dx;
}
else
{
x1= x1 + lx;
y1= y1 + ly;
pk = pk + 2*dx-2*dy;
}
putpixel (x1,y1,RED);
}
}
getch();
closegraph ();
}
```

2.3 Circle

The equation of circle in Cartesian form is

$(x - x_c)^2 + (y - y_c)^2 = r^2$

$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$

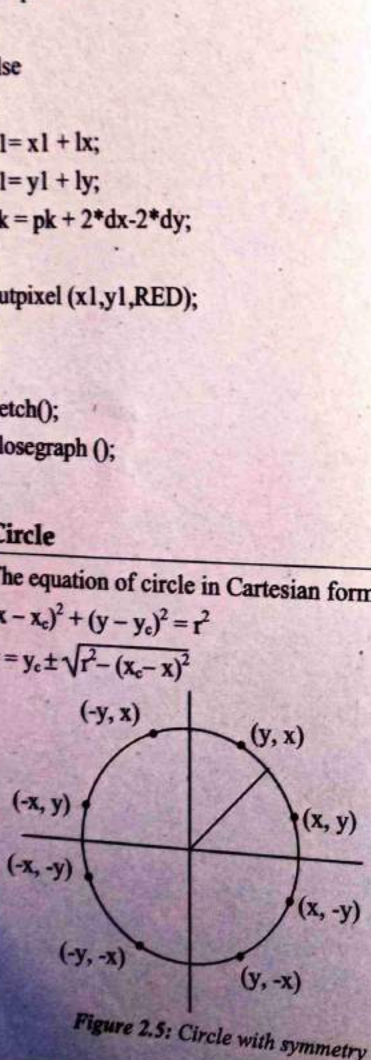


Figure 2.5: Circle with symmetry points

- We sample at unit intervals and determine the closest pixel position to the specified circle at each steps.
- Non uniform spacing of plotted pixel is a problem.
- Interchange the rate to x and y wherever the absolute value of slope of the circle tangent greater than1.
- To solve the computational complexity, we use symmetry of circle i.e., calculate for one octant and use symmetry for others.
- Bresenham's line algorithm for raster displays is adapted for circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.
- We test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.

2.3.1 Midpoint Circle Algorithm (Derivation)

The equation of circle is $x^2 + y^2 = r^2$

With center (0, 0) and radius r, let's define a circle function as circle $(x, y) = x^2 + y^2 - r^2$.

The distance of pixel to adjacent pixel is unit.

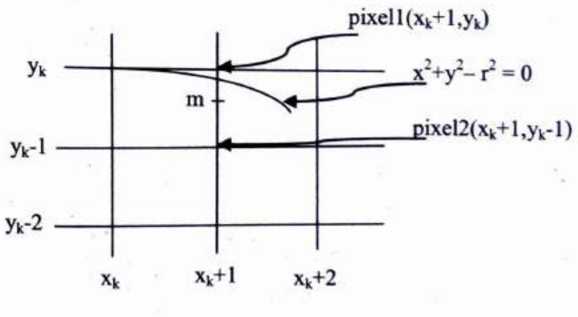


Figure 2.6: Mid-point circle pixels

In figure, length between pixel 1 and pixel 2 is $(y_k - y_{k+1}) = 1$

and half the length $= \frac{1}{2}$.

$$\text{Circle }(x,y)\begin{cases} < 0 \Rightarrow \text{if}(x,y) \text{ is inside the circle boundary} \\ = 0 \Rightarrow \text{if}(x,y) \text{is on the circle boundary} \\ > 0 \Rightarrow \text{if}(x,y) \text{is outside the circle boundary} \end{cases}$$

- Assume (x_k, y_k) is plotted, then next point close to the circle is (x_{k+1}, y_k) or (x_k+1, y_k-1)
- Decision parameter is the circle function evaluated at the midpoint between these two points.

$$p_k = f_{circle}(x_k+1, y_k - \frac{1}{2})$$

$$p_k = (x_k+1)^2 + (y_k - \frac{1}{2})^2 - r^2 \quad \text{..........(2)}$$

So, if $p_k<0$, then midpoint is inside the circle and y_k is closer to circle boundary. Else, midpoint is outside the circle. And y_k-1 is closer to the circle boundary

- Successive decision parameters are obtained using incremental calculations i.e., next decision parameter is obtained at

$$x_{k+1}+1 = x_k + 1 + 1 \text{ and } y_{k+1} - \frac{1}{2}$$

$$p_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$$

$$p_{k+1} = (x_k + 1 + 1)^2 + (y_{k+1} - \frac{1}{2})^2 - r^2 \quad \text{........(3)}$$

Subtracting equation (2) from (3)

$$p_{k+1} - p_k = (x_k + 1 + 1)^2 + (y_{k+1} - \frac{1}{2})^2 - r^2 - (x_k + 1)^2 - (y_k - \frac{1}{2})^2 + r^2$$

$$= (x_k + 1)^2 + 2(x_k + 1) + 1 + y_k^2 + 1 - 2y_{k+1} \times \frac{1}{2} + \frac{1}{4} -$$

$$(x_k + 1)^2 - y_k^2 + 2y_k \times \frac{1}{2} - \frac{1}{4}$$

$$= 2(x_k + 1) + (y_k^2 + 1 - y_k^2) - (y_{k+1} - y_k) + 1$$

Where y_{k+1} is either y_k or $y_k - 1$ depending on sign of p_k

If $p_k < 0$, then next pixel is at $(x_k + 1, y_k)$

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

If $p_k \geq 0$, then next pixel is at $(x_k + 1, y_k - 1)$

$$p_{k+1} = p_k + 2x_{k+1} + [(y_k - 1)^2 - y_k^2] - (y_k - 1 - y_k) + 1$$

$$= p_k + 2x_{k+1} + (y_k^2 - 2y_k + 1 - y_k^2) + 1 + 1$$

$$= p_k + 2x_{k+1} - 2y_k + 1 + 1 + 1$$

$$= p_k + 2x_{k+1} - 2(y_k - 1) + 1$$

$$= p_k + 2x_{k+1} - 2y_{k+1} + 1$$

Initial decision parameter

The initial decision parameter is obtained by evaluating the circle function at starting point $(x_0, y_0) = (0,r)$.

$$p_0 = f_{circle}(1, r - \frac{1}{2})$$

$$= 1^2 + (r - \frac{1}{2})^2 - r^2$$

$$= 1 + r^2 - 2r \times \frac{1}{2} + \frac{1}{4} - r^2$$

$$p_0 = \frac{5}{4} - r$$

If the radius r is specified as an integer, we can simply round to $p_0 = 1 - r$

Mid-point circle algorithm

Step 1. Start

Step 2. Declare variables x_c, y_c, r, x_0, y_0, p_0, p_k, p_{k+1}.

Step 3. Read values of x_c, y_c, r.

Step 4. Initialize the x_0 and y_0 i.e., set the co-ordinates for the first point on the circumference of the circle centered at origin as

$x_0 = 0$

$y_0 = r$

Step 5. Calculate initial value of decision parameter

$$p_0 = \frac{5}{4} - r$$

Step 6. At each x_k position, starting from $k = 0$

If $p_k < 0$

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k$

$p_{k+1} = p_k + 2x_{k+1} + 1$

else

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k - 1$

$p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$

Step 7. Determine the symmetry in other seven octants.

Step 8. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c)

Step 9. Plot the co-ordinates values

$x = x + x_c$

$y = y + y_c$

Step 10. Repeat steps 6 to 9 until $x \geq y$.

Step 11. Stop

2.4 Ellipse

An ellipse is defined as the set of points such that the sum of the distances from two fixed point/positions (foci) is same for all points

Major axis: The straight line segment extending from one side of the ellipse to the other through foci.

Minor axis: The shorter dimension of the ellipse, bisecting the major axis at the halfway position (ellipse center) between the two foci.

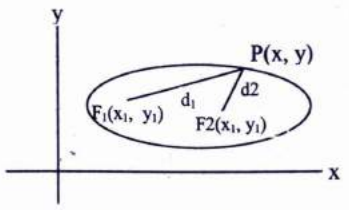


Figure 2.7: *Ellipse*

General equation: $\frac{(x - x_c)^2}{r_x^2} + \frac{(y - y_c)^2}{r_y^2} = 1$

In polar form,

$x = x_c + r_x \cos\theta$

$y = y_c + r_y \sin\theta$

2.4.1 Mid-Point Ellipse Algorithm

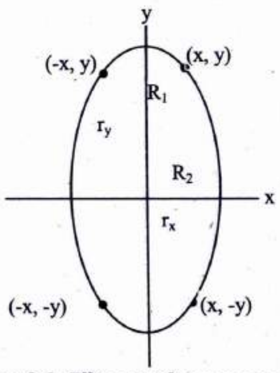


Figure 2.8: *Ellipse with symmetry points*

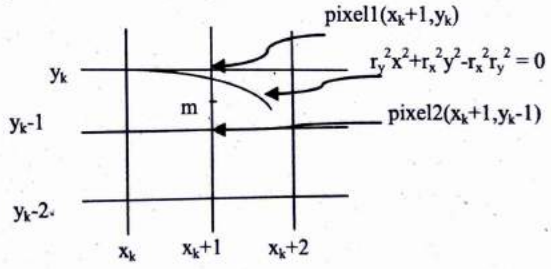


Figure 2.9: *Mid-point in region 1*

- Applied throughout the 1st quadrant in two parts
- Figure shows the division of 1st quadrant according to the slope of an ellipse with $r_x<r_y$
- Process this quadrant by taking unit steps in x-direction where slope of curve in y-magnitude less than 1, and taking unit step in y-direction where slope has magnitude greater than 1. Region 1 and 2 can be processed in different ways.

i) Start at position $(0, r_y)$ and step clock wise along the elliptical path in first quadrants, shifting from unit step in x to unit step in y when slope becomes greater than -1

ii) Alternatively, start at position $(r_x, 0)$ and select points in counter clockwise, shifting from unit step in y to unit step in x when the slope becomes less than-1.

Here, we start at position $(0, r_y)$

-we define an ellipse function with $(x_c, y_c) = (0,0)$

$$f_{ellipse}(x, y) = r_y^2x^2 + r_x^2y^2 - r_x^2r_y^2 \qquad \text{........(i)}$$

With properties

$$f_{ellipse}(x,y) = \begin{cases} < 0, if(x,y) \text{ is inside the ellipse boundary} \\ = 0, \quad if(x,y) \text{is on the ellipse boundary} \\ > 0, if(x,y) \text{is outside the ellipse boundary} \end{cases}$$

Thus ellipse function serves as the decision parameter.

- at each sampling position, we select the next pixel along the ellipse path according to the sign of the ellipse function evaluated at the midpoint between the two candidate pixels.
- at each step, we test the value of the slope of the curve,
- slope can be calculated as: $r_y^2x^2 + r_x^2y^2 = r_x^2r_y^2$

Differentiating with respect to x,

$$2r_y^2x + 2r_x^2y\frac{dy}{dx} = 0$$

$$\therefore \frac{dy}{dx} = -\frac{2r_y^2x}{2r_x^2y} \qquad \text{.......... (ii)}$$

At the boundary between region 1 and region 2, slope = −1.

So,

$$\frac{dy}{dx} = -1 = \frac{-2r_y^2x}{2r_x^2y}$$

or, $2r_y^2x = 2r_x^2y$

We move out of region 1, whenever

$2r_y^2x > 2r_x^2y$

We move out from region 2, whenever

$2r_y^2x < 2r_x^2y$

Assuming (x_k, y_k) has been illuminated (selected) we determine the next position along the ellipse path by evaluating the decision parameter at the midpoint $(x_k + 1, y_k - \frac{1}{2})$.

We have to determine the next point is $(x_k + 1, y_k)$ or $(x_k + 1, y_k - 1)$

We define decision parameter at mid-point as

$$p_{1k} = f_{ellipse}(x_k + 1, y_k - \frac{1}{2})$$

$$p_{1k} = r_y^2(x_k+1)^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2r_y^2 \text{.............(iii)}$$

If $p_{1k}<0$, the mid-point is inside the ellipse and the pixel on scan line y_k is closer to the ellipse boundary

Otherwise, the mid-point is outside or on the boundary and we select the pixel on scan line y_k-1

At the next sampling position $(x_{k+1}+1 = x_k+2)$, the decision parameter for region 1 is evaluated as

$$p_{1k+1} = f_{ellipse}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$$

$$p_{1k+1} = r_y^2[(x_k + 1) + 1]^2 + r_x^2(y_{k+1} + \frac{1}{2})^2 - r_x^2r_y^2 \text{.......(iv)}$$

Subtracting equation (iii) from (iv),

$$p_{1k+1} - p_{1k} = r_y^2[(x_k + 1 + 1)^2 - (x_k+1)^2] + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right] - r_x^2r_y^2 + r_x^2r_y^2$$

$$= r_y^2[(x_k + 1)^2 + 2(x_k + 1) + 1 - (x_k+1)^2] + r_x^2[y^2_{k+1} - 2y_{k+1}\times\frac{1}{2}+\frac{1}{4} - y_k^2 + 2y_k\times\frac{1}{2} - \frac{1}{4}]$$

$$= 2r_y^2(x_k + 1) + r_y^2 + r_x^2 + r_x^2\,[(y^2_{k+1} - y_k^2) - (y_{k+1} - y_k)]$$

$$= 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(y^2_{k+1} - y_k^2) - (y_{k+1} - y_k)]$$

Where y_{k+1} is either y_k or y_k-1, depending on the sign of p_{1k}

If $p_{1k} \le 0$ i.e., $y_{k+1} = y_k$, then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^2(x_k + 1) + r_y^2$$

If $p_{1k} > 0$ i.e., $y_{k+1} = y_k - 1$, then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^2(x_k + 1) + r_y^2 - 2r_x^2 y_{k+1}$$

The initial decision parameter is evaluated at start position $(x_0, y_0) = (0, r_y)$ as

$$p_{10} = f_{ellipse}\left(1,\ r_y - \frac{1}{2}\right)$$

$$= r_y^2 + r_x^2\left(x_y - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

$$p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 \text{.............(v)}$$

Region 2

We sample at unit steps in the negative y direction and the midpoint is now taken between horizontal pixels at each step. The decision parameter is

$$p_{2k} = f_{ellipse}\left(x_k + \frac{1}{2},\ y_k - 1\right)$$

$$p_{2k} = r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2 \text{............(vi)}$$

Put k = 0 for initial decision parameter for region 2

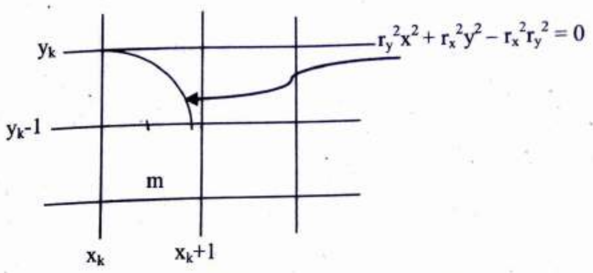


Figure 2.10: *Midpoint in region 2*

If $p_{2k} > 0$, the midpoint is outside the ellipse boundary and we select the pixel x_k. If $p_{2k} < 0$, the midpoint is inside or on the ellipse boundary and we select pixel position x_k+1.

Now, at next sampling position $y_{k+1} - 1 = y_k - 2$

$$p_{2k+1} = \text{fellipse}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$

$$p_{2k+1} = r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2(y_{k+1} - 1)^2 - r_x^2 r_y^2 \text{ ....(vii)}$$

Subtracting equation (vi) from (vii)

$$p_{2k+1} = p_{2k} + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] + r_x^2\left[(y_k - 1 - 1)^2 - (y_k - 1)^2\right]$$

$$= p_{2k} + r_x^2\left[(y_k - 1)^2 - 2(y_k - 1) + 1 - (y_k - 1)^2\right] + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

$$= p_{2k} + r_x^2[-2(y_k - 1) + 1] + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

If $p_{2k} > 0$, then

$$x_{k+1} = x_k$$

$$p_{2k+1} = p_{2k} - 2r_x^2 y_{k+1} + r_x^2$$

If $p_{2k} < 0$, then

$$x_{k+1} = x_k + 1$$

$$p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

Mid-point ellipse algorithm

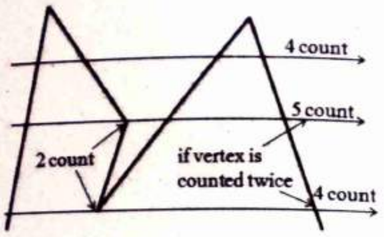
```
Step 1.  Start
Step 2.  Declare variables $x_c, y_c, r_x, r_y, x, y, p_0, p_k, p_{k+1}$
Step 3.  Read Values of $x_c, y_c, r_x, r_y$.
Step 4.  Obtain the first point on an ellipse centered on origin (x,
         y) by initializing the x and y as
             $x = 0$
             $y = r_y$
Step 5.  Calculate the initial value of the decision parameter in
         region 1 as
             $p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$
Step 6.  For each $x_k$ position in region 1, starting at k = 0, perform
         the following test.
         If $p_{1k}<0$,
             $x_{k+1} = x_k+1$
             $y_{k+1} = y_k$
             $p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2$
         else
             $x_{k+1} = x_k+1$
             $y_{k+1} = y_k-1$
             $p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$
         and continue until $2r_y^2 x \geq 2r_x^2 y$
Step 7.  Calculate the initial decision parameter in region 2 using
         the last point $(x_0, y_0)$ calculated in region 1 as
             $p_{20} = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0-1)^2 - r_x^2 r_y^2$
Step 8.  At each $y_k$ position in region 2, starting at k = 0, perform
         the following test.
         If $p_{2k}>0$,
             the next point along the ellipse centered on (0,0) is $(x_k, y_k-1)$ and
             $p_{2k+1} = p_{2k} - 2r_x^2 y_{k+1} + r_x^2$
         else
             the next point along the ellipse is
             $x_{k+1} = x_k+1$
             $y_{k+1} = y_k-1$
             $p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$
         Use the same incremental calculations for x and y as in
         region 1.
Step 9.  Determine the symmetry points in the other three
         quadrants.
Step 10. Move each calculated pixel position(x, y) onto the
         elliptical path centered on $(x_c, y_c)$ and plot the coordinate
         values.
             $x = x+ x_c$
             $y = y+ y_c$
Step 11. Repeat the steps for region 2 until $y<0$.
Step 12. Stop.
```

2.5 Filled Area Primitive

A standard output primitive in general graphics is solid color or patterned polygon area. Other kinds of area primitives are sometimes available, but polygons are easier to process since they have linear boundaries. The main idea behind the 2D or 3D object filling procedure is that it provides us more realism on the object of interest. There are two basic approaches to area filling in raster systems.

- One way to fill an area is to determine the overlap intervals for scan lines that crosses the area.
- Another method for area filling is to start from a given interior position and point outward from this until á specified boundary is met.

2.5.1 SCAN-LINE Polygon Fill Algorithm:

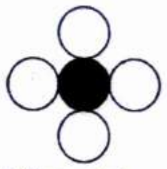


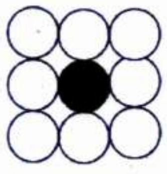
In scan-line polygon fill algorithm, for each scan-line crossing a polygon, it locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified color. In this method, the scan line must be even. At two edge connecting point called the vertex, we count the scan line two to handling the problem for scan line passing through vertex, but this consideration also may creates problem for some instant as shown in figure. To handle such problem shown by count-5, we keep the vertex blank and hence the count become 1, so that overall count in that scan line become even as shown in figure

2.5.2 Boundary-fill Algorithm:

In Boundary filling algorithm starts at a point called seed pixel inside a region and paint the interior outward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by until the boundary color is reached. Starting from (x, y), the procedure tests neighboring positions to determine whether they are of boundary color. If not, they are painted with the fill color, and their neighbors are tested. This process continues until all pixel up to the boundary color area have tested. The neighboring pixels from current pixel are proceeded by two method:

- 4-Connected (if they are adjacent horizontally and vertically.)
- 8-Connected (if they adjacent horizontally, vertically and diagonally.)


4- Connected


8- Connected

Algorithm for Boundary fill 4- connected:

```
void Boundary_fill4(int x,int y,int.b_color, int fill_color)
{
        int value=getpixel (x,y);
        if (value ! = b_color && value != fill_color)
        {
        putpixel (x,y,fill_color);
        Boundary_fill4(x-1,y, b_color, fill_color);
        Boundary_fill4(x+1,y, b_color, fill_color);
        Boundary_fill4(x,y-1, b_color, fill_color);
        Boundary_fill4(x,y+1, b_color, fill_color);
        }
}
```

Algorithm for Boundary fill 8- connected:

```
void Boundary-fill8(int x,int y,int b_color, int fill_color)
{
        int current =getpixel(x,y);
        if (current != b_color && current != fill_color)
        {
        putpixel (x,y,fill_color); Boundary_fill8(x-1,y,b_color,fill_color);
        Boundary_fill8(x+1,y,b_color,fill_color);
        Boundary_fill8(x,y-1,b_color,fill_color);
        Boundary_fill8(x,y+1,b_color,fill_color);
        Boundary_fill8(x-1,y-1,b_color,fill_color);
        Boundary_fill8(x-1,y+1,b_color,fill_color);
        Boundary_fill8(x+1,y-1,b_color,fill_color);
        Boundary_fill8(x+1,y+1,b_color,fill_color);
        }
}
```

Recursive boundary-fill algorithm not fills regions correctly if some interior pixels are already displayed in the fill color. Encountering a pixel with the fill color can cause a recursive branch to terminate, leaving other interior pixel unfilled. To avoid this we can first change the color of any interior pixels that are initially set to the fill color before applying the boundary fill procedure.

2.5.3 Flood-fill Algorithm:

Flood_fill Algorithm is applicable when we want to fill an area that is not defined within a single color boundary. If fill area is bounded with different color, we can paint that area by replacing a specified interior color instead of searching of boundary color value. This approach is called flood fill algorithm. We start from a specified interior pixel (x,y) and reassign all pixel values that are currently set to a given interior color with desired fill_color. Using either 4-connected or 8-connected region recursively starting from input position, The algorithm fills the area by desired color.

Algorithm:

```
void flood_fill4(int x,int y,int fill_color,int old_color)
{
        int current;
        current = getpixel (x,y); if (current == old_color)
        {
        putpixel (x,y,fill_color);
        flood_fill4(x-1,y, fill_color, old_color); flood_fill4(x,y-1,
         fill_color, old_color); flood_fill4(x,y+1, fill_color,
         old_color); flood_fill4(x+1,y, fill_color, old_color);
        }
}
```

SOLVED NUMERICAL AND DERIVATIONS

1. ***Consider a line from (3,7) to (8,3). Using simple DDA algorithm, rasterize this line.***

Solution:

Here,

Starting point $(x_1, y_1) = (3,7)$

Ending point $(x_2, y_2) = (8,3)$

$$\text{Slope (m)} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$= \frac{3-7}{8-3}$$

$$= -0.8$$

Since $|m|<1$, from DDA Algorithm we get,

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k + m$

k	x_k	y_k	x_{plot}	y_{plot}	(x_{k+1}, y_{k+1})
1	3	7	3	7	(3,7)
2	4	6.2	4	6	(4,6)
3	5	5.4	5	5	(5,5)
4	6	4.6	6	5	(6,5)
5	7	3.8	7	4	(7,4)
6	8	3	8	3	(8,3)

2. ***Use Bresenham's algorithm to scan convert a straight line connecting the end points (20, 10) and (30, 18).***

Solution:

$(x_0, y_0) = (20, 10)$

$\Delta x = |x_2 - x_1| = 10$

$\Delta y = |y_2 - y_1| = 8$

$p_0 = 2\Delta y - \Delta x$ (since $\Delta x > \Delta y$ i.e., $m < 1$)

$= 6$

For $\Delta x > \Delta y$,

if $p_k < 0$,

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k$

$p_{k+1} = p_k + 2\Delta y$

if $p_k > 0$,

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k + 1$

$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

k	p_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	−2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	−2	(28, 16)
8	14	(29, 17)
9	10	(30,18)

3. ***Determine the raster location along the circle octant in the first quadrant for a circle with radius 10.***

Solution:

$r = 10$

Initial decision parameter,

$p_0 = 1 - r = 1 - 10 = -9$

$(x_0, y_0) = (0,10)$

$2x_0 = 0, 2y_0 = 20$

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	−9	(1, 10)	2	20
1	−6	(2, 10)	4	20
2	−1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	−3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7,7)	14	14

4. ***Using midpoint circle algorithm, calculate the co-ordinate on the first quadrant of a circle having radius 6 and centre at (20, 10).***

Solution:

$r = 6$

$(x_2, y_2) = (20, 10)$

$p_0 = 1 - r = 1 - 6 = -5$

$(x_0, y_0) = (0,6)$

If $p_k < 0$

$(x_{k+1}, y_{k+1}) = (x_{k+1}, y_k)$

$p_{k+1} = p_k + 2x_{k+1} + 1$

Else

$(x_{k+1}, y_{k+1}) = (x_k + 1, y_k - 1)$

$p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$

For first octant,

k	p_k	x_{k+1}, y_{k+1}	$2x_{k+1}$	$2y_{k+1}$	Pixel to plot $x_{k+1}+x_c, y_{k+1}+y_k$
		(0,6)	0	12	(20,16)
0	−5	(1,6)	2	12	(21,16)
1	−2	(2,6)	4	12	(22,16)
2	0	(3,5)	6	10	(23, 15)

k	p_k	x_{k+1}, y_{k+1}	$2x_{k+1}$	$2y_{k+1}$	Pixel to plot $x_{k+1}+x_c, y_{k+1}+y_k$
3	−3	(4,5)	8	10	(24, 15)
4	9	(5,4)	10	9	x>y, not necessary to plot pixels

For second octant, using symmetry (x,y)→(y,x)

(0,6)→(6,0), (1,6) →(6,1),(2,6) →(6,2),(3,5) →(5,3) and (4,5) → (5,4).

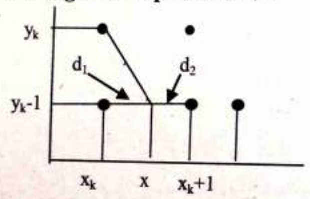
At last we need to shift the co-ordinates in the center (20, 10) so,

(6, 0) → (26, 10), (6, 1)→(26, 11), (6, 2)→(26, 12), (5, 3) → (25, 13) and (5, 4)→(25, 14)

5. ***Derive the Bresenham's decision parameter to draw a line moving from left to right and having negative slope. State the condition to identify you are in the second region of the ellipse using midpoint algorithm.*** *[2072 Kartik]*

Solution:

i) **For line with negative slope and $|m|>1$**



- Pixel positions are determined by sampling at unity intervals.
- Starting from left end position (x_0, y_0) of a given line, we step to each successive row (y position) and plot the pixel whose x value is closest to the line path.
- Assuming the pixel at (x_k, y_k) to be displayed is determined, we next need to decide which pixel to plot in row y_k-1.

- The candidate pixels are at position (x_k, y_k-1) and (x_k+1, y_k-1)
- At sampling position y_k-1,we label horizontal pixel separation from the mathematical line path d_1 and d_2.
- The x-coordinate on the mathematical line at pixel row position y_k-1 is calculated as

$y_k-1 = mx + b$

or, $x = \frac{y_k-1-b}{m}$

where $m = -\frac{\Delta y}{\Delta x}$ since m is negative.

Then,

$d_1 = x - x_k$..................(ii)

$d_2 = x_k + 1 - x$............(iii)

And,

$$\begin{aligned}
d_1 - d_2 &= x - x_k - x_k - 1 + x \\
&= 2x - 2x_k - 1 \\
&= 2\left(\frac{y_k-1-b}{m}\right) - 2x_k - 1 \\
&= 2\left(-\frac{\Delta x}{\Delta y}\right)\left(\frac{y_k-1-b}{2}\right) - 2x_k - 1
\end{aligned}$$

$(d_1-d_2)\Delta y = -2\Delta x(y_k-1-b) - (2x_k+1)\Delta y$

Now, defining decision parameter $p_k = \Delta y(d_1-d_2)$

$$\begin{aligned}
p_k = \Delta y(d_1-d_2) &= -2\Delta\times(y_k-1-b) - (2x_k+1)\Delta y \\
&= -2\Delta xy_k + 2\Delta x + 2\Delta\times b - 2\Delta yx_k - \Delta y \\
&= -2\Delta xy_k - 2\Delta yx_k + 2\Delta x + 2\Delta xb - \Delta y
\end{aligned}$$

$\therefore p_k = -2\Delta xy_k - 2\Delta yx_k + c$...............(iv)

Where $C = 2\Delta x + 2\Delta xb - \Delta y$ (All are constant)

Here, the sign of p_k is same as the sign of $d_1 - d_2$, as $\Delta y > 0$

Case I:

If $p_k \geq 0$, then $d_1 > d_2$ which implies that x_k+1 is nearer than x_k, so the next pixel to choose is (x_k+1, y_k-1)

Case II:

If $p_k < 0$, then $d_1 < d_2$ which implies that x_k is nearer than x_k+1, so the next pixel to select is (x_k, y_k-1)

Now,

Similarly, pixel at (y_k-2) can be determined whether it is (x_k+1, y_k-2) or (x_k+2, y_k-2) by looking the sign of deciding parameter p_{k+1} assuming pixel at (y_k-1) is known.

$p_{k+1} = -2\Delta x y_{k+1} - 2\Delta y x_{k+1} + C$

where C is same as in p_k.

Now,

$$\begin{aligned}
p_{k+1} - p_k &= -2\Delta x y_{k+1} + C + 2\Delta x y_k + 2\Delta y x_k - C \\
&= -2\Delta x(y_{k+1} - y_k) - 2\Delta y(x_{k+1} - x_k) \\
&= -2\Delta x(y_k - 1 - y_k) - 2\Delta y(x_{k+1} - x_k) \\
&= 2\Delta x - 2\Delta y(x_{k+1} - x_k)
\end{aligned}$$

This implies that decision parameter for the current row can be determined if the decision parameter of the last row is known.

Here (x_k+1-x_k) could either 0 or 1 which depends on sign of p_k.

If $p_k \geq 0$ (i.e., $d_1 > d_2$), $x_{k+1} = x_k+1$

Then, $p_{k+1} = p_k + 2\Delta x - 2\Delta y$

If $p_k < 0$, then $x_{k+1} = x_k$

$p_{k+1} = p_k + 2\Delta x$

Initial decision parameter

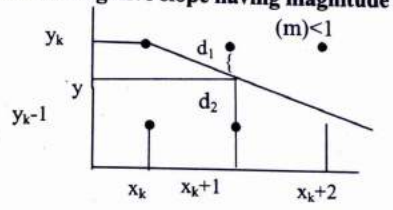
$p_k = -2\Delta x y_k - 2\Delta y x_k + 2\Delta x b - 2\Delta x - \Delta y$

Now,

$p_0 = -2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x b - 2\Delta x - \Delta y$

$$\begin{aligned}
&= -2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x\left(y_0 + \tfrac{\Delta y}{\Delta x} x_0\right) + 2\Delta x - \Delta y \\
&\quad \left[\because y = mx + b, b = y - mx, m = -\tfrac{\Delta y}{\Delta x}\right] \\
&= -2\Delta x y_0 - 2\Delta y x_0 + \Delta x\left(\tfrac{2\Delta x y_0 + 2\Delta y x_0}{\Delta x}\right) + 2\Delta x - \Delta y \\
&= -2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta x - \Delta y \\
&= 2\Delta x - \Delta y
\end{aligned}$$

ii) For line with negative slope having magnitude less than 1.



$d_1 = y_k - y$

$d_2 = y - (y_k - 1) = y - y_k + 1$

$d_1 - d_2 = y_k - y - y + y_k - 1 = 2y_k - 2y - 1$

Again, $y = m(x_k+1)+b$

$d_1 - d_2 = 2y_k - 2(m(x_k+1)+b] - 1$

$m = -\dfrac{\Delta y}{\Delta x}$ ($\because$ m is negative)

$d_1 - d_2 = 2y_k - 2\left(-\frac{\Delta y}{\Delta x}\right)(x_k+1) - 2b - 1$

$= \dfrac{2yk * \Delta x + 2\Delta y x k + 2\Delta y - \Delta x(2b) - \Delta x}{\Delta x}$

$p_k = (d_1 - d_2)\Delta x = 2\Delta x y_k + 2\Delta y x_k + 2\Delta y - \Delta x(2b) - \Delta x$

$p_k = 2\Delta x y_k + 2\Delta y x_k + c$

where $c = 2\Delta y - \Delta x(2b) - \Delta x$

For next decision parameter,

$p_{k+1} = 2\Delta x y_{k+1} + 2\Delta y x_{k+1} + c$

Now,

$p_{k+1} - pk = 2\Delta x(y_{k+1} - y_k) + 2\Delta y(x_{k+1} - x_k)$

if $p_k < 0$, $d1 < d2$

so,

$(x_{k+1}, y_{k+1}) = (x_k+1, y_k)$

So, $p_{k+1} = p_k + 2\Delta y$

if $p_k \geq 0$, $d1 \geq d2$

so, $(x_{k+1}, y_{k+1}) = (x_k+1, y_k-1)$

so, $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

Now, for initial decision parameter,

$p_k = p_0,\ x_k = x_0,\ y_k = y_0$

$$p_0 = 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - \Delta x(2\left(y_0 - \left(-\frac{\Delta y}{\Delta x}\right)x_0\right) + 1)$$

$$= 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - \Delta x\left(\frac{2\Delta x y_0 + 2\Delta y x_0 + \Delta x}{\Delta x}\right)$$

$$= 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$$

$$= 2\Delta y - \Delta x$$

6. ***Determine the pixel positions of following curve in first quadrant using mid-point algorithm.***

$$\frac{x^2}{64} + \frac{y^2}{36} = 1$$ *[2076 Ashwin Back]*

Solution:

Here, $r_x^2 = 64$, $r_y^2 = 36$

Here, Half of major axis $(r_x) = 8$

Half off minor axis $(r_y) = 6$

First point of the ellipse starting with $(0, r_y)$ is

$x_0 = 0$

$y_0 = r_y$

So, $x_0 = 0$, $y_0 = 6$

Now,

Initial decision parameter in region 1 is

$$P_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

$$= 6^2 - 8^2 \times 6 + \frac{1}{4} \times 8^2 = -332$$

Now, for each x_k position in region 1, starting at $k = 0$, we perform the following test

If $P_{1k} < 0$,

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k$

$P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} + r_y^2$

Else

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k - 1$

$P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$

and will continue until $2r_y^2 x \geq 2r_x^2 y$

Similarly, we calculate the initial decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$P_{20} = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

Now, at each y_k position in regions, starting at $k = 0$, we perform following test

If $P_{2k} > 0$,

$x_{k+1} = x_k$

$y_{k+1} = y_k - 1$

$P_{2k+1} = P_{2k} - 2r_x^2 y_{k+1} + r_x^2$

Else

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k - 1$

$P_{2k+1} = P_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

We repeat the steps for region 2 until $y < 0$

(x_k, y_k)	P_{1k}	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$	P_{1k+1}
(0,6)	–332	(1,6)	72	768	–224
(1,6)	–224	(2,6)	144	768	–44
(2,6)	–44	(3,6)	216	768	208
(3,6)	208	(4,5)	288	640	–108
(4,5)	–108	(5,5)	360	640	288
(6,5)	288	(6,4)	432	512	244
(6,4)	244	(7,3)	504	384	

The point (7, 3) will be the initial point for region 2. The initial decision parameter for region 2 is P_{2k}

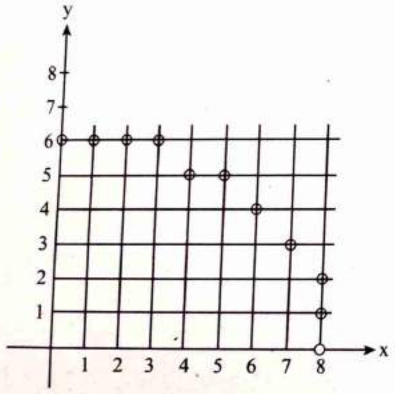
$$= 6^2\left(7 + \frac{1}{2}\right)^2 + 8^2(3-1)^2 - 8^2 * 6^2$$

$= -23$

(x_k, y_k)	P_{2k}	(x_{k+1}, y_{k+1})	P_{2k+1}
(7,3)	−23	(8,2)	361
(8,2)	361	(8,1)	297
(8,1)	297	(8,0)	

So, the pixel positions or points to plot of the given curve in first quadrant using mid-point algorithm are

(0,6), (1,6), (2,6), (3,6), (4,5), (5,5), (6,4), (7,3), (8,2), (8,1) and (8,0)



Chapter-3

Two-Dimensional Transformations

3.1 Introduction

Complex picture can be treated as a combination of straight line, circles, ellipse, etc. and if we are able to generate these basic figures, we can also generate combinations of them. Transformation means changing the graphics by changing the position, orientation or size of the original graphics by applying rules. When transformation occurs in 2D plane then it is called 2D transformations. Geometrical transformation is a mathematical procedure which changes/alters orientation, size, and shape of objects i.e., the co-ordinate description of objects.

3.2 Basic Transformations

The basic transformations are:

- Translation
- Scaling
- Rotation

3.2.1 Translation / Shifting

Translation repositions an object along a straight line path from one co-ordinate location to another. A two dimensional point can be translated by adding translation distances t_x and t_y to the original co-ordinate position to move the point to a new position (x', y')

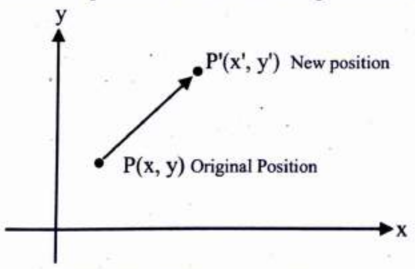


Figure 3.1: Translation of a point

$x' = x + t_x$

$y' = y + t_y$

Translation distance pair (t_x, t_y) is called translation vector or shift vector.

In matrix form, we can represent translation as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$P' = P + T$

In translation, just position is changed but shape and size are not changed. Straight line is translated by applying transformation equation to both end points and redrawing the line between those translated end points. Polygon is translated by adding translation vector to the co-ordinates position of each vertex and new polygon is regenerated using that new vertices.

Circle and ellipse are translated by translating the center co-ordinates and then, circle and ellipse are redrawn in new location.

3.2.2 Scaling

A scaling is a basic transformation that alters the size of object.

Points can be scaled by s_x along x axis and s_y along y axis in new points.

Transformation equations are:

$x' = x . s_x$

$y' = y . s_y$

In matrix form,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$P' = s.P$

If $s_x = s_y$, it is uniform scaling. If $s_x \neq s_y$, it is differential scaling.

If$(s_x, s_y) < 1$, then size is reduced.

$= 1$, size is unchanged (remains same).

>1, size is enlarged.

If scaling factor is less than 1, then it moves the object closer to origin. If scaling factor is greater than 1, then it moves the objects away from origin.

Fixed point scaling

The location of the scaled object can be controlled by choosing a position called fixed point that is to remain unchanged after the scaling transformation. Fixed point (x_f, y_f) can be chosen as one of the vertices, centroid of the object, or any other position.

Steps:

1. Translate object so that the fixed point coincides with the co-ordinate origin.
2. Scale the object with respect to the co-ordinate origin.
3. Use the inverse translation of steps 1 to return the object to its original position.

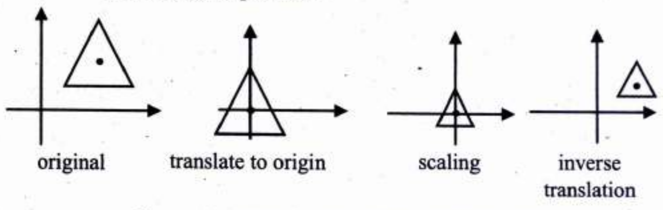


Figure 3.2: *Fixed point scaling of a triangle*

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

$T(x_f, y_f). S(s_x, s_y). T(-x_f, -y_f) = S(x_f, y_f, s_x, s_y)$

$x' = x_f + (x - x_f)\, s_x$

$y' = y_f + (y - y_f)\, s_y$

OR

$x' = x.s_x + x_f(1-s_x)$

$y' = y.s_y + y_f(1-s_y)$

where the terms $x_f(1-s_x)$ and $y_f(1-s_y)$ are constant for all points in object.

3.2.3 Rotation

Rotation repositions an object along a circular path in the xy plane. To generate rotation, we specify a rotation angle θand the position (x_r, y_r) of the rotation point about which the object is to be rotated. If θ is positive, object is rotated in counterclockwise direction and if θ is negative, object is rotated in clockwise direction.

1. **Transformation equation for rotation when pivot point is origin**

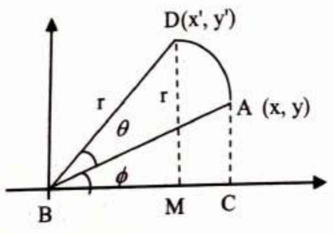


Figure 3.3(a): *Rotation of a point*

From figure, using standard trigonometric identities

In ΔABC,

$x = r\cos\phi,\ y = r\sin\phi$

In ΔDBM,

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

$= r\cos\theta\cos\phi - r\sin\theta\sin\phi$

$= x\cos\theta - y\sin\theta$

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

$= r\sin\theta\cos\phi + r\cos\theta\sin\phi$

$= x\sin\theta + y\cos\theta$

Representing in matrix form,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$P' = R.P$

2. **Rotation of a point about an arbitrary pivot position**
   1. Translation object so that pivot point is moved to co-ordinate origin.
   2. Rotate object about origin.
   3. Translate object so that pivot point is returned to its original position.

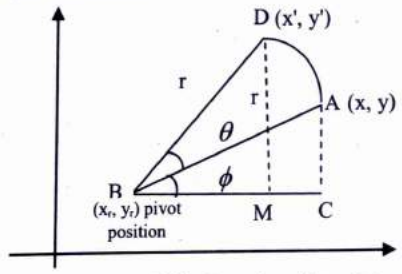


Figure 3.3(b): *Rotation of a point*

Composite matrix, C.M. = T'R(θ)T

$$= \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta)+y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta)-x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$

$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$

3.3 Homogeneous Co-ordinates and Matrix Representations

The matrix representation for translation, scaling, and rotation are respectively:

$P' = T + P$

$P' = S.P$

$P' = R.P$

Translation involves addition of matrices, whereas scaling and rotation involve multiplication. We may have to perform more than one transformation in same object like scaling the object, then rotate the same object, and finally translation. For this, first co-ordinate positions are scaled, then this scaled co-ordinates are rotated and finally translated. A more efficient approach would be to combine the transformation so that final positions are obtained directly from initial co-ordinates thereby eliminating the calculation of intermediate co-ordinates. This allows us to represent all geometric transformation as matrix multiplication.

We represent each Cartesian co-ordinate position (x, y) with homogeneous triple co-ordinate (x_h, y_h, h)

where $x = \frac{x_h}{h}, y = \frac{y_h}{h}$

Thus, general homogenous co-ordinate representation can also be written as

$(x_h, y_h, h) = (h.x, h.y, h)$

where h may be any non-zero value. But for convenience, h = 1 is used.

So, 2D position is represented with homogeneous co-ordinates (x, y, 1).

Expressing position in homogeneous co-ordinates allows us to represent all geometric transformation equation as matrix multiplication.

1. **Translation**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$P' = T(t_x, t_y).P$

Inverse of translation matrix can be obtained by replacing the translation parameter t_x and t_y with their negatives. i.e.,$-t_x$ and $-t_y$.

2. **Rotation**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$P' = R(\theta).P$

Inverse rotation matrix can be obtained by replacing θ with −θ.

3. **Scaling**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$P' = S(s_x, s_y).P$

Inverse translation matrix can be obtained by replacing s_x and s_y with $\frac{1}{s_x}$ and $\frac{1}{s_y}$ respectively.

Code in C for 2D Transformation

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void draw(int a[2][5]);
void translate(int a[2][5]);
void scale(int a[2][5]);
void rotate(int a[2][5]);
void main()
{
int a[2][5], i, gd = DETECT, gm;
initgraph(&gd,&gm,"c:\\tc\\bgi");
printf("Enter the coordinate positions of the lines");
for(i=0;i<5;i++)
{
```

```
scanf("%d %d",&a[0][i],&a[1][i]);
}

draw(a);
translate(a);
draw(a);
scale(a);
draw(a);
rotate(a);
draw(a);
getch();
closegraph();
}

void draw(int a[2][5])
{
int i=0;
for(i=0;i<4;i++)
{
line(a[0][i],a[1][i],a[0][i+1],a[1][i+1]);
}
}

void translate(int a[2][5])
{int i;
int tx,ty;
scanf("%d %d",&tx,&ty);
for(i=0;i<5;i++)
{
a[0][i]=a[0][i]+tx;
a[1][i]=a[1][i]+ty;
```

```
}
}

void scale(int a[2][5])
{int i;
int sx,sy;
scanf("%d%d",&sx,&sy);
for(i=0;i<5;i++)
{
a[0][i]=a[0][i]*sx;
a[1][i]=a[1][i]*sy;
}
}

void rotate(int a[2][5])
{int i, temp1, temp2;
float angle;
scanf("%f",&angle);
for(i=0;i<5;i++)
{
temp1=a[0][i];
temp2=a[1][i];
a[0][i]=temp1*cos(angle)-temp2*sin(angle);
a[1][i]=temp1*sin(angle)+temp2*cos(angle);
}
}
```

3.4 Composite Transformation

3.4.1 Translation

If two successive transformation vectors $(t_{x1}, t_{y1})$and (t_{x2}, t_{y2}) are applied to a point P, the final position or transformed location P' is calculated as:

$$P' = T(t_{x2}, t_{y2})\ \{T(t_{x1}, t_{y1}).P\}$$
$$= \{T(t_{x2}, t_{y2})\ T(t_{x1}, t_{y1})\}.P$$
$$= T(t_{x1} + t_{x2}, t_{y1} + t_{y2})\}.P$$

The composite transformation matrix for this sequence of transformation is

$$\begin{bmatrix} 1 & 0 & tx_2 \\ 0 & 1 & ty_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & tx_1 \\ 0 & 1 & ty_1 \\ 0 & 0 & 1 \end{bmatrix} . = \begin{bmatrix} 1 & 0 & tx_1 + tx_2 \\ 0 & 1 & ty_1 + ty_2 \\ 0 & 0 & 1 \end{bmatrix}$$

This shows that two successive translations are additive.

3.4.2 Rotation

$$P' = R(\theta_2).\{R(\theta_1).p\} = \{R(\theta_2).R(\theta_1)\}.P = R(\theta_1 + \theta_2) * P$$

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Successive rotations are additive.

3.4.3 Scaling

$$P' = S(Sx_2, Sy_2)\{S(Sx_1, Sy_1).P\} = \{S(Sx_2, Sy_2).S(Sx_1, Sy_1)\}.P = S(Sx_1, Sx_2, Sy_1.Sy_2).P$$

$$\begin{bmatrix} sx_2 & 0 & 0 \\ 0 & sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} sx_1 & 0 & 0 \\ 0 & sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sx_1.sx_2 & 0 & 0 \\ 0 & sy_1.sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Two successive scaling are multiplicative

Composite transformation indicates combination of different basic transformation in sequence to obtain desire result or combination could be a sequence of two or more successive transformation (e.g., two successive translation), two successive rotation or two or more successive scaling, etc.

3.5 Other Transformations

Other transformations are:

- Shearing
- Reflection

3.5.1 Shearing

A transformation that distorts the shape of the object such that the transformed shape appears as if the object was composed of internal layers that had been caused to slide over each other is called shearing.

i) **Shearing toward x-direction relative to x-axis is given by**

$x' = x + sh_x * y$

$y' = y$

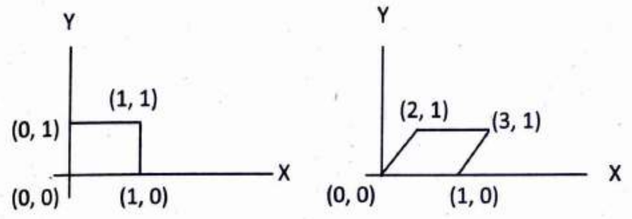


Figure 3.4: *Shearing towards x-direction*

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- sh_x is any real number
- co-ordinate position is shifted horizontally by an amount proportional to its distance y value from the axis.
- If sh_x is negative, object shearing is towards left.

ii) **Shearing towards y-direction relative to y-axis is given by**

$x' = x$

$y' = x \times sh_y + y$

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

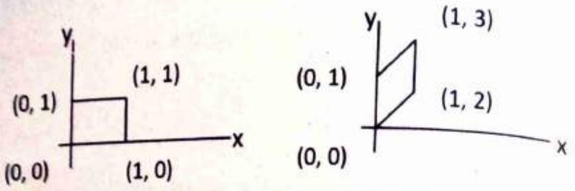


***Figure 3.5:** Shearing towards y-direction*

iii) Shearing in both direction is given by

$x' = x + sh_x \times y$

$y' = x \times sh_y + y$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

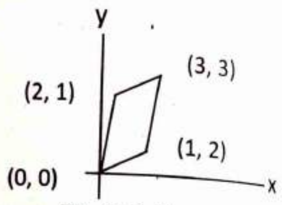


***Fig. 3.6:** Shearing towards both direction*

iv) x-direction shearing relative to other reference line is

$x' = x + sh_x.(y - y_{ref})$

$y' = y$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & -sh_x * yref \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

v) y-direction shearing relative to other reference line is

$x' = x$

$y' = sh_y(x - x_{ref}) + y$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y * x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This transformation shifts a co-ordinate position vertically by an amount proportional to its distance from the reference line $x = x_{ref}$.

3.5.2 Reflection

It is the transformation that produces mirror image of an object. The mirror image for a two-dimensional reflection is obtained by rotating the object 180^0 about the reflection axis.

i) Reflection about x-axis or about line y = 0

Keep 'x' value same but flip 'y' value

$x' = x$

$y' = -y$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

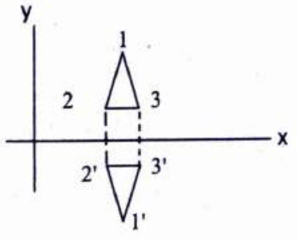


***Fig. 3.7:** Reflection about x-axis*

ii) Reflection about y-axis or about line x = 0

This transformation keeps y value same but flip x value. That is,

$x' = -x$

$y' = y$

$P' = R_{fy}.P$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

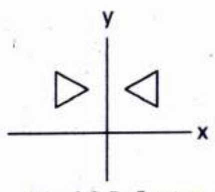


***Fig. 3.8:** Reflection about y-axis*

iii) Reflection about origin

It flips both x and y value i.e.,

$x' = -x$

$y' = -y$

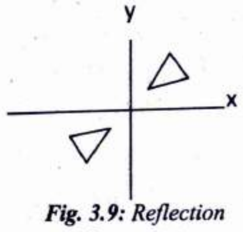


***Fig. 3.9:** Reflection about origin*

iv) Reflection about the line y = x

Steps:

1. Rotate about origin in clockwise direction by 45°, rotates the line y = x to x-axis.
2. Take reflection against x-axis

3. Rotate in anti-clockwise direction by same angle.

$Rf_{(y=x)} = R(\theta)^{-1} \times Rf_x \times R(\theta)$

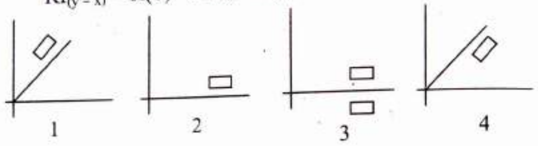


Figure 3.10: *Reflection about the line y = x*

$$R(\theta)^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Rf_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Rf_{(y=x)} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

v) **Reflection about the line y = –x**

Steps:

1. Rotate about origin in clockwise direction by 45°, in, rotates line y = x to y-axis.
2. Take reflection against y-axis.
3. Rotate in clockwise direction by same angle.

$Rf_{(y=-x)} = R(\theta)^{-1} \times Rf_y \times R(\theta)$

where

$$R(\theta)^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Rf_y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Rf_{y=-x} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

vi. **Reflection about y = mx+c**

Steps:

1. Translate line and object so that line passes through origin.

2. Rotate the line and object about origin until the line coincides with one of the co-ordinate axis.
3. Reflect the object through about that axis.
4. Apply inverse rotation about that axis.
5. Translate back to original location.

$C.M. = T^{-1}R(\theta)^{-1}R_f.R(\theta).T$

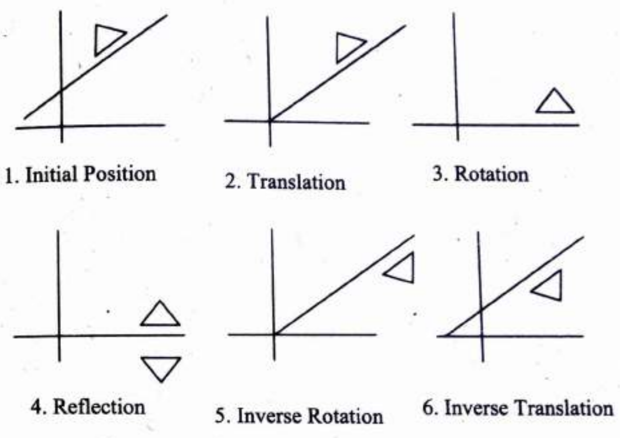


Figure 3.11: *Reflection about the line y = mx + c*

Here,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\text{OR, } \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1-m^2}{1+m^2} & \frac{2m}{1+m^2} & \frac{-2cm}{1+m^2} \\ \frac{2m}{1+m^2} & \frac{m^2-1}{1+m^2} & \frac{2cm}{1+m^2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3.6 Two-Dimensional Viewing

Two- Dimensional Viewing is the method about how to display 2D object in display device. It is the formal mechanism for displaying views of picture on an output device. Graphics package allows a user to specify which part of a defined picture is to be displayed and where is to be displayed in device. Any convenient Cartesian coordinate system, referred to as the world co-ordinate reference frame, can be used to define the pictures.

For a 2D picture, a view is selected by specifying a sub area of the total picture area. A user can select a single area for display, or several areas could be selected for simultaneous display or for an animated panning sequence across a scene. The picture parts within the selected areas are then mapped onto specified areas of the device co-ordinates. When multiple view areas are selected, these areas can be placed in separated display locations, or some areas could be inserted into other, larger display areas.

Transformation from world to device co-ordinate involve translation, rotation, and scaling operations as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area, also known as clipping. Windowing is the process of selecting and enlarging the portions of a drawing.

3.7 Coordinate Representation

With few exceptions general packages are designed to be used with Cartesian co-ordinate system. If in other coordinate system, it must be converted

Modeling coordinate/local coordinate/master coordinate

Every individual object or model has its own coordinate system. This coordinate system is called modeling coordinate or local co-ordinate or master co-ordinate. The individual object has its own dimension and its image can be constructed in separate coordinate system. It is known as modeling coordinate.

World coordinate

Once individual objects have been identified or specified, those objects are placed into appropriate position within a scene using reference frame to interact each other. This reference frame is called world co-ordinate. It contains many objects with one unit.

Viewing coordinate

Viewing coordinate is used to define window in the world co-ordinate plane with any possible orientation, i.e. viewing some objects or items at a time.

Normalized device coordinate

In normalized coordinate the coordinates are in range 0 to 1. Generally, a graphical system, world coordinate positions are converted to normalized device coordinates before final conversion to specified device coordinate. This makes the system independent of the various devices that might be used at a particular workstation.

Device coordinate

When the world coordinate description of the scene is transformed to one or more output device reference frame for display; the display coordinate system is referred to a device coordinate or screen coordinates in the case of video monitor.

3.8 The Viewing Pipeline

Window

A world coordinate area selected for display is called window.

View port

An area on a display device to which a window is mapped is called view port.

Window defines what is to be viewed. View port defines where is to be displayed. Window and viewport are rectangular in standard position, with the rectangular edge parallel to co-ordinate axis. Other shapes are also possible but take longest time to process.

The mapping of a part of world co-ordinate scene to device co-ordinates is referred to as a viewing transformation. Sometimes, 2D viewing transformation is simply referred to window to view port or windowing transformation.

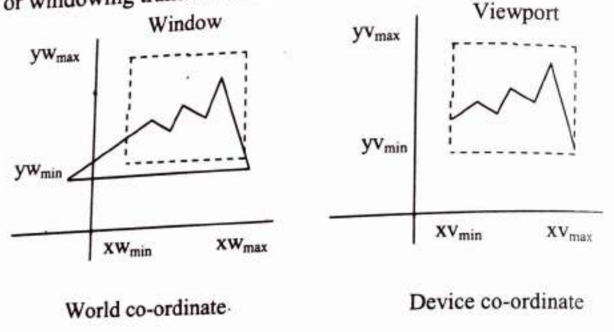
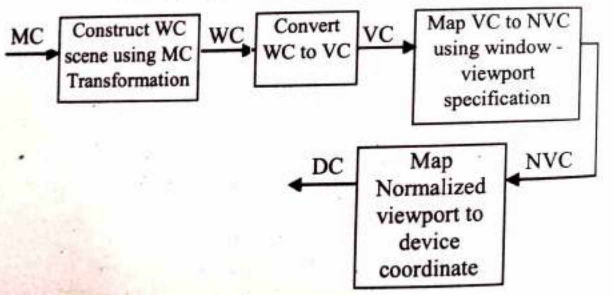


***Figure 3.12:** A viewing transformation using standard rectangles for the window and viewport.*



MC
Construct WC scene using MC Transformation
WC
Convert WC to VC
VC
Map VC to NVC using window - viewport specification
NVC
Map Normalized viewport to device coordinate
DC

***Figure 3.13:** 2D viewing transformation pipeline*

As in the real life, we see through a small window or the view finder of a camera, a computer-generated image often depicts a partial view of a large scene. Objects are placed into the scene by modeling transformations to a master coordinate system, commonly referred to as the world coordinate systems (WCS).

A rectangular window with its edges parallel to the axis of the WCS is used to select the portion of the scene for which an image is to be generated (displayed).Sometimes an additional coordinate system called the viewing coordinate system (VCS) is introduced to simulate the effort of moving or/and tilting the camera. On the other hand, an image representing a view often becomes part of a larger image, like a photo on an album page, which models a computer monitor's display area.

Since monitor sizes differ from one system to another, we introduce a device-independent tool to describe the display area called normalized device coordinate system (NDCS) in which a unit (1X1) square whose lower left corner is at the origin of the coordinate system that defines the display area of the display device. A rectangular viewport with its edges parallel to the axes of the NDCS is used to specify a sub-region of the display area that embodies the image. The process that convert object coordinates in WCS to normalized device coordinate is called window to viewport mapping or normalization transformation.

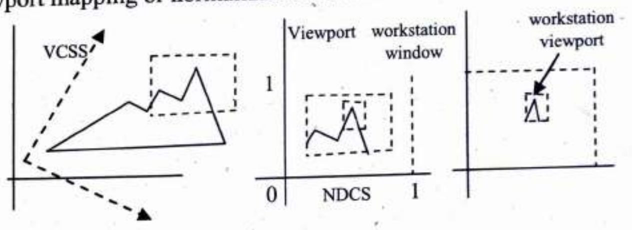


***Figure 3.14:** WCS, VCS, NDCS and workstation*

The process of mapping normalized device coordinates to discrete device coordinates is called workstation transformation, which is essentially a second window to viewport mapping, with a workstation window in the normalized device coordinate system and a workstation viewport in the device coordinate system. Collectively these two coordinate mapping operations are referred to as viewing transformation.

3.9 Window to Viewport Mapping (Coordinate Transformation)

We transfer object description to normalized device coordinates using a transformation that maintains the same relative placement of objects in normalized space they had in viewing coordinates.

If a coordinate position is at the center of the viewing window for instance it will be displayed at the center of viewport.

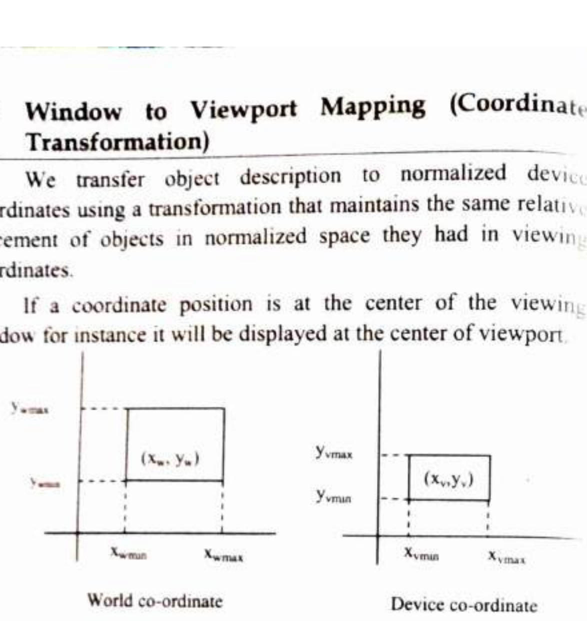


Figure 3.15: *Window to viewport mapping*

A window is specified by four world co-ordinates x_{wmin}, x_{wmax}, y_{wmin}, y_{wmax}. A viewport is described by four device co-ordinate x_{vmin}, x_{vmax}, y_{vmin}, y_{vmax}. To maintain the same relative placement in the viewport as in window, we require.

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

And

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

Solving the equation for viewport

$x_v = x_{vmin} + (x_w - x_{wmin})s_x$

$y_v = y_{vmin} + (y_w - y_{wmin})s_y$

where, scaling factor,

$$S_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$S_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

Steps (for Window to Viewport Transformation):

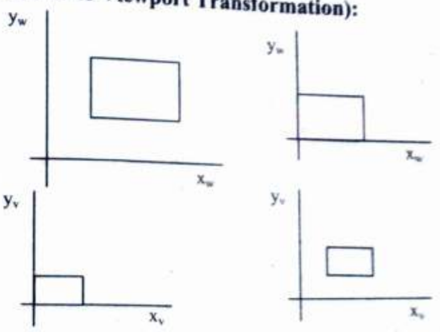


Figure 3.16: *Window to viewport transformation*

1. The object together with its window is translated until the lower left corner of the window is at origin.
2. Object and window are scaled until window has dimension of viewport.

   Perform a scaling transformation using a fixed-point position of (x_{wmin}, y_{wmin}) that scales the window area to the size of the viewport
3. Again, translate to move viewport to its correct position.

Viewing Transformation:

1. Translate window to origin by

   $T_x = -x_{wmin}$

   $T_y = -y_{wmin}$
2. Scale window such that its size is matched to viewport.

$$S_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$S_y = \frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}}$$

3. Retranslate it by

$T_x = x_{v\min}$

$T_y = y_{v\min}$

Composite matrix (CM) = $T_v \times S_{wv} \times T_w$

$$T_w = \text{Translate window to origin} = \begin{bmatrix} 1 & 0 & -x_{w\min} \\ 0 & 1 & -y_{w\min} \\ 0 & 0 & 1 \end{bmatrix}$$

S_{wv} = Scaling of window to viewport

$$= \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

T_V = Translate viewport to original position

$$= \begin{bmatrix} 1 & 0 & x_{v\min} \\ 0 & 1 & y_{v\min} \\ 0 & 0 & 1 \end{bmatrix}$$

Q. ***Consider the window is located from $X_{wmin} = 20$, $X_{wmax} = 80$, $Y_{wmin} = 40$, $Y_{wmax} = 80$, and a point is located in 30, 80. Identify the new location of the point in the view port considering viewport size $X_{vmin} = 30$, $X_{vmax} = 60$, $Y_{vmin} = 40$, $Y_{vmax} = 60$.***

Solution:

$X_{wmin} = 20$	$X_{wmax} = 80$
$Y_{wmin} = 40$	$Y_{wmax} = 80$
$X_{vmin} = 30$	$X_{vmax} = 60$
$Y_{vmin} = 40$	$Y_{vmax} = 60$

$$S_x = \frac{X_{vmax} - X_{vmin}}{X_{wmax} - X_{wmin}} = \frac{60-30}{80-20} = \frac{3}{6} = \frac{1}{2}$$

$$S_y = \frac{Y_{vmax} - Y_{vmin}}{Y_{wmax} - Y_{wmin}} = \frac{60-40}{80-40} = \frac{2}{4} = \frac{1}{2}$$

$X_v = X_{vmin} + (X_w - X_{wmin})S_x$

$= 30 + (30-20)\frac{1}{2} = 30 + 5 = 35$

$Y_v = Y_{vmin} + (Y_w - Y_{wmin})S_y$

$= 40 + (80-40)\frac{1}{2} = 60$

By 2nd method

1. Translate window to origin

   $T_x = -X_{wmin}$

   $T_y = -Y_{wmin}$

2. Scale window such that its size is matched to viewport

   $$S_x = \frac{X_{vmin} - X_{vmin}}{X_{wmax} - X_{wmin}}$$

   $$S_y = \frac{Y_{vmax} - Y_{vmin}}{Y_{wmax} - Y_{wmin}}$$

3. Retranslate it

   $T_x = X_{vmin}$

   $T_y = Y_{vmin}$

$$\text{C.M.} = \begin{bmatrix} 1 & 0 & +X_{vmin} \\ 0 & 1 & +Y_{vmin} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -X_{wmin} \\ 0 & 1 & -Y_{wmin} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 30 \\ 0 & 1 & 40 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -20 \\ 0 & 1 & -40 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 30 \\ 0 & 1 & 40 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 & -10 \\ 0 & \frac{1}{2} & -20 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2} & 0 & -10+30 \\ 0 & \frac{1}{2} & -20+40 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 20 \\ 0 & \frac{1}{2} & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P^1 = C.M. \times P$$

$$= \begin{bmatrix} \frac{1}{2} & 0 & 20 \\ 0 & \frac{1}{2} & 20 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 30 \\ 80 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 15+20 \\ 40+20 \\ 1 \end{bmatrix} = \begin{bmatrix} 35 \\ 60 \\ 1 \end{bmatrix}$$

$$P^1 = (35, 60)$$

3.10 Clipping Operations

3.10.1 Introduction

Procedures that identifies those portions of a picture that are either inside or outside of a specified region of a specified region of space is referred to as a clipping algorithm, or simply clipping. The region against which an object is clipped is called a 'clip window'.

3.10.2 Applications of clipping

Clipping is used for extracting part of a defined scene to view and identifying visible surfaces. It is used for drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing or duplicating. Clipping is used for creating objects using solid-modeling procedures. Clipping algorithm can be applied to window coordinates, so that only the content of the window interior is mapped to device coordinates. Alternatively, the complete world co-ordinate picture can be mapped first to device coordinate or NDC, then clipped against viewport boundaries.

WC clipping removes those primitives outside the window form further consideration, thus eliminating the processing necessary to transform those primitives to device space necessary to transform those primitives to device space. Viewport clipping on the other hand, can reduce calculations by allowing concatenation of viewing and geometric transformation matrices. But viewport clipping does require that the transformation to device co-ordinates

be performed for all objects, including those outside the window area.

3.10.3 Some Primitive Types of Clipping

- Point clipping
- Line clipping (straight line segment)
- Area clipping (polygon)
- Curve clipping
- Text clipping

1. Point Clipping

Assuming that the clip window is a rectangle in standard position, we save a point P(x, y) for display, if the following inequalities are satisfied.

$$xw_{\min} \le x \le xw_{\max}$$

$$yw_{\min} \le y \le yw_{\max}$$

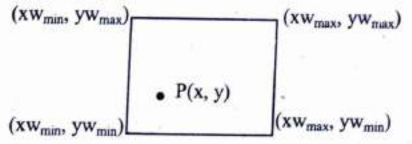
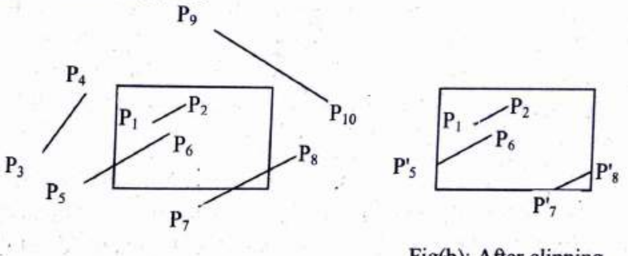


Figure 3.17: Point clipping

The edges of the clip window can be either WC window boundary or viewport boundaries. If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).

2. Line Clipping



Fig(a): Before Clipping

Fig(b): After clipping

Figure 3.18: Line clipping

Line clipping against rectangular clip window

A line-clipping procedure involves several parts. First, a given segment is tested to determine whether it lies completely inside the clipping window. If it does not, it is tested to determine whether it lies completely outside the window. Finally, if we cannot identify a line as complete inside or outside, we must perform intersection calculation with one or more clipping boundaries.

We process line through the 'Inside-outside" test by checking the line endpoints. A line with both endpoints outside anyone of the clip boundaries (line P_3 to P_4) is outside the window. All other lines cross one or more clipping boundaries and may require calculation of multiple intersection points. For a line segment with end points (x_1, y_1) and (x_2, y_2) and one or both end points outside the clipping rectangle, the parametric representation could be used

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

$y = y_1 + u(y_2 - y_1)$, where $0 \le u \le 1$

If the value of u is outside the range 0 to 1, the line does not enter the interior of the window at that boundary. If the value of u is within the range from 0 to 1, the line segment does indeed cross into the clipping area. This method can be applied to each clipping boundary edge in turn to determine whether any part of the line segment is to be displayed.

Cohen-Sutherland Line Clipping

Cohen-Sutherland line clipping algorithm is one of the oldest and most popular procedures. It uses bit operation to perform this test. It speeds up the processing of line segments by performing tests that reduce the no. of intersection that must be calculated.

For each end-point, a 4 digits binary code (called a region **code) is assigned to identify** the location relative to boundary. **Lower order bit (bit1) set to '1' if end point** is at the left side of the **window, else set to '0' (by numbering the bit position** in the region **code as 1 to 4 from right to left. Bit 2 is** set to '1' if end point lies at

right side else set '0'. Bit 3 is set to '1' if end point lies bottom, else set '0'. Bit 4 is set to '1' if end point lies top, else set '0'.

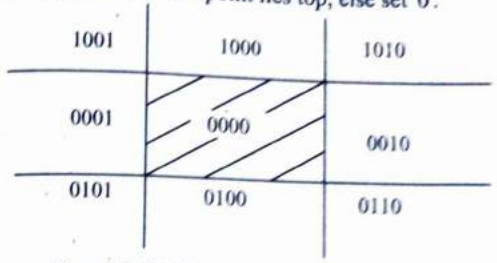


Figure 3.19: Clip window and region code

Algorithm

Step 1: Assign Region Code (TBRL)

Step 2: Establish region code for all line end points.

Bit 1 is set to '1' if $x < x_{min}$ else set to '0'.

Bit 2 is set to '1' if $x > x_{max}$ else set to '0'.

Bit3 is set to '1' $y < y_{min}$ else set '0'

Bit 4 is set to '1' $y > y_{max}$ else set to '0'

Step 3: Determine whether line is completely inside or outside window using test.

a) If both end pint have region code ' 0000' line is completely inside.

b) If logical AND of end points of a line not '0000' line is completely outside.

Step 4. If both condition of step 2 fails. i.e. Logical AND give '0000' we need to find the intersection with window boundary.

Here,

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

a) If bit 1 is 1, line intersection with left boundary, so,

$y_i = y_1 + m(x - x_1)$ where $x = x_{min}$

b) If bit 2 is 1, line intersect with right boundary, so,

$y_i = y_1 + m(x - x_1)$ where $x = x_{max}$

c) If bit 3 is 1, line intersect with lower boundary, so,

$$x_i = x_1 + \frac{1}{m}(y - y_1) \text{where } y = y_{min}$$

d) If bit 4 is 1, line intersects with upper boundary, so,

$$x_i = x_1 + \frac{1}{m}(y - y_1) \quad \text{where } y = y_{max}$$

Here, x_i and y_i are x, y intercepts for that line, update

Step 5: Repeat step 2 and 4 till completely accepted

Liang-Barsky Line Clipping

Faster line clippers have been developed that are based on analysis of the parametric equation of a line segment, which can be written in the form,

$x = x_1 + u\,\Delta x$

$y = y_1 + u\,\Delta y,\ 0 \le u \le 1$

where $\Delta x = x_2 - x_1, \Delta y = y_2 - y_1$

Using these parametric equations, Cyrus and Beck developed an algorithm that is generally more efficient than the Cohen-Sutherland algorithm. Later, Liang and Barsky independently devised an even faster parametric line-clipping algorithm. Following the Liang-Barsky approach, we first write the point clipping condition in the parametric from,

$xw_{min} \le x_1 + u\,\Delta x \le xw_{max}$

$yw_{min} \le y_1 + u\,\Delta y \le yw_{max}$

Now, we can write

$-u\,\Delta x \le x_1 - xw_{min}$

$u\,\Delta x \le xw_{max} - x_1$

$-u\,\Delta y \le y_1 - yw_{min}$

$u\,\Delta y \le yw_{max} - y_1$

Each of these four inequalities can be expressed $u.p_k \le q_k$, k=1, 2, 3, 4where parameters p and q are defined as

k = 1 (is the line inside the left boundary) $p1 = -\Delta x$, $q1 = x_1 - x_{wmin}$

k = 2 (is the line inside the right boundary)$p2 = \Delta x$, $q2 = x_{wmax} - x_1$

k = 3 (is the line inside the bottom boundary)$p3 = -\Delta y$, $q3 = y_1 - y_{wmin}$

k = 4 (is the line inside the top boundary) $p4 = \Delta y$, $q4 = y_{wmax} - y_1$

when $P_k < 0$, the infinite extension of line proceeds from the outside to inside of the infinite extension of this particular clipping boundary. When $P_k > 0$, the line proceeds from inside to outside

Trivial Rejection

The line with $p_k = 0$ for some k and one $q_k < 0$ for these k is rejected. For line with $p_k = 0$ for some k and all $q_k >= 0$ for those k, line is parallel to one of the clipping boundary and some portion of the line is inside. For intersection with boundary, the parameters are supposed to be r_k, r_k is given by

$$r_k = \frac{q_k}{p_k}$$

Clipped line will be s

$x_1' = x_1 + u_1\Delta x$

$y_1' = y_1 + u_1\Delta y$

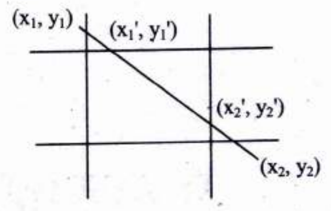


Figure 3.20: Clip window, line and intersection points

The value of r_k becomes candidate for $u_1$if $pk < 0$. The value of u1 is greater or equal to 0. It is for intersection with the boundaries to which line enters the boundary = maximum value between 0 and r.

$x_2' = x_1 + u_2\Delta x$

$y_2' = y_1 + u_2\Delta y$

u_2 is lesser or equal to 1. It is for intersection with the boundaries to which line leaves the boundary. Its minimum value is between r and 1.

Algorithm

1) If $p_k = 0$ for some k, then the line is parallel to the clipping boundary now test q_k
   For these k, If one $q_k < 0$ then the line is completely outside. And can be eliminated.
2) For non zero p_k, calculate $r_k = \frac{q_k}{p_k}$, it gives the intersection point.
2) If $p_k<0$ then line proceeds from outside to inside boundary. Calculate $u_1 = \max(0, \{r_k : r_k = \frac{q_k}{p_k}\})$ to determine intersection point with the possible extended clipping boundary k and obtain a new starting point for the line at u_1.
3) If $p_k>0$ then line proceeds from inside to outside the boundary.Calculate $u_2 = \min(1, \{r_k : r_k = \frac{q_k}{p_k}\})$ to determine intersection point with extended clipping boundary k and obtain a new point at u_2
4) If $u_1>u_2$, then the line is outside and therefore rejected or line is discarded.
5) The line is now between $[u_1, u_2]$

SOLVED NUMERICALS AND DERIVATIONS

1. ***Rotate a triangle A(5,6), B(6,2) and C(4,1) by 45 degree about an arbitrary point (3,3).*** *[2076 Ashwin Back]*

Solution:

Composite matrix

$$= T_{(3,3)} R_{(45)} T_{(-3,-3)}$$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos45° & \sin45° & 0 \\ \sin45° & \cos45° & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, $P^1 = C.M. * P$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 3 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{-6}{\sqrt{2}}+1 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 5 & 6 & 4 \\ 6 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{1}{\sqrt{2}}+3 & -\frac{4}{\sqrt{2}}+3 & 3 \\ \frac{5}{\sqrt{2}}+1 & \frac{2}{\sqrt{2}}+1 & -\frac{1}{\sqrt{2}}+1 \\ 1 & 1 & 1 \end{bmatrix}$$

2. ***Consider a triangle A(0,0), B(1,1), C(5,2). The triangle has to be rotated by an angle 45° about the point P(–1,–1). What will be the co-ordinate of new triangle.***

Solution:

$$CM = T_{(-1,-1)} R_{45} T_{(1,1)}$$

$$= \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos45° & -\sin45° & 0 \\ \text{sun}45° & \cos45° & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$P^1 = C.M \times P$

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

$$= \begin{bmatrix} -1 & -1 & \frac{3}{\sqrt{2}}-1 \\ \sqrt{2}-1 & 2\sqrt{2}-1 & \frac{9}{\sqrt{2}}-1 \\ 1 & 1 & 1 \end{bmatrix}$$

$A^1 = (-1, \sqrt{2} - 1)$

$B^1 = (-1, 2\sqrt{2} - 1)$

$C^1 = \left(\frac{3}{\sqrt{2}} - 1, \frac{9}{\sqrt{2}} - 1\right)$

3. ***Scale an object (4, 4), (3, 2), (5, 2) about a fixed point** (4, 3) **by 2.***

Solution:

$P' = C.M. \times P$

$$C.M. = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -4 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & -8 \\ 0 & 2 & -6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & -8-4 \\ 0 & 2 & -6-3 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & -4 \\ 0 & 2 & -3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} 4 & 3 & 5 \\ 4 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$P' = C.M. \times P$

$$= \begin{bmatrix} 2 & 0 & -4 \\ 0 & 2 & -3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 & 5 \\ 4 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 8-4 & 6-4 & 10-4 \\ 8-3 & 4-3 & 4-3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 & 6 \\ 5 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

So, scaled points are (4, 5), (2, 1) and (6, 1)

4. ***Rotate the triangle (5,5), (7,3), (3,3) about fixed points** (5,4) **in counter clockwise by 90°.***

Solution:

$C.M. = T_{(5,4)} R_{90} T_{(-5,-4)}$

$$= \begin{bmatrix} 1 & 0 & 5 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 90^0 & -\sin 90^0 & 0 \\ \sin 90^0 & \cos 90^0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & 9 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Now $P' = C.M. * P$

$$= \begin{bmatrix} 0 & -1 & 9 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 7 & 3 \\ 5 & 3 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -5+9 & -3+9 & -3+9 \\ 8-1 & 7-1 & 3-1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 6 & 6 \\ 4 & 6 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$\therefore$ New co-ordinates are (4,4), (6,6) (6,2)

5. ***Reflect an object (2, 3), (4, 3), (4, 5) about line $y = x + 1$.***

Solution:

Here, m = 1

c = 1

$C.M. = T' R_\theta' Rf_x . R_\theta . T$

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_\theta = \begin{bmatrix} \cos 45^0 & \sin 45^0 & 0 \\ -\sin 45^0 & \cos 45^0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{f_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_\theta' = \begin{bmatrix} \cos 45^0 & -\sin 45^0 & 0 \\ \sin 45^0 & \cos 45^0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$P' = C.M. \times P$

$$= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 \\ 3 & 3 & 5 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 4 \\ 3 & 5 & 5 \\ 1 & 1 & 1 \end{bmatrix}$$

Required points are(2,3), (2,5), (4,5)

6. ***Reflect an object (2,3), (4,3) and (4,5) about line y = 2x+1***

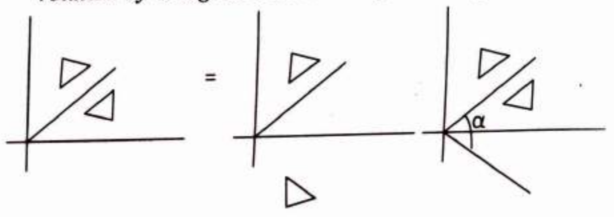
Solution:

Hints:

Here, m = 2, c = 1

1. Translate with value of c
2. Rotate by angle $\tan^{-1}(m)$
3. Reflect about x axis
4. Again re-rotate
5. Translate

7. ***The reflection along the line y = x is equipment to the reflection along the x axis followed by counter clockwise rotation by α degree. Find the angle α.*** *[2071 Chaitra]*



Solution:

Steps for reflection about line y = x

i) Rotate about origin in clockwise direction by 45°

ii) Reflection about x –axis

iii) Rotate in anticlockwise direction in 45°

$Rf_{(y=x)} = R(\theta)^{-1} \times Rf_x \times R(\theta)$

$$= \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{.....................(i)}$$

Again, the matrix for the reflection along x-axis followed by counter clockwise rotation by α degree

$$= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ \sin\alpha & -\cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{...........(ii)}$$

Now equating matrix (1) and (2)

$\cos\alpha = 0$

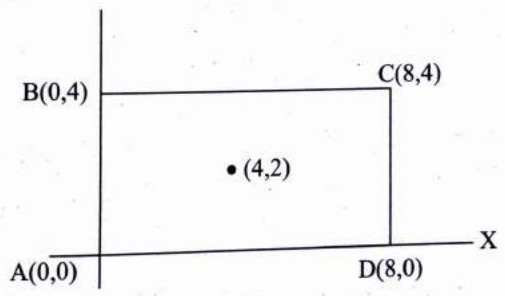
$\sin\alpha = 1$

$-\cos\alpha = 0$

$\because \alpha = 90$

8. ***Find the transformation matrix that transforms the rectangle ABCD whose center is at (4, 2) is reduced to half of its size, the center will remain same. The co-ordinate of ABCD are A(0, 0), B(0, 4), C(8, 4) and D(8, 0). Find co-ordinate of new square. Also derive the transformation matrix to convert this rectangle to square.*** *[2072 kartik]*

Solution:



The transformation matrix that transformations the rectangle ABCD whose center is (4, 2) is reduced to half of its size keeping the center same is,

P' = C.M. *P

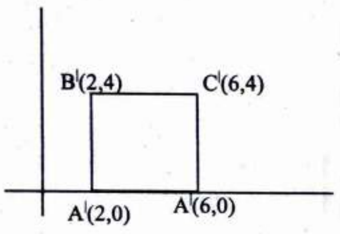
$$C.M. = \begin{bmatrix}1 & 0 & 4\\0 & 1 & 2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1/2 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -4\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}$$

$$C.M = \begin{bmatrix}1 & 0 & 4\\0 & 1 & 2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1/2 & 0 & -2\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1/2 & 0 & 2\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

Now to find new coordinate of new square,

A' = C.M*A

$$\begin{bmatrix}1/2 & 0 & 2\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}2\\0\\1\end{bmatrix}$$



B' = C.M*B

$$\begin{bmatrix}1/2 & 0 & 2\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\4\\1\end{bmatrix} = \begin{bmatrix}2\\4\\1\end{bmatrix}$$

C' = C.M. × C

$$\begin{bmatrix}1/2 & 0 & 2\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}8\\4\\1\end{bmatrix} = \begin{bmatrix}6\\4\\1\end{bmatrix}$$

D' = C.M. × D

$$= \begin{bmatrix}1/2 & 0 & 2\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}8\\0\\1\end{bmatrix} = \begin{bmatrix}6\\0\\1\end{bmatrix}$$

The coordinates of new square are (2,0), (2,4), (6,4), and (6,0)

Steps to get the transformation matrix to convert this rectangle to square are;

i. Translate by (-4, -2)

ii. Scale by S_x=1/2

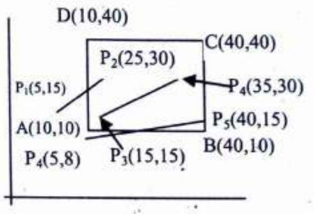
iii. Rotate by (4, 2)

$$C.M. = \begin{bmatrix}1 & 0 & 4\\0 & 1 & 2\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1/2 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -4\\0 & 1 & -2\\0 & 0 & 1\end{bmatrix}$$

9. ***Given a clipping window A(10, 10), B(40,10), C(40, 40) and D(10, 40). Using Cohen-Sutherland line clipping algorithm find region code of each end points of lines P_1P_2, P_3P_4 and P_5P_6 where co-ordinates are P_1(5,15), P_2(25, 30), P_3(15, 15), P_4(35, 30), P_5(5, 8) and P_6(40, 15). Also find clipped lines using above parameters. ?*** *[2071 Shrawan]*

Solution:

Step 1: Assign the region code.



For P_1 (5, 15)

5 < 10 true so bit 1= 1

5 > 40 False so bit 2 = 0

15 < 10 false so bit 3 = 0

15 > 40 false so bit 4 = 0

Region code for

P_1 (5,15) = 0 0 0 1

For P_2 (25,30)

25 < 10 false so bit 1= 0

25 > 40 False so bit 2= 0

30 < 10 false so bit 3= 0

30 > 40 false so bit 4= 0

Region code for P_2 (25,30)= 0 0 0 0

For P_3 (15,15)

15 < 10 false so bit 1= 0

15 > 40 False so bit 2= 0

15 < 10 false so bit 3= 0

15 > 40 false so bit 4= 0

Region code for P_3 (15,15)= 0 0 0 0

For P_4 (35,30)

35 < 10 false so bit 1= 0

35 > 40 False so bit 2= 0

30 < 10 false so bit 3= 0

30 > 40 false so bit 4= 0

Region code for P_4 (35,30)= 0 0 0 0

For P_5 (5,8)

5 < 10 true so bit 1 = 1

5 > 40 False so bit 2 = 0

8 < 10 true so bit 3 = 1

8 > 40 false so bit 4 = 0

Region code for P_5 (5,8)= 0 1 0 1

For P_6 (40,15)

40 < 10 false so bit 1= 0

40 > 40 False so bit 2= 0

15 < 10 false so bit 3= 0

15 > 40 false so bit 4= 0

Region code for P_6 (40,15)= 0 0 0 0

Now, for line P_1 P_2, P_1(5, 15), P_2(25,3 0)

Region code for P_1= 0001

Region code for P_2 = 0000

i. Both end point have not region code '0000', line is not completely inside.

ii. The logical AND of end points of the line P_1 P_2 is '0000', so line is not completely outside.

So now,

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{30-15}{25-5} = \frac{15}{20} = \frac{3}{4}$$

If bit1 is 1, line intersects with left boundary.

So,

$$y_i = y_1 + m(x_{min} - x_1)$$
$$= 15 + \frac{3}{4}(10-5)$$
$$= 18.75$$

y_i = 18.75 = 19

x_i = 10

Region code is,

10 < 10 false bit 1 =0

10 > 40 false bit 2 = 0

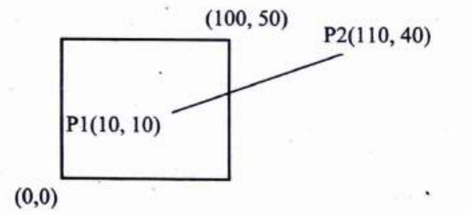
19 < 10 false bit 3= 0

19 > 40 false bit 4= 0

So, the required line is P_1' (10, 19) and P_2' (25,30)

10. ***Use Liang Barsky clipping method to clip a line starting from P1 (10, 10) and ending at P2(110, 40) against the window having its lower corner at (0, 0) and upper right corner at (100, 50)***

Solution:



K	p_k	q_k	r_k
1	$-\Delta x$ = -(110-10) = -100 i. e., p_k<0	x_1-x_{wmin} = 10-0 = 10	r1 = 10/-(100) = -1/10 = candidate for u_1

K	p_k	q_k	r_k
2	Δx $= (110-10)$ $= 100$ i. e., $p_k>0$	$x_{wmax}-x_1$ $= 100-10$ $= 90$	$r_2 = 90/100$ $= -0.9$ = candidate for u_2
3	$-\Delta y$ $= -(40-10)$ $= -30$ i. e.$Pk<0$	y_1-y_{wmin} $= 10-0$ $= 10$	$r_3 = 10/-30$ $= -1/3$ = candidate for u_1
4	Δy $= (40-10)$ $= 30$ i.e. $Pk > 0$	$y_{wmax}-y_1$ $= 50-10$ $= 40$	$r_4 = 40/30$ $= 4/3$ = candidate for u_2

We take $u_1 = 0$ and $u_2 = 0.9$

Clipped line

$x_1' = 10 + 0 \times 100 = 10$

$y_1' = 10 + 0 \times 30 = 10$

$x_2' = x1 + u_2 \times 100 = 10 + 0.9 \times 100 = 100$

$y_2' = y1 + u_2 \times 100 = 10 + 0.9 \times 30 = 37$

11. ***State the condition of point clipping perform clipping operation for the following using Liang Barskey line clipping algorithm.*** *[2070 Chaitra]*

Solution:

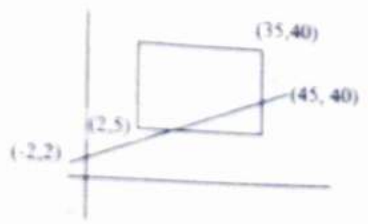
Clipping window: $(x_{min}, y_{min}) = (2,5)$

And $(x_{max}, y_{max}) = (35,50)$

Line $(x_1, y_1) = (-2,2)$ and $(x_2, y_2) = (45,40)$

$\Delta x = x_2 - x_1 = 45 - (-2) = 47$

$\Delta y = y_2 - y_1 = 40 - 2 = 38$



k	p_k	q_k	$R_k = \frac{q_k}{p_k}$
0	$\Delta x = -47, p_k<0$	$x_1 - x_{min} = -4$	$0.0851(u_1)$
1	$\Delta x = 47$	$x_{max} - x_1 = 37$	$0.787(u_2)$
2	$-\Delta y = -38$	$y_1 - y_{min} = -3$	$0.0789(u_1)$
3	$\Delta y = 38$	$y_{max} - y_1 = 48$	$1.263(u_2)$

$u_1 = \max(0, r_k)$

$= \max(0, 0.0851, 0.0789)$

$= 0.0851$

$u_2 = \min(1, r_k) = \min(1, 0.787, 1.263) = 0.787$

$x_1' = x_1 + u_1 \Delta x = -2 \times 0.051 \times 47 = 1.997 \approx 2$

$y_1' = y_1 + u_1 \Delta y = -2 + 0.0851 \times 38 = 5$

$x_2' = x_1 + u_2 \Delta x = -2 + 0.787 \times 47 = 35$

$y_2' = y_1 + u_2 \Delta y = 2 + 0.787 \times 38 = 32$

Required points are (2, 5) and (35, 32)

12. ***Write down the condition for point clipping. Find the clipped region in window of diagonal vertex (10, 10) and (100, 100) for line P_1(5, 120) and P_2(80, 7) using Liang-Barsky line clipping method.*** *[2072 kartik]*

Solution:

Assuming that the clip window is a rectangle in standard position, we save a point P(x, y) for display, if the following inequalities are satisfied.

K	p_k	q_k	r_k
2	Δx $= (110-10)$ $= 100$ i. e., $p_k>0$	$x_{wmax}-x_1$ $= 100-10$ $= 90$	$r_2 = 90/100$ $= -0.9$ $=$ candidate for u_2
3	$-\Delta y$ $= -(40-10)$ $= -30$ i. e.Pk<0	y_1-y_{wmin} $= 10-0$ $= 10$	$r_3 = 10/-30$ $= -1/3$ $=$ candidate for u_1
4	Δy $= (40-10)$ $= 30$ i.e. Pk > 0	$y_{wmax}-y_1$ $= 50-10$ $= 40$	$r_4 = 40/30$ $= 4/3$ $=$ candidate for u_2

We take $u_1 = 0$ and $u_2 = 0.9$

Clipped line :

$x_1' = 10 + 0 \times 100 = 10$

$y_1' = 10 + 0 \times 30 = 10$

$x_2' = x1 + u_2 \times 100 = 10 + 0.9 \times 100 = 100$

$y_2' = y1 + u_2 \times 100 = 10 + 0.9 \times 30 = 37$

11. ***State the condition of point clipping perform clipping operation for the following using Liang Barsky line clipping algorithm.*** *[2070 Chaitra]*

Solution:

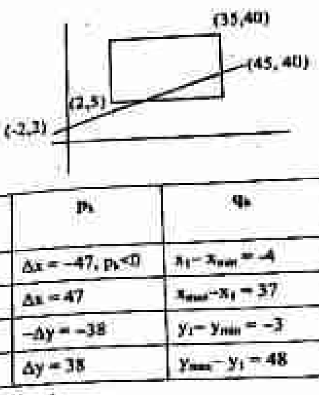
Clipping window: $(x_{min}, y_{min}) = (2,5)$

And $(x_{max}, y_{max}) = (35,50)$

Line $(x_1, y_1) = (-2,2)$ and $(x_2, y_2) = (45,40)$

$\Delta x = x_2 - x_1 = 45 - (-2) = 47$

$\Delta y = y_2 - y_1 = 40 - 2 = 38$



k	p_k	q_k	$R_k = \frac{q_k}{p_k}$
0	$\Delta x = -47$, $p_k<0$	$x_1 - x_{min} = -4$	0.0851(u_1)
1	$\Delta x = 47$	$x_{max} - x_1 = 37$	0.787(u_2)
2	$-\Delta y = -38$	$y_1 - y_{min} = -3$	0.0789(u_1)
3	$\Delta y = 38$	$y_{max} - y_1 = 48$	1.263(u_2)

$u_1 = \max(0, r_k)$

$= \max(0, 0.0851, 0.0789)$

$= 0.0851$

$u_2 = \min(1, r_k) = \min(1, 0.787, 1.263) = 0.787$

$x_1' = x_1 + u_1\Delta x = -2 \times 0.051 \times 47 = 1.997 \approx 2$

$y_1' = y_1 + u_1\Delta y = -2 + 0.0851 \times 38 = 5$

$x_2' = x_1 + u_2\Delta x = -2 + 0.787 \times 47 = 35$

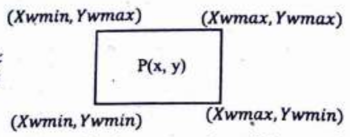
$y_2' = y_1 + u_2\Delta y = 2 + 0.787 \times 38 = 32$

Required points are (2,5) and (35, 32)

12. ***Write down the condition for point clipping. Find the clipped region in window of diagonal vertex (10, 10) and (100, 100) for line P_1(5, 120) and P_2(80, 7) using Liang-Barsky line clipping method.*** *[2072 kartik]*

Solution:

Assuming that the clip window is a rectangle in standard position, we save a point P(x, y) for display, if the following inequalities are satisfied.



$Xwmin \le X \le Xwmax$
$Ywmin \le Y \le Ywmax$

- The edges of the clip window can be either we window boundary or viewport boundaries.
- If any one of these four inequalities is not satisfied, the point is clipped i.e., not saved for display.

Here,

$x_{wmin} = 10$

$y_{wmin} = 10$

$x_{wmax} = 100$

$y_{wmax} = 100$

$(x_1, y_1) = (5, 120)$

$(x_2, y_2) = (80, 7)$

Now, we have to find out the value of u_1 and u_2 by calculating p_k, q_k, r_k from k = 1 to 4.

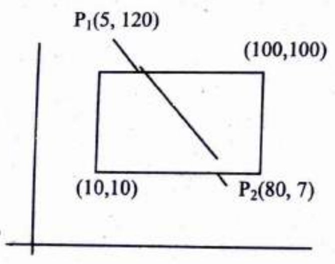
Then, the clipped line will be

$x_1' = x_1 + u_1\Delta x$

$y_1' = y_1 + u_1\Delta y$

$x_2' = x_1 + u_2\Delta x$

$y_2' = y_1 + u_2\Delta y$



K	p_k	q_k	$r_k = \frac{q_k}{p_k}$
1	$-\Delta x = -(80-5) = -75$ i.e., $p_k<0$	$x_1 - x_{wmin}$ $(5-10) = -5$	$r_1 = \frac{-5}{-75} = \frac{1}{15}$ (u_1)
2	Δx $= -(80-5)$ $= 75$ i.e., $p_k>0$	$x_{wmax} - x_1$ $= 100 - 5$ $= 95$	$r_2 = \frac{95}{75}$ (u_2)
3	Δy $= -(7-120)$ $= 113$ i.e., $p_k>0$	$y_1 - y_{wmm}$ $= (120-10)$ $= 110$	$r_3 = \frac{110}{113}$ (u_2)
	Δy $= -113$ i.e., $p_k<0$	$y_{wmax} - y_1$ $= 100 - 120$ $= -20$	$r_4 = \frac{-20}{-113} = \frac{20}{113}$ (u_1)

Now,

$u_1 = \max(0, r_k)$

$= \max(0, \frac{1}{15}, \frac{20}{113}) = \frac{20}{113}$

$u_2 = \min(1, r_k)$

$= \min(1, \frac{95}{75}, \frac{110}{113}) = \frac{110}{113}$

$x_1' = x_1 + u_1\Delta x$

$= 5 + \frac{20}{113} \times 75 = 18.27$

$y_1' = y_1 + u_1\Delta y$

$= 120 + \frac{20}{113} \times (-113) = 100$

$x_2' = x_1 + u_2\Delta x$

$= 5 + \frac{110}{113} \times 75 = 78$

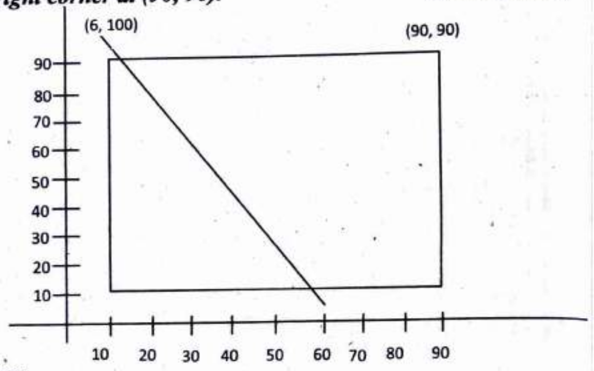
$y_2' = y_1 + u_2\Delta y$

$= 120 + \frac{110}{113} \times (-113) = 10$

$P_1'(x_1', y_1') = P_1'(18,27,100)$

$P_2'(x_2', y_2') = P_2'(78, 10)$

13. ***Use Liang Barsky line clipping algorithm to clip a line starting from (6, 100) and ending at (60, 5) against the window having its lower left corner at (10, 10) and upper right corner at (90, 90).*** *[2075 Ashwin]*



Solution:

Clipping window $(x_{min}, y_{min}) = (10, 10)$

$(x_{max}, y_{max}) = (90, 90)$

$(x_1, y_1) = (6, 100)$ and $(x_2, y_2) = (60, 5)$

$\Delta x = x_2 - x_1 = 60 - 6 = 54$

$\Delta y = y_2 - y_1 = 5 - 100 = -95$

K	p_k	q_k	$r_k = q_k/p_k$
0	$-\Delta x = -54 (p_k < 0)$	$x_1 - x_{min} = 6 - 10 = -4$	$0.074(u_1)$
1	$\Delta x = 54$	$x_{max} - x_1 = 90 - 6 = 84$	$1.55(u_2)$
2	$-\Delta y = 95$	$y_1 - y_{min} = 100 - 10 = 90$	$0.947(u_2)$
3	$\Delta y = -95$	$y_{max} - y_1 = 90 - 100 = -10$	$0.105(u_1)$

$u_1 = \max(0, r_k) = 0.105$

$u_2 = \min(0, r_k) = 0.947$

$x_1' = x_1 + u_1\Delta x = 6 + 0.105 \times 54 = 11.67 \approx 12$

$y_1' = y_1 + u_1\Delta y = 100 + 0.105 \times (-94) = 90.025 \approx 90$

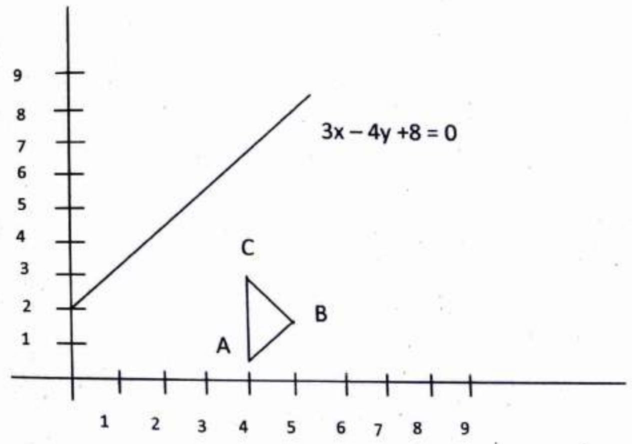
$x_2' = x_1 + u_2\Delta x = 6 + 0.947 \times 54 = 57.118 \approx 57$

$y_2' = y_1 + u_2\Delta y = 100 + 0.947 \times (-94) = 10.035 \approx 10$

Required points are (12, 90) and (57, 10)

14. ***Reflect the triangle ABC about the line 3x – 4y + 8 = 0 the position vector of coordinate ABC as A(4, 1), B(5, 2) and C(4, 3).*** *[2075 Ashwin]*

Solution:



The arbitrary line about which the triangle ABC has to be reflected is $3x - 4y + 8 = 0$

i.e., $y = \frac{3}{4}x + 2$

$m = \frac{3}{4}$

$c = 2$

$$\theta = \tan^{-1}(m) = \tan^{-1}\frac{3}{4} = 36.8698^{\circ}$$

$$C.M. = T^{-1}R^{-1}_{\theta}R_{fx}R_{\theta}T$$

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{\theta} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(36.87) & \sin(36.87) & 0 \\ -\sin(36.87) & \cos(36.87) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{fx} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} \cos(-36.87) & \sin(-36.87) & 0 \\ -\sin(-36.87) & \cos(-36.87) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C.M. = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.8 & -0.6 & 0 \\ 0.6 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

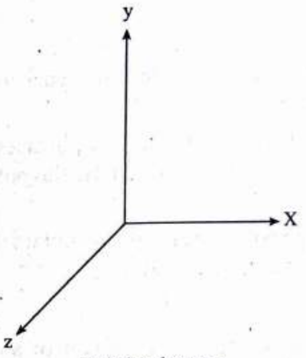
$$\begin{bmatrix} 0.8 & 0.6 & 0 \\ -0.6 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = C.M. \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

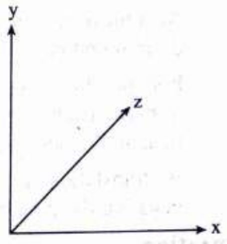
Three-Dimensional Transformations

Three Dimensional Graphics

Three dimensional graphics uses three dimensional representations of geometric data. 3D adds the depth (z) dimension in length (x) and breadth (y) dimension in 2D. 3D is more complex than 2D because in 3D relatively more co-ordinate points are needed, object boundaries can be constructed with various combination of plane and curved surfaces, viewing direction, position in space, orientation, projection consideration, visible surface detections etc. matters in displaying the graphics.



Right hand system



Left hand system

4.1 Three-Dimensional Transformations

- Translation
- Rotation
- Scaling
- Reflection
- Shear

Matrix used in 3d transformation is of order 4×4 homogeneous co-ordinate for 3D is (x, y, z, 1)

Translation/Shifting

In a 3D homogeneous co-ordinate representation, a point is translated form position P(x,y,z) to position P'(x', y', z') with the matrix operation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

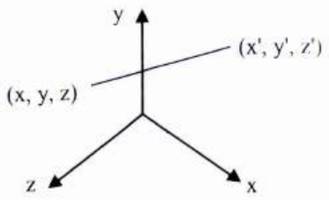


Fig 4.1: Translation of a point

OR

P' = T.P

$x' = x + t_x$

$y' = y + t_y$

$z' = z + t_z$

- An object is translated in 3D by transforming each of the defining points of the object.
- For an object represented as set of polygon surfaces, we translate each vertex of each surface and redraw the polygon facts in the new position
- A translation in the opposite direction in obtained by negative the translation distance t_x, t_y, and t_z

Rotation

- We must define an axis of rotation and amount of angular rotation to generate rotation transformation
- Unlike 2D application, where all transformations are carried out in the xy-plane, a 3D rotation can be specified around any line in space.

Co-ordinate Axes Rotations:

i) **Rotation about z- axis or (z axis rotation)**

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

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

$z' = z$

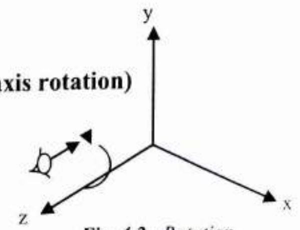


Fig 4.2 : Rotation about z-axis

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

i.e. $P' = R_z(\theta).P$

ii) **Rotation about x –axis**

$y' = y\cos\theta - z\sin\theta$

$z' = y\sin\theta + z\cos\theta$

$x' = x$

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

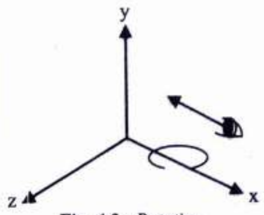


Fig 4.3 : Rotation about x-axis

$P' = R_x(\theta).P$

iii) **Rotation about y –axis**

$z' = z\cos\theta - x\sin\theta$

$x' = z\sin\theta + x\cos\theta$

$y' = y$

In matrix form

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

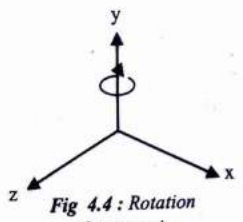


Fig 4.4 : Rotation about y-axis

$P' = R_y(\theta).P$

An inverse rotation matrix is formed by replacing the rotation angle θ by -θ

General 3D rotations

i. **About an axis that is parallel to one of the co-ordinate axes.**

a. Translate the object so that the rotation axis coincides with parallel co-ordinate axes.

b. Perform the specified rotation about that axis.

c. Translate the object so that the window axis is moved back to its original position.

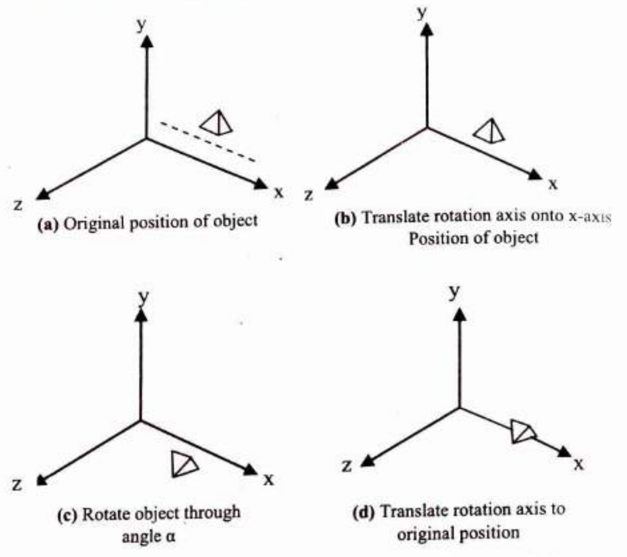

y
x
z
(a) Original position of object
(b) Translate rotation axis onto x-axis Position of object
(c) Rotate object through angle α
(d) Translate rotation axis to original position

Figure 4.5: Rotation about an axis that is parallel to one of the co-ordinate axes.

ii) **Rotation about an arbitrary axis**

a. Translate the object so that the rotation axis through the origin

b. Rotate the object so that the axis of rotation coincides with one of the co-ordinate axis.

c. Perform the specified rotation about that co-ordinate axis.

d. Apply inverse rotations to bring the rotation axis back to its original orientation.

e. Apply the inverse translation to bring the rotation axis back to its original position.

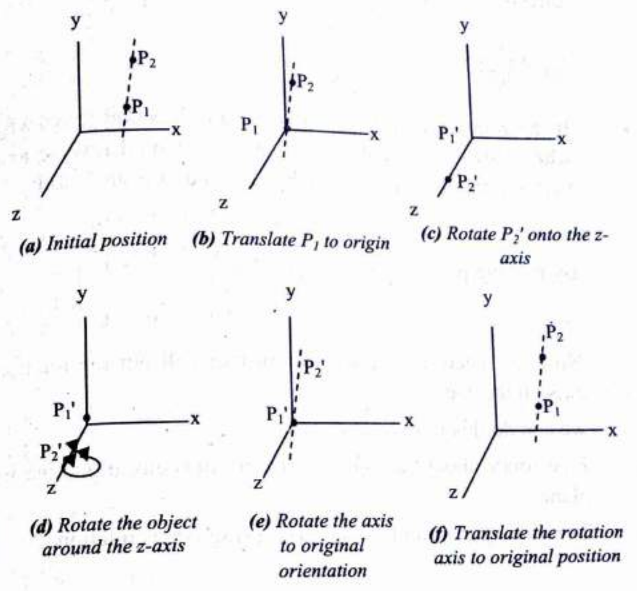

y
x
z
P_1
P_2
P_1'
P_2'
(a) Initial position
(b) Translate P_1 to origin
(c) Rotate P_2' onto the z-axis
(d) Rotate the object around the z-axis
(e) Rotate the axis to original orientation
(f) Translate the rotation axis to original position

Figure 4.6: Rotation about an arbitrary axis

- A rotation axis can be defined with two co-ordinate positions or with one coordinate point and direction angles (or direction cosines) between the rotation axis and two of the co-ordinate axes.
- Assume that the rotation axis is defined by two points and that the direction of rotation is to be counterclockwise when looking along the axis from P_2 to P_1.
- An axis vector is then defined by the two points as

$V = P_2 - P_1 = (x_2-x_1, y_2-y_1, z_2-z_1)$

An unit vector u is then defined along the rotation axis as

$$\frac{\mathbf{u}}{|\mathbf{v}|} = (a, b, c)$$

where the components a, b and c of unit vector u are the direction cosines for the rotation axis $a = \frac{x_2 - x_1}{|V|}$

$$b = \frac{y_2 - y_1}{|V|}, \qquad c = \frac{z_2 - z_1}{|V|}$$

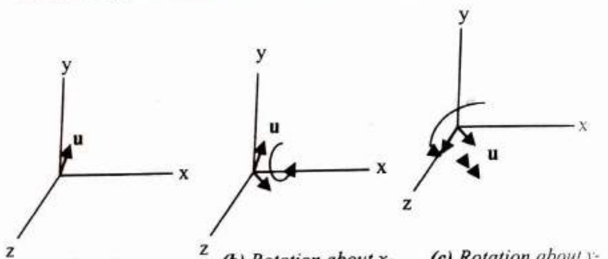
- If the rotation is to be in the opposite direction (clockwise when viewing from P_2 to P_1), then we would reverse axis vector v and unit vector u so that they point from P_2 to P_1.

By moving point P_1 to the origin. $T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Now, we need the transformation that will put the rotation axis on the z axis.

we can do this in two steps.

- First rotate about the x-axis to transform vector u into the xz plane.
- Then swing u around to the z axis using y-axis rotation.



(a) Translation *(b) Rotation about x-axis* *(c) Rotation about y-axis*

***Figure 4.7** Unit vector u is translation and rotation*

Since rotation calculations involve sine and cosine functions, we can use standard vector operations to obtain elements of the two rotation matrices.

$$\mathbf{V}_1.\mathbf{V}_2 = |\mathbf{V}_1||\mathbf{V}_2|\cos\theta \qquad 0 \le \theta \le \pi$$

$$\mathbf{V}_1 \times \mathbf{V}_2 = \mathbf{u}|\mathbf{V}_1||\mathbf{V}_2|\sin\theta \qquad 0 \le \theta \le \pi$$

$$\mathbf{V}_1.\mathbf{V}_2 = V_{1x}V_{2x} + V_{1y}V_{2y} + V_{1z}V_{2z}$$

$$\mathbf{V}_1 \times \mathbf{V}_2 = \begin{vmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \\ V_{1x} & V_{1y} & V_{1z} \\ V_{2x} & V_{2y} & V_{2z} \end{vmatrix}$$

- We establish the transformation matrix for rotation around the x axis by determining the values for the sine and cosine of the rotation angle necessary to get u into the xz plane.

  This rotation angle is the angle between the projection by u in the yz plane and the positive z axis.

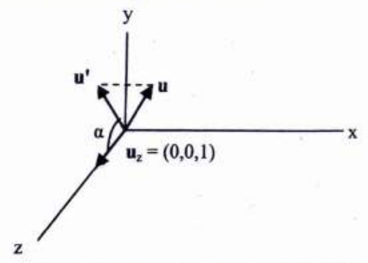


Figure 4.8** Rotation of **u** around the x axis into the xz plane is accomplished by rotating **u'

- If we designate the projection of u in yz plane as vector **u'** = (0, b, c) then cosine of the rotation angle α can be determined from the dot product of u' and the unit vector uz along z axis

$$\cos\alpha = \frac{\mathbf{u}^{|}.\mathbf{u}_z}{|\mathbf{u}^{|}||\mathbf{u}_z|} = \frac{c}{d} \text{ where } d = \sqrt{b^2 + c^2}$$

Similarly, $\mathbf{u}'X\mathbf{u}_z = \mathbf{u}_x|\mathbf{u}^{|}||\mathbf{u}_z|\sin\alpha$ and the Cartesian form for the cross product given us $u' \times u_z = u_x.b$

$$|\mathbf{u}_z| = 1$$

$$|\mathbf{u}'| = d$$

$d \sin\alpha = b$

$$\sin\alpha = \frac{b}{d}$$

Now that we have determined the values of sin α and cos α

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix rotates unit vector u about the x-axis into xz plane.

Now, swing the unit vector in xz plane, counterclockwise around the y axis onto the positive z axis

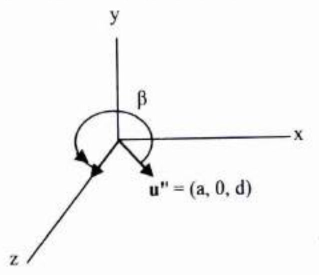


*Figure 4.9 Rotation of unit vector **u''** about y axis*

- The vector labeled u'' has the value a for its x component since rotation about the x-axis leaves the x-component unchanged. Its z component is d (the magnitude of **u'**). Because vector u' has been rotated onto the z axis. And the y component of u'' is 0 because now it lies in xz plane

$$\cos\beta = \frac{\mathbf{u''}.\mathbf{u}_z}{|\mathbf{u'}||\mathbf{u}_z|} = d$$

Since $|\mathbf{u}_z| = |\mathbf{u'}| = 1$, comparing the co-ordinate independent form of the cross product.

$\mathbf{u''} \times \mathbf{u}_z = \mathbf{u}_y |\mathbf{u'}||\mathbf{u}_z| \sin\beta$

with the Cartesian form

$\mathbf{u''} \times \mathbf{u}_z = \mathbf{u}_y.(-a)$

We find that $\sin\beta = -a$

$$R_f(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The specified rotation angle θ can be applied as a rotation about the z axis.

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

- To complete the required rotation about the given axis, we need to transform the rotation axis back to its original position.

$$R(\theta) = T^{-1}.R_x^{-1}(\alpha)R_y^{-1}(\beta)R_z(\theta).R_y(\beta).R_x(\alpha).T$$

- A somewhat quicker but perhaps less intuitive, method for obtaining the composite rotation matrix $R_y(\beta).R_x(\alpha)$ is to take advantage of the form of the composite matrix for any sequence of three dimensional rotations.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The upper left 3×3 sub matrix of this matrix is orthogonal. This means row (or columns) of this sub-matrix form a set of orthogonal unit vectors that are rotated by matrix R onto the x, y, and z axes respectively.

$$R.\begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix},\ R.\begin{bmatrix} r_{21} \\ r_{12} \\ r_{23} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix},\ R.\begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

- We can consider a local co-ordinate system defined by the rotation axis and simply form a matrix whose columns are the local unit co-ordinate vectors. Assume rotation axis is not parallel to any co-ordinate axis.

  $\mathbf{u}_z{}' = \mathbf{u},\ \mathbf{u}'_y = \dfrac{\mathbf{u} \times \mathbf{u}_x}{|\mathbf{u} \times \mathbf{u}_x|},\ \mathbf{u}'_x = \mathbf{u}'_y\,\mathbf{u}'_z$

  And if we express element of local unit vectors for rotation axis as

  $\mathbf{u}'_x = (u'_{x1}, u'_{x2}, u'_{x3}),\ \mathbf{u}'_y = (u'_{y1}, u'_{y2}, u'_{y3})$

  $\mathbf{u}'_z = (u'_{z1}, u'_{z2}, u'_{z3})$ then the required composite matrix equal to the product $R_y(\beta).R_x(\alpha)$ is

$$R = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1. ***Perform rotation of a line (10, 10, 10), (20, 20,15) about Y-axis in clock wise direction by 90 degree. Explain about vector display.?*** *[2071 Shrawan]*

Solution:

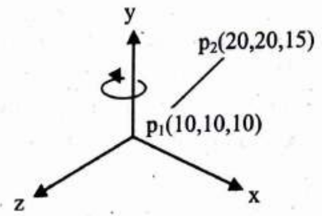
Rotation about y-axis in clock wise direction

$z' = z\cos\theta + x\sin\theta$

$x' = -z\sin\theta + x\cos\theta$

$y' = y$

$\theta = 90^0$



In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x1' \\ y1' \\ z1' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 90 & 0 & -\sin 90 & 0 \\ 0 & 1 & 0 & 0 \\ \sin 90 & 0 & \cos 90 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} -10 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

so, $(x_1', y_1', z_1') = (-10, 10, 10)$

$$\begin{bmatrix} x2' \\ y2' \\ z2' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 20 \\ 20 \\ 15 \\ 1 \end{bmatrix} = \begin{bmatrix} -15 \\ 20 \\ 20 \\ 1 \end{bmatrix}$$

so, $(x_2', y_2', z_2') = (-15, 20, 20)$

2. ***Write rotation matrix in clockwise direction with respect to x-axis, y-axis and z-axis, Reflect the object (0; 0, 0), (2, 3, 0) and (5, 0, 4) about the plane y = 4.*** *[2071 Chaitra]*

Solution:

(Refer the Theory)

Steps to reflect the object.

i) Translate the plane so that the plane coincides with the xz plane

ii) Perform the xz reflection

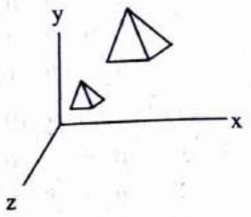
iii) Translate the object so that reflection plane is moved back to its original position.

$$C.M. = \begin{bmatrix}1&0&0&0\\0&1&0&4\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&-1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&-4\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$P' = C.M. \times p$

$$\begin{bmatrix}x_1'\\y_1'\\z_1'\\1\end{bmatrix} = C.M.\times\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

$$\begin{bmatrix}x_2'\\y_2'\\z_2'\\1\end{bmatrix} = C.M.\times\begin{bmatrix}2\\3\\0\\1\end{bmatrix}$$



$$\begin{bmatrix}x_3'\\y_3'\\z_3'\\1\end{bmatrix} = C.M.\times\begin{bmatrix}5\\0\\4\\1\end{bmatrix}$$

Scaling

$x' = x.s_x$

$y' = y.s_y$

$z' = z.s_z$

In matrix form,

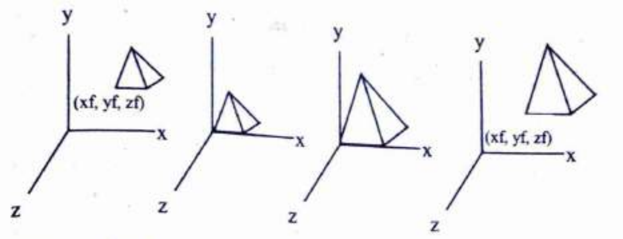
$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix} = \begin{bmatrix}1&0&0&0\\0&\cos\theta&-\sin\theta&0\\0&\sin\theta&\cos\theta&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

Scaling about fixed point

i. Translate the fixed point to the origin.

ii. Scale the object relation to the co-ordinate origin.

iii. Translate the fixed point back to its original.



$C.M. = T^{-1}.S.T.$

$= T(x_f, y_f, z_f).\ S(s_x, s_y, s_z).\ T(-xf, -yf, -zf)$

$$= \begin{bmatrix}s_x&0&0&(1-s_x)x_f\\0&s_y&0&(1-s_y)y_f\\0&0&s_z&(1-s_z)z_f\\0&0&0&1\end{bmatrix}$$

Reflection

- A 3D reflection can be performed relative to a selected 'reflection axis' or with respect to a selected 'reflected plane'.
- Reflection relative to a given axis is equivalent to 180° rotations about that axis.
- Reflection with respect to a plane is equivalent to 180° rotations in three dimensional space.
- When the reflection plane is a co-ordinate plane (either xy, xz or yz) we can think of the transformation as a conversion between left handed and right handed systems.

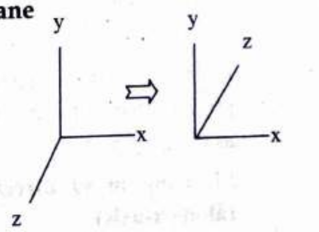
  For example,

  If an object is reflected about xy plane, it simply changes the sign of z values keeping the sign of x and y values same. So

Reflection about z-axis or xy plane

$P' = Rf_{(z)}.P$

$$\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix} = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&-1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$



Reflection about x-axis or yz plane

$P' = R_{f(x)}.P$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Reflection about y-axis or xz plane

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Shearing

- Shearing transformation are used to modify shape of the object.
- In space, one can push in z co-ordinate axis direction, keeping the third axis fixed.

Shearing in x and y direction keeping z co-ordinate same (along z-axis)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$x' = x + sh_x.z$

$y' = x + sh_y.z$

$z' = z$

It alters the x and y co-ordinates values by an amount that is proportional to the z axis while leaving z co-ordinate. same as $p' = sh_{xy}.p$

Shearing in yz direction keeping x co-ordinate same (along x-axis)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_y & 1 & 0 & 0 \\ sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$P' = sh_{yz}.P$ $\quad$ $x' = x$

$y' = y + sh_y . x$ $\quad$ $z' = z + sh_z. x$

Shearing in xz direction keeping y co-ordinate same (along y-axis)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$p' = sh_{xz}.p$ $\quad$ $y' = y$

$x' = x + sh_x . y$ $\quad$ $z' = z + sh_z. y$

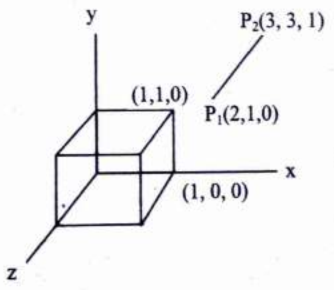
1. ***Find the new coordinates of a unit cube 90° rotated about an axis defined by its end points A(2, 1, 0) and B(3, 3, 1).***

Solution:

$\mathbf{v} = P_2 - P_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$

$a = \frac{x_2-x_1}{|v|} = \frac{1}{\sqrt{6}}$ $\quad$ $b = \frac{y_2-y_1}{|v|} = \frac{2}{\sqrt{6}}$

$c = \frac{z_2-z_1}{|v|} = \frac{1}{\sqrt{6}}$ $\quad$ $d = \sqrt{b^2+c^2} = \sqrt{\frac{5}{6}}$



$R(\theta) = T^{-1}Rx^{-1}(\alpha)Ry^{-1}(\beta).Rz(\theta).Ry(\beta).Rx(\alpha).T$

$$= \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 1 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos 90 & -\sin 90 & 0 & 0 \\ \sin 90 & \cos 90 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 1 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D viewing

In 2D graphics application, viewing operations transfer position from the world coordinate plane to pixel position in the plane of the output device.

- But in 3D graphics application, we have to consider spatial position (i.e., an object can be viewed from the front, from above or from the back) or we could generate a view of what we would see if we were standing in the middle of a group of objects or inside a single object, such as buildings.
- Additionally, 3D descriptions of object must be projected onto the flat viewing surface of the output device.
- And the clipping boundaries now enclose a volume of space, whose shape depends on the type of projection we select.

Viewing pipeline

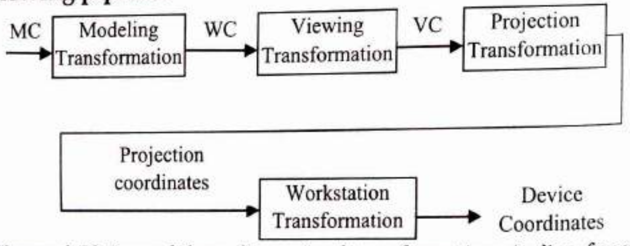


***Figure 4.10** General three-dimensional transformation pipeline, from modeling co-ordinate to find device co-ordinates*

3D viewing pipeline describes the conversion of 3D object into 2D projection or mapping by using some processes.

- The steps for computer generation of a view of a three dimensional scene are analogous to the process involved in taking a photograph.
- To take a snapshot, we first need to position the camera at a particular point in space, then need to decide on the camera orientation. (i.e., which way do we point the camera and how should we rotate it around the line of sight to set the up direction for the picture). Finally, when we snap the shutter, the scene is cropped to the size of the shutter, the scene is cropped to the size of the 'window' (aperture) of the camera and light from the visible surface is projected.

Each model or object has its own dimension and coordinate system. It is called modeling coordinate. Modeling transformation is to take all the objects in a single scene by using transformation such translation, rotation etc. To set object is called modeling translation.

After modeling translation, the objects come to a scene or a coordinate system is called world coordinate.

To set the camera on some position, angle or orientation is called viewing transformation. From viewing transformation we get viewing coordinate.

Projection transformation is to adjust focus, zoom in, zoom out etc. Projection transformation creates projection coordinates.

Workstation or viewport translation is just like to click the button to save the image in the device.

Once the scene has been modeled, world coordinate positions are converted to viewing co-ordinate.

- The VC system is used in graphics system as a reference for specifying the observer viewing position and the position of the projection plane analogous to camera film plane.

- Projection operations are performed to convert VC description of a scene to co-ordinate positions on the projection plane.
- The projection plane is then mapped to output device.

Viewing co-ordinates:

Views of a scene can be generated by given spatial position (i.e., various distances), angle (i.e., angle with z_v axis), orientation, aperture (i.e., 'window') size of the camera.

Generating a view of an object in three dimensions is similar to photographing the object

Specifying the view plane

- We choose a particular view for a scene by first establishing the viewing co-ordinate system also called view reference co-ordinate system
- A view plane or projection plane is then set up perpendicular to the viewing z_v axis.
- WC positions in the scene are transformed to VC then VC are projected onto the view plane.

Establishing the VC reference frame

- First, pick a WC position called view reference point, the origin of our viewing co-ordinate system.
- The view reference point is often chosen to be close to or on the surface of some object in a scene. But it may be center of object or group object.
- Next, the positive direction for the viewing z_v axis and the orientation of the view plane is selected by specifying the view plane normal vector **N**
- We choose a WC position and this point establishes the direction for **N** relative either to the world origin or to the viewing co-ordinate origin. Establish direction of N using the selected co-ordinate position as a look at point relative to view reference point (view coordinate origin).

- Finally, we choose the up direction for the view by specifying a vector **V**, called view-up vector.
- This vector is used to establish the true direction for the yv axis.

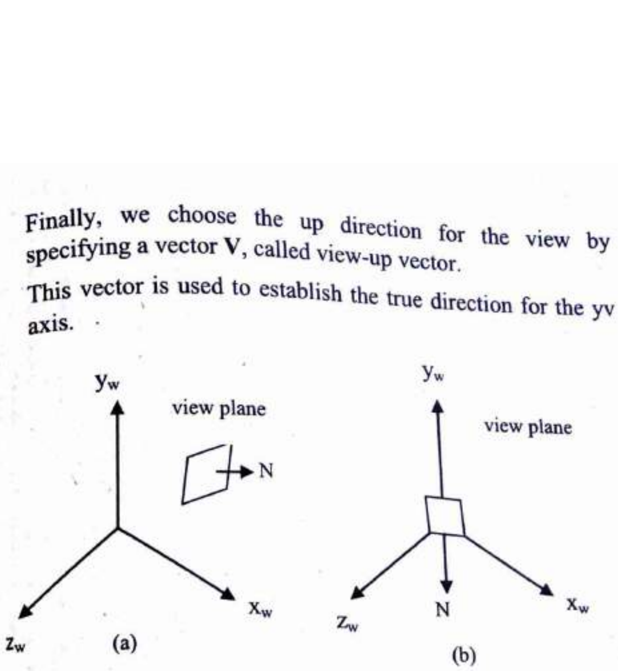


***Figure 4.10:** Orientation of the view plane for specified normal vector co-ordinates relative to the world origin position (1,0,0) orients the view plane as in (a) and (1,0,1) gives the orientation in (b)*

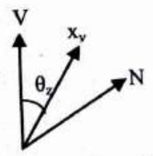


***Fig. 4.11:** specifying the view-up vector with a twist angle θt*

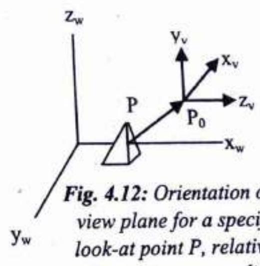


***Fig. 4.12:** Orientation of the view plane for a specified look-at point P, relative to the viewing - co-ordinate origin p_0*

- Vector **V** can be defined as a world co-ordinates origin P_0.
- Vector **V** can be defined as a world co-ordinate vector or in some packages it is specified with twist angle θ_t about the z_v axis.
- For general orientation of the normal vector, it can be difficult (or time consuming) to determine the direction for **V** that is precisely perpendicular to **N**.

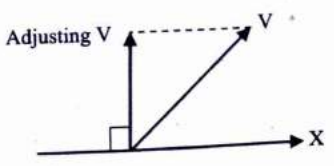


Figure 4.13:** Adjusting the input position of the view-up vector **V** to a position perpendicular to the normal vector **N

- Viewing procedures typically adjust the user-defined orientation of vector **V**,

  So that v is projected into a plane that is perpendicular to the normal vector.
- We choose the view up vector **V** to be in any convenient direction, as long as it is not parallel to **N**
- Using vector **N** and **V**, the graphics package can compute a third vector **U** perpendicular to both **N** and **V**. to define the direction for the $\mathbf{X}_v$ axis.
- Then the direction of **V** can be adjusted so that it is perpendicular to both **N** and **V** to establish the viewing y_v direction
- In transformation from world to viewing co-ordinates, these computations are conveniently carried out with unit axis vector.

View plane

- The window is defined in this plane.
- The origin of this plane which defines the position of the eye or camera is called the view reference point

  $\mathbf{e} = (e_x, e_y, e_z)$
- A unit vector to this plane is the view plane normal N.
- Another vector called the view up vector. V_{up} is a unit vector perpendicular to N.

View coordinate system

- Usually left-handed system called the uvn system

- **V**, the y-axis of the view co-ordinates system is perpendicular projection $\mathbf{V}_{up}$ of on the view plane.
- u, the x axis of the view co-ordinate is is orthogonal to V and N i.e., $\mathbf{U} = \mathbf{V} \times \mathbf{N}$
- Positive u and v are to the right and up-direction from eye's point of view.
- **N** is the z-axis of the view co-ordinate. It increases in positive direction with depth of a point from the eye.

Viewing co-ordinate Parameters

We call the viewing co-ordinate frame **uvn**, where **u**, **v** and **n** are three orthogonal vectors.

Let, P_0 be the view co-ordinate origin.

- P_{ref} be the look at point in the scene.
- **N** be the vector from p_{ref} to P_0
- Then n is the unit vector in the direction of **N**.
- Let **v** be the unit vector in the view up direction **N**.
- Vector **u** is perpendicular to **u** and **n**, where $U = \mathbf{V} \times \mathbf{N}$

  Usually, the user specifies P_0 and p_{ref}, and the view up vector **v**
- An eye defined within this system.

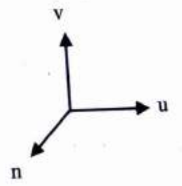
Usually, user doesn't give precise **V** exactly perpendicular to **N**.

Therefore, we use the following method to find u, v and n

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|}$$

$$\mathbf{u} = \frac{\mathbf{V x n}}{|\mathbf{V}|}$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u}$$



Let $\mathbf{a} = (a_x, a_y, a_z)$ be look-at-point

For perspective views, the view plane normal as a unit vector from eye to a look-at point is given by

$$N = \frac{1}{|a-e|}(a_x - e_x, a_y - e_y, a_z - e_z)$$

$$|a-e| = \sqrt{(a_x - e_x)^2 + (a_y - e_y)^2 + (a_z - e_z)^2}$$

The view up vector is the tilt(rotation) of the head or camera.

For parallel views it is convenient to think of the view plane normal as determining the direction of projection.

Transformation from world to viewing coordinates

Suppose that the viewing co-ordinates are specified in world co-ordinates. We need to transform each vertex specified in world co-ordinates to view co-ordinates.

1. Translate viewing co-ordinate origin to world co-ordinates origin.
2. Apply rotation to align u, v and n with the world x, y and z axes.

Transformation matrices:

Translation

If the view reference point is specified at world position (x_0, y_0, z_0) this point is translated to world origin with the matrix transformation.

$$\begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

The composite rotation matrix for viewing transformation is then

$$R = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Another approach,

- An object in world co-ordinate space, whose vertices are (x, y, z) can be expressed in term of view co-ordinates (u,v,n)
- Translate the view reference point e to the origin.
- Rotate about the world co-ordinates of axis to bring the view co-ordinate axis into the yz plane of world co-ordinate
- Rotate about the world co-ordinates z axis to align the axis with the y.
- Reflect relative to xy - plane, revising sign of each z co-ordinate to change into a left handed coordinate system

The viewing transformation are $V = T.R_y.\ R_x.\ R_z\ R_f$

Projection

Projection can be defined as a mapping of point p(x, y, z) onto its image P'(x', y', z') in the projection plane or view plane, which constitutes the display surface

The mapping is determined by a projection line called the projector that passes through p and intersects the view plane. The intersection point is P'.

Two basic methods

i. Parallel projection

ii. Perspective projection

Parallel projections:

Co-ordinate positions are transformed to the view plane along parallel lines.

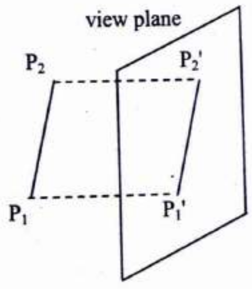


Figure 4.14: *Parallel projection of an object to the view plane*

Parallel projection preserves relative proportions of objects, and this method is used in drafting to produce scale drawing of three dimensional objects.

Accurate views of the various sides of an object are obtained with a parallel projection. But this does not give us a realistic representation of the appearance of a three dimensional object.

Perspective projection:

Object positions are transformed to the view plane along lines that converge to a point called projection reference point (or center of projection).

The projected view of an object is determined by calculating the intersection of the projection plane with the view plane.

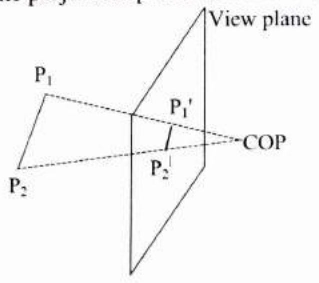


Figure 4.15: Perspective projection of an object to view plane

Perspective projection produces realistic views but does not preserve relative proportions.

Projection of distant objects are smaller than the projection of objects of the same size that are closer to the view plane.

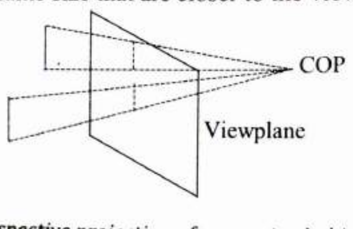


Figure 4.16: Perspective projection of equa- sized objects at different distances from the view plane

Parallel projections:

We can specify a parallel projection with a projection vector that denies the direction for the projection line.

Orthographic parallel projection:

The projection is perpendicular to the view plane.

Oblique parallel projection:

The projection is not perpendicular to the view plane.

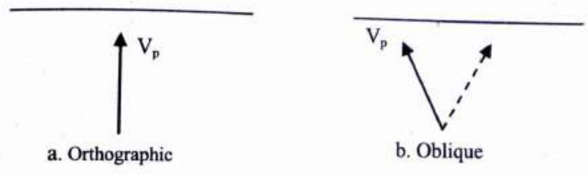


Figure 4.17: *Orientation of the projection vector* V_p *to produce an orthographic projection (a) and an oblique projection (b)*

- Orthographic projections are used to produce elevation (front, side, real projections) and plan view (top projection)
- Used in engineering and architectural drawings.
- We can also form orthographic projections that display more than one face of an object such views are called axonometric orthographic projections.
- Most generally used axonometric projection is the isometric projection.
- We generate isometric projection by aligning the projection plane so that it intersects each co-ordinate axis in which the object is defined (principal axes) at the same distance from the origin.
- If view plane is placed at position z_{vp} along the zv axis, then any point (x, y, z) in viewing co-ordinates is transformed to projection co-ordinate as, $x_p = x, y_p = y$

Where the original z co-ordinate value is preserved for the depth information needed in visible surface detection procedures.

Oblique projection

- An oblique projection is obtained by projecting point along parallel lines that are not perpendicular to the projection plane.
- Some application packages define an obligate projection vector with two angles α and ϕ.

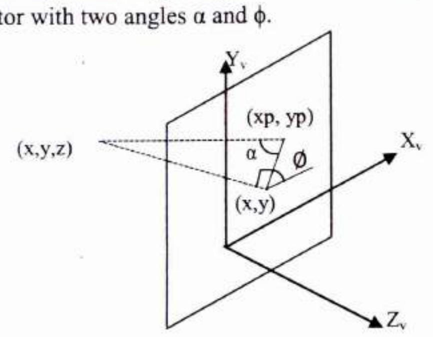


***Figure 4.18:** Oblique projection of coordinate position (x, y, z) to position (x_p, y_p) on the view plane*

- Point (x, y, z) is projected to position (xp, yp) an the view plane
- (x, y) is the orthographic projection co-ordinates of (x, y, z)

  Oblique projection line from (x, y, z) to (xp, yp) makes an angle ∝with the line on the projection plane that joins (xp, yp) and (x, y)
- This line of length L, is at an angle Ø with the horizontal direction in the projection plane

  then, $xp = x + L\cos Ø$

  $yp = y + L\sin Ø$

  But, $\tan ∝ = \frac{z}{L}$

  $L = \frac{z}{\tan ∝} = ZL1$

Now,

$xp = x + zL1\cos Ø$

$yp = y + zL1\sin Ø$

Then, the transformation matrix for producing any parallel projection onto the $x_v y_v$ Plane becomes

$$M_{parallel} = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- An or the graphic projection is obtained when $L_1 = 0$ (which occurs at a projection angle $\alpha = 90$)
- Oblique projections are generated with non zero values for L_1
- Projection matrix for parallel projection is similar to that of a z axis shear matrix
- In fact, the effect of this projection matrix is to shear planes of constant Z and project them onto view plane
- The x and y co-ordinate values within each plane of constant z are shifted by an amount proportional to the z value of the plane so that angles, distances and parallel lines in the plane are projected accurately.

Perspective projection

Suppose that cop is at z_{prp} along z_v axis, and view plane is at z_{vp}

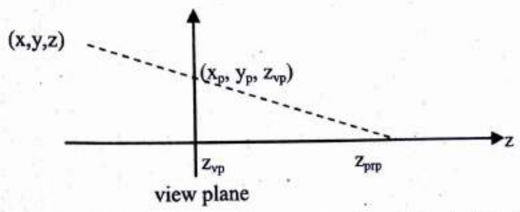


***Figure 4.19:** Perspective projection of a point **P** with coordinates (x, y, z) to position (x_p, y_p, z_{vp}) on the view plane*

We can write equations describing co-ordinate positions along the perspective projection line as

$x' = x - xu$ .... (i)

$y' = y - yu$ ....(ii)

$z' = z - (z - z_{prp})u$ ....(iii)

Where u varies from 0 to 1

When $u = 0$,

$x' = x$

$y' = y$

$z' = z$, at original position

When $u = 1$,

$x' = 0$

$y' = 0$

$z' = z_{prp}$ at the projection reference point

On the view plane $z' = z_{vp}$, so equation (iii) becomes $z_{vp} = z - (z - z_{prp})u$

or $u = \frac{z_{vp} - z}{z_{prp} - z}$ iv

also, $x' = x_p$ and $y' = y_p$, so equation(i) and (ii) becomes

$$x_p = x - x\frac{(z_{vp} - z)}{(z_{prp} - z)}$$

$$= x \times \frac{(z_{prp} - z - z_{vp} + z)}{(z_{prp} - z)}$$

$$= x \times \frac{(z_{prp} - z_{vp})}{(z_{prp} - z)}$$

$$x_p = x \times \frac{(d_p)}{(z_{prp} - z)} \text{.................(v)}$$

Where $d_p = z_{prp} - z_{vp}$, is the difference between COP and view plane distances

also,

$$y_p = y - y\frac{(z_{vp} - z)}{(z_{prp} - z)}$$

$$= y \times \frac{(z_{prp} - z - z_{vp} + z)}{(z_{prp} - z)}$$

$$= y \times \frac{(z_{prp} - z_{vp})}{(z_{prp} - z)}$$

$$y_p = y \times \frac{(d_p)}{(z_{prp} - z)} \text{..............(vi)}$$

And $z_p = z_{vp}$ ....(vii)

using equation (v), (vi) and (vii), we form a matrix

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} d_p/(z_{prp} - z) & 0 & 0 & 0 \\ 0 & d_p/(z_{prp} - z) & 0 & 0 \\ 0 & 0 & 0 & z_{vp} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/d_p & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

where $h = \frac{z_{prp} - z}{d_p}$

$x_p = \frac{x_h}{h}$, $y_p = \frac{y_h}{h}$

and original z co-ordinate value would be retained for visible surface and other depth processing

Special cases

i. When $z_{vp} = 0$, view plane passes through origin

$$x_p = x \times \frac{(d_p)}{(z_{prp} - z)} = x \times \frac{1}{(1 - z/z_{prp})}$$

$$y_p = y \times \frac{(d_p)}{(z_{prp} - z)} = y \times \frac{1}{\left(1 - \frac{z}{z_{prp}}\right)}$$

ii. When $z_{prp} = o$, reference point at origin

$$x_p = x \times \frac{z_{vp}}{z} = x \times \frac{1}{(z/z_{vp})}$$

$$y_p = y \times \frac{z_{vp}}{z} = y \times \frac{1}{(z/z_{vp})}$$

Vanishing point

- A set of parallel lines that are not parallel to view plane are projected as converging lines that appear to converge at a point called vanishing point.
- A set of parallel lines that are parallel to view plane are projected as parallel lines.

- More than one set of parallel lines form more than one vanishing point in the scene.
- The use of vanishing point is for realistic representation

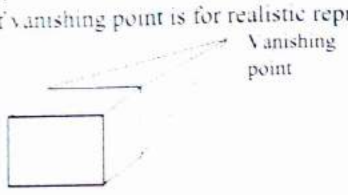


Figure 4.20 (a): Perspective views and vanishing point

Principal vanishing point

- Principal vanishing points are formed by the apparent intersection of lines parallel to one of the three principal x, y z axes
- The vanishing point for any set of lines that are parallel to one of the principal axes of an object is referred to as a principal vanishing point.
- The number of principal vanishing point is determined by number of principal axes intersected by the view plane.

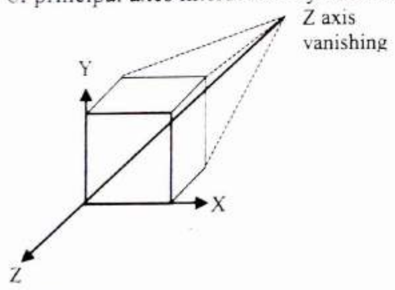


***Figure 4.20 (b):** Perspective views and z axis vanishing point*

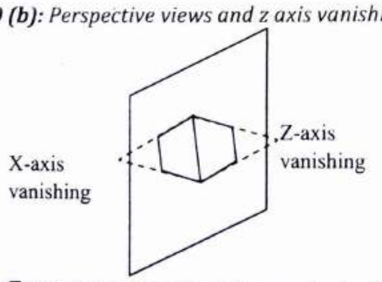
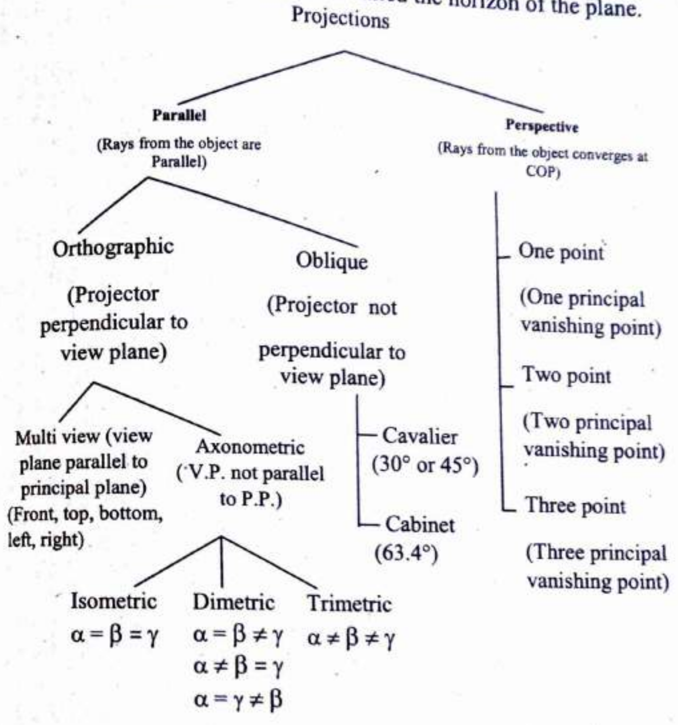
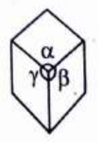


***Figure 4.20 (c):** Two-point perspective views and principal vanishing point*

- **We can control the number of principal vanishing point to one, two or three with the orientation of projection plane and classify as one, two or three point perspective projections.**
- **Sets of parallel lines on the same plane lead to collinear vanishing points. The line is called the horizon of the plane.**


Projections
Parallel
(Rays from the object are Parallel)
Perspective
(Rays from the object converges at COP)
Orthographic
(Projector perpendicular to view plane)
Oblique
(Projector not perpendicular to view plane)
One point
(One principal vanishing point)
Two point
(Two principal vanishing point)
Three point
(Three principal vanishing point)
Multi view (view plane parallel to principal plane) (Front, top, bottom, left, right)
Axonometric (V.P. not parallel to P.P.)
Cavalier (30° or 45°)
Cabinet (63.4°)
Isometric
α = β = γ
Dimetric
α = β ≠ γ
α ≠ β = γ
α = γ ≠ β
Trimetric
α ≠ β ≠ γ



SOLVED NUMERICALS

1. ***List down the steps for rotating a 3D object by 90° in counter clockwise direction about an axis joining end points (1, 2, 3) and (10, 20, 30). Also derive the final transformation matrix.*** *[2076 Ashwin Back]*

Steps:

i. Translate (1, 2, 3) to origin so that the rotation axis passes through the origin.

ii. Rotate the line so that the line coincides with one of the axes, say 2 axis.

iii. Rotate the object about that co-ordinate axis by 90° in counter clockwise direction.

iv. Apply the inverse of step (ii) i.e. inverse rotation to bring the rotation axis back to its original orientation.

v. Apply the inverse of step (i) i.e. inverse translation to bring the rotation axis back to its original position.

For step (ii) i.e. for coinciding the arbitrary axis with any co-ordinate axis, the rotations are needed about other two axes.

Direction cosines of the given line is

$$[v] = [(x_1 - x_0)\ (y_1 - y_0)\ (z_1 - z_0)]$$

$$[c_x\ c_y\ c_z] = \left[\frac{(x_1 - x_0)\ (y_1 - y_0)\ (z_1 - z_0)}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}}\right]$$

$$c_x = \frac{x_1 - x_0}{|v|} = \frac{10 - 1}{\sqrt{9^2 + 18^2 + 27^2}} = \frac{9}{33.67}$$

$$c_y = \frac{y_1 - y_0}{|v|} = \frac{18}{33.67}$$

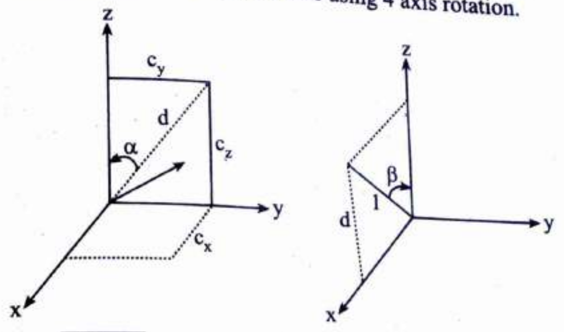
$$c_z = \frac{z_1 - z_0}{|v|} = \frac{27}{33.67}$$

To calculate the angles of rotation about x and y axes we use the direction cosines.

To put the line or rotation axis on the z axis we have to follow two steps

a. First rotate about the x axis to transform vector u into the x z plane.

b. The swing u around to the z axis using 4 axis rotation.



$$d = \sqrt{c_y^2 + c_z^2}$$

$$\cos\alpha = \frac{c_z}{d}$$

$$\sin\alpha = \frac{c_y}{d}$$

$$\cos\beta = d$$

$$\sin\beta = -c_x$$

The complete the sequence of operations can be summarized as

$$T = [T_r]^{-1}[R_x(\alpha)]^{-1}[R_y(\beta)]^{-1}[R_z(\theta)][R_y(\beta)][R_x(\alpha)][T_r]$$

$$[T_r] = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[R_x(\alpha)] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[R_y(\beta)] = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 1 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[R_z(\theta)] = \begin{bmatrix} \cos90^\circ & -\sin90^\circ & 0 & 0 \\ \sin90^\circ & \cos90^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[R_y(\beta)] = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 1 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[R_x(\alpha)]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T_z]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

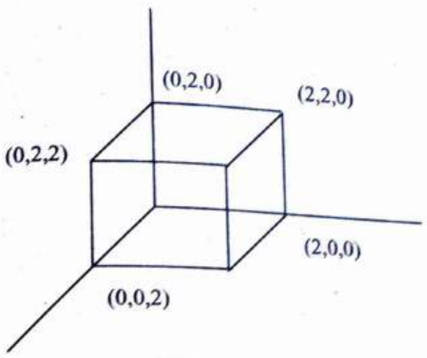
2. ***Develop the matrix to transform an object from three-dimensional world coordinate to viewing coordinate system. A unit length cube with diagonal passing through (0, 0, 0) and (2, 2, 2) is shared with respect to zx - plane with share constants = 3 in both directions. Obtain the final coordinates of the cube after shearing.*** *[2075 Ashwin*

Solution:

Shearing with respect to zx - plane

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & shx & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & shz & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$sh_x = sh_y = 3$



$$A' = \begin{bmatrix} 1 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Similarly, we can calculate B′, C′, D′, E′, F′, G′, H′.

2. ***Obtain perspective projection co-ordinates for the pyramid with vertices of base (15, 15,10), (20,20,10), (25,15,10), (20,10,10) and apex (20,15,20) given that z_{prp} =20 and z_{vp} = 0.***

Solution:

$z_{prp} = 20$, $z_{vp} = 0$

$x' = x - xu$

$y' = y - yu$

$z' = z - (z - z_{prp})u$

On the view plane, $z' = z_{vp}$. So,

$z_{vp} = z - (z - z_{prp})u$

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

For the vertex (15,15,10),

$$x_p = x - x\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

$$= x\left(\frac{z_{prp} - z - z_{vp} + z}{z_{prp} - z}\right)$$

$$= x\left(\frac{Z_{prp} - Z_{vp}}{Z_{prp} - Z}\right)$$

$$= 15\left(\frac{20-0}{20-10}\right)$$

$$= 30$$

$$y_{vp} = y - y\left(\frac{Z_{vp} - Z}{Z_{prp} - Z}\right)$$

$$= 10 - 10\left(\frac{0-10}{20-10}\right)$$

$$= 30$$

$$Z_{vp} = 0$$

Projected points is $(X_1', Y_1', Z_1') = (30,10,0)$

Similarly for $P_2(X_2, Y_2, Z_2) = P_2(20,20,10)$

$$X_{vp} = 20\left(\frac{20-0}{20-10}\right) = 40$$

$$Y_{vp} = 20\left(\frac{20-0}{20-10}\right) = 40$$

$$Z_{vp} = 0$$

Projected points is $(X_2', Y_2', Z_2') = (40, 40, 0)$

Similarly, for vertex (25, 15, 10)

$$X_{vp} = 50$$

$$Y_{vp} = 30$$

$$Z_{vp} = 0$$

Projected points is$P_3(X_3', Y_3', Z_3') = (50, 30, 0)$

for vertex (20, 10, 10)

$$X_{vp} = x - x\left(\frac{Z_{vp} - z}{Z_{prp} - z}\right)$$

$$= 20 - 20\left(\frac{0-10}{20-10}\right)$$

$$Y_{vp} = y - y\left(\frac{Z_{vp} - z}{Z_{prp} - z}\right)$$

$$= 10 - 10\left(\frac{0-10}{20-10}\right)$$

$$= 20$$

$$Z_{vp} = 0$$

Curve Modeling

5.1 Spline Representations

5.1.1 Spline Curve and Spline Surface

Curve is the set of points that are joined continuously. Spline is the smooth curve passing through the set of given points. In computer graphics, continuous curve that are formed with polynomial pieces with certain boundary conditions are called spline curve or simply spline. A spline surface is combination of spline curves or simply splines. A spline surface can be described with two sets of orthogonal spline curves.

In drafting terminology, a spline is a flexible strip used to produce a smooth curve through a designated set of points. Several small weights are distributed along the length of the strip to hold the position of spline curve as required. We can mathematically describe such a curve with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve sections. The general shape of a spline curve is indicated by a set of coordinate positions called ***control points***. A spline curve is defined, modified, and manipulated with operations on the control points.

Splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation path for the objects or image. Typical CAD applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, and ship hulls.

The different types of spline curve are:

1. Piecewise cubic spline
   i. Hermite spline
   ii. Cardinal spline
   iii. Kochanek-Bartels spline

2. Bezier spline
3. B-spline
4. Beta spline

5.1.2 Interpolation and Approximation Splines

We specify a spline curve by giving a set of coordinate positions, called control points, which indicates the general shape of curve. The control points are fitted with piecewise continuous parametric polynomial function in one of the two ways.

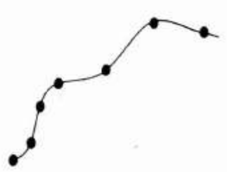


Fig 5.1(a): A set of control points interpolated with piecewise continuous polynomial sections

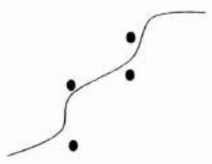


Fig 5.1(b): A set of six points control points approximated with piecewise continuous polynomial sections

When polynomial sections are fitted so that the curve passes through each control point as in **Figure a**, the resulting curve is said to interpolate the set of control points. Interpolation curves are commonly used to digitize drawings or to specify animation paths, and graphs of data trends of discrete set of data points.

When the polynomials are fitted to the general control point path without necessarily passing through any control point, the resulting curve is said to approximate the set of control points (**Figure b**). Approximation curves are primary used as design tools to structure objects surfaces. It is used in drawing contour lines for GIS (Geographical Information System) applications.

Convex hull

The convex polygon boundary that encloses a set of control points is called the convex hull. Convex hulls provide a measure for the deviation of a curve or surface from the region bounding the control points. Some splines are bounded by the convex hull that ensures the polynomials smoothly follow the control points without erratic oscillations. Also, the polygon region inside the convex hull is useful in some algorithms as a clipping region.

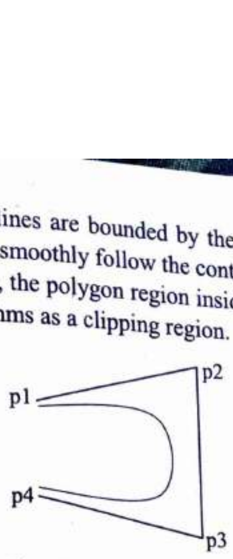


Fig 5.2: Convex hull

Curve Continuity or Smoothness of Curve

There are two approaches which determine the smoothness of curve or curve continuity. They are as follows:

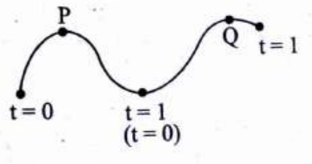
i) Parametric continuity conditions(C)
ii) Geometric continuity conditions (G)

5.1.3 Parametric Continuity Conditions

Parametric continuity deals in parametric equations associated to piecewise parametric polynomial curve not the shape or appearance of the curve. We set parametric continuity by matching the parametric derivatives of adjoining curve sections at their common boundary.

i. Zero order parametric continuity (C^0):

C^0 continuity means that two piece of curves are joined or meet at same point. The two pieces of curve P and Q are in zero order parametric continuity if P (t = 1) = Q (t=0)



ii. **First order parametric continuity (C^1):**

Two successive curve sections are in first order parametric continuity if first parametric derivative of the coordinate function are equal at the joining point.

P'(t=1) = Q'(t=0)

Where P' and Q' are first order derivative.

iii. **Second order parametric continuity (C^2):**

Two curve are in second order parametric continuity if both first and second derivatives of the two curve sections are same at the intersection point.

P''(t=1) = Q''(t=0)

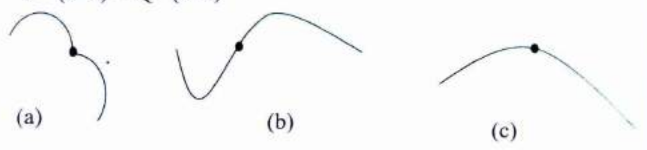


Figure 5.3: Piecewise construction of a curve by joining two curve segments using different order of continuity (a) zero-order continuity only (b) first-order continuity (c) second-order continuity.

5.1.4 Geometric Continuity Conditions

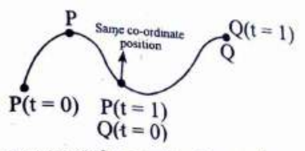
Another method for joining two successive curve sections is to specify conditions for geometric continuity. Geometric continuity refers to the way that a curve or surface looks. In this case, we only require parametric derivatives of the two sections to be proportional to each other at their common boundary instead of equal to each other.

i. **Zero order geometric continuity (G^0):**

Zero-order geometric continuity (G^0 continuity) is the same as zero-order parametric continuity. That is, the two curves sections must have the same coordinate position at the boundary point.

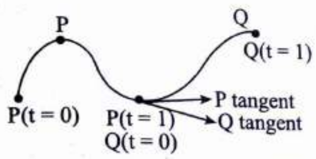
P(t=1) = Q(t=0)

P and Q are two segments of curves.



ii. **First order geometric continuity (G^1):**

First-order geometric continuity (G^1 continuity) means that the parametric first derivatives are proportional at the intersection of two successive sections. If P and Q are two piece of curves, then P'(t) and Q'(t) must have same direction of tangent vector but not necessary to be the same magnitude.

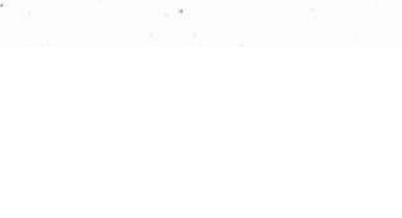


iii. **Second order geometric continuity (G^2):**

Second-order geometric continuity (G^2 continuity) means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under G^2 continuity, curvatures of two curve sections will match at the joining position.

In general C1 continuity implies G1 continuity but G1 continuity doesn't imply C1 continuity.

C1 continuity doesn't imply G1 continuity when segments tangent vector are [0 0 0] at join point. In this case, the tangent vectors are equal but there directions are different.

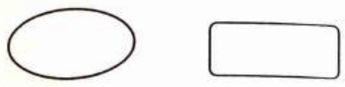


Three types of Curve

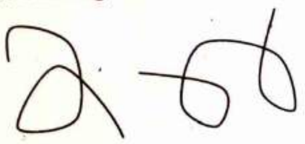
There are three types of curve. They are:

i. Open curve:

ii. Closed curve:



iii. Crossing curve:



Representation of Curve

All objects are not flat but may have many bends and deviations. We have to compute all curves. We can represent curve by three mathematical function:

i) Explicit function

ii) Implicit function

iii) Parametric function

i. Explicit representation of curve:

In this method the dependent variable is given explicitly in terms of the independent variable as;

$y = f(x)$

e.g., $y = mx + c$

$y = 5x^2 + 2x + 1$

In explicit representation, for each single value of x, only a single value of y is computed

ii. Implicit representation of curve:

In this method, dependent variable is not expressed in terms of some independent variables as;

$F(x,y)=0$

e.g.;

$x^2+y^2-1=0$

In implicit representation, for each single value of x, multiple values of y is computed.

If we convert implicit function to explicit function it will be more complex and will give different values.

e.g. $y=\pm\sqrt{1-x^2}$

iii. Parametric representation of curve:

We cannot represent all curves in single equation in terms of only x and y. Instead of defining y in terms of x (i.e. y=f(x)) or x in terms of y (i.e x=h(y)); we define both x and y in terms of a third variable in parametric form.

Curves having parametric form are called parametric curves.

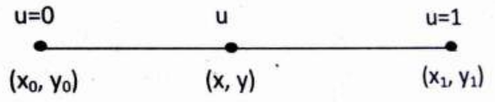
$x = f_x(u)$

$y = f_y(u)$ where u is parameter

similarly, parametric equation of line is;

$x = (1-u)\, x_0 + u\, x_1$

$y = (1-u)\, y_0 + u\, y_1$



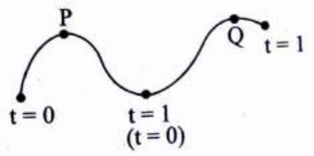
Parametric Curve

The parametric representation for curve is as follows:

$x = x(t)$

$y = y(t)$

$z = z(t)$



cubic polynomial means the polynomials which represent the curve with degree three.

The cubic polynomial that define a curve can be represented as

$Q(t) = [x(t) \quad y(t) \quad z(t)]$

$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$

(Cubic polynomial function equation)

$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$

$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$

$$Q(t) = [t^3 \quad t^2 \quad t \quad 1].\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

$Q(t) = T.C.$

$C = M.G.$

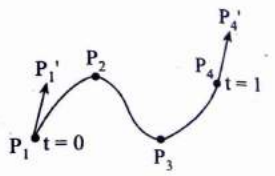
Where M is 4 X 4 basis matrix and G is a four element column vector of geometric constants called geometric vector.

5.2 Hermite Cubic Spline

i. Hermite spline curve is interpolation spline curve (curve passes through control point)

ii. It uses cubic polynomial function (to make Hermite function four point is necessary)

iii. To make Hermite function it uses four point P_1, P_4, P_1', P_4'. Where P_1 and P_4 are position vector and P_1' and P_4' are tangent vectors (first order derivative) which show direction of the curve.



Let Q(t) is the curve $t \in [0, 1]$

$Q(t) = [x(t) \quad y(t) \quad z(t)]$, $t \in [0, 1]$ where all points satisfy cubic parametricity.

The general curve equation is,

$p(t) = at^3 + bt^2 + ct + d$ where $0 \le t \le 1$

So,

$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$

$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$

$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$

$$Q(t) = [t^3 \quad t^2 \quad t \quad 1].\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

$Q(t) = T.C.$

$C = M.G.$

M = basis matrix that provides blending function.

G = Geometric vector

$Q(t) = T.M_H.G_H$

$= [t^3 t^2 \quad t \quad 1].\ M_H.G_H$

$Q_x(t) = P_1(t) = [0\ 0\ 0\ 1]\ M_H.G_H$

$t = 0$

$Q_x(t) = P_4(t)\ [1\ 1\ 1\ 1]\ M_H.G_H$

$t = 1$

$Q'_x(t) = R_1(t) = [3t^2\ 2t\ 1\ 0] M_H.G_H$

$t = 0$

$= [0\ 0\ 1\ 0]\ M_H.G_H$

$Q_H'(t) = R_a(t) = [3\ 2\ 1\ 0]\ M_H.G_H$

$t = 1$

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} M_H G_H$$

$$M_H.G_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

$$M_H = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$G_{HX} = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

$Q(t) = TM_HG_H$

$$Q(t) = [t^3\ t^2\ t\ 1] \begin{bmatrix} 2 & -1 & 1 & 1 \\ -3 & 3 & -2 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

$= (2t^3 - 3t^2 + 1)P_1 + (-2t^2 + 3t^2)P_2 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$

$= P_1H_0(t) + P_4H_1(t) + R_1H_2(t) + R_4H_3(t)$

Where $H_0(t)$, $H_1(t)$, $H_2(t)$, $H_3(t)$ are Hermite blending function.

Alternative way to derive Hermite curve

A parametric cubic curve is defined as

$$P(t) = \sum_{i=0}^{3} a_i t^i \quad 0 \le t \le 1 \qquad \text{................(i)}$$

Where, $P(t)$ is a point on the curve.

Expanding equation (i) yields,

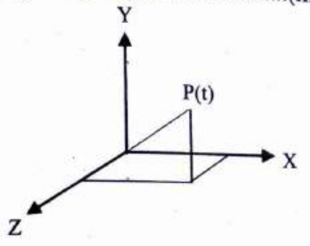
$P(t) = a_3t^3 + a_2t^2 + a_1t + a_0$ ....(ii)

Separating three components of $P(t)$, we have

$x(t) = a_{3x}t^3 + a_{2x}t^2 + a_{1x}t + a_0x$

$y(t) = a_{3y}t^3 + a_{2y}t^2 + a_{1y}t + a_0y$

$z(t) = a_{3z}t^3 + a_{2z}t^2 + a_{1z}t + a_{0z}$........................(iii)



In order to solve equation (iii), twelve unknown coefficients must be specified. From the known end point coordinates of each segment, six of the twelve needed equations are obtained. The other six are found by using tangent vectors at the two ends of each segment.

The direction of the tangent vectors establishes the slopes (direction cosines) of the curve at the end points. This procedure for defining a cubic curve using ends points and tangent vector is one form of Hermite interpolation (cubic spline).

Each cubic curve segment is parameterized from 0 to 1 so that known end points correspond to the limit values of the parametric variable t, that is, $P(0)$ and $P(1)$.

Substituting $t = 0$ and $t = 1$ in equation (ii), we have

$P(0) = a_0$ ....(iv)

$P(1) = a_3 + a_2 + a_1 + a_0$ ....(v)

To find the tangent vectors, equations (ii) must be differentiated with respect to t,

$P'(t) = 3a_3t^2 + 2a_2t + a_1$

The tangent vectors at the two end points are found by substituting t = 0 and t = 1 in the above equation.

$P'(0) = a_1$ ....(vi)

$P'(1) = 3a_3 + 2a_2 + a_1$ ....(vii)

The values of $a_2$and a_3 can be determined by solving equations (iv), (v), (vi), and (vii)

$a_0 = P(0)$

$a_1 = P'(0)$

$a_2 = -3P(0) + 3P(1) - 2P'(0) - P'(1)$

$a_3 = 2P(0) - 2P(1) + P'(0) + P'(1)$

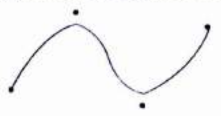
Substituting these values of a_i in equation (ii) and rearranging the terms yields

$$P(t) = (2t^3 - 3t^2 + 1)P(0) + (-2t^3+3t^2)P(1) + (t^3-2t^2+t)P'(0) + (t^3 - t^2)P'(1)$$

The values of P(0), P(1), P'(0), P'(1) are called geometric coefficients and represent the known vector quantities. The polynomial coefficients of these vector quantities are commonly known as blending functions. By varying parameter t in these blending functions from 0 to 1, several points on curve segments can be found.

5.3 Bezier Curves

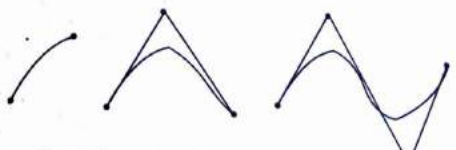
i. Bezier curve is approximate spline curve



ii. Instead of end points and tangents, we have four control points in the case of cubic Bezier curve.

iii. It has Bernstein polynomial function (Its own polynomial function to provide continuity)

iv. All control points uses to make Bezier curve.



Curve doesn't go out of polygon boundary (Convex hull)

- Bezier splines are widely used in various CAD system COREL DRAW packages and many more Graphic packages.
- As with splines, a Bezier curve can be specified with boundary conditions with a characterizing matrix or with blending function. For general Bezier curves, the blending function specification is most convenient.

The Bezier curve was developed by the French engineer Pierre Bezier for use in the design of Renault automobile bodies. Bezier spline is highly useful and convenient for curve and surface design. They are also easy to implement. For these reasons, Bezier splines are widely available in various CAD systems, in general graphics packages, and in assorted drawing and painting packages.

The control points are blended using Bernstein polynomials to compute a set of position vectors **P**(u) which are then joined by straight line segments to get the curve.

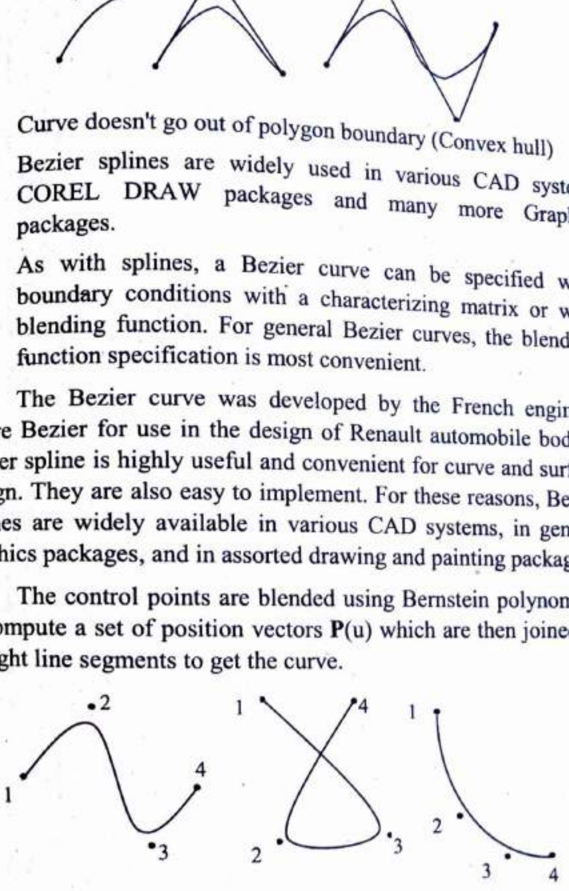


Figure 5.4: *Bezier curves generated four control points*

Suppose we are given n+1control-point positions: $\mathbf{p}_k = (x_k, y_k, z_k)$ where k varies from0 to n. These points can be blended to produce the position vector **P**(u), which describes the path of an approximating Bezier polynomial function between P_0 and P_n.

$$P(u) = \sum_{k=0}^{n} p_k \; BEZ_{k,n}(u) \quad \text{..........(i)} \quad \text{where } 0 \le u \le 1$$

The Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials

$$BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k} \quad \text{..........(ii)}$$

where C(n,k) are the binomial coefficients and is given as

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

Equation (i) represents a set of three parametric equations for the individual curve coordinates,

$$P_x(u) = \sum_{k=0}^{n} x_k \; BEZ_{k,n}(u)$$

$$P_y(u) = \sum_{k=0}^{n} y_k BEZ_{k,n}(u)$$

$$P_z(u) = \sum_{k=0}^{n} z_k BEZ_{k,n}(u)$$

If we take n = 3, then number of control points= n+1 = 4. Then the above equations become

$P_x(u) = x_0\, BEZ_{0,3}(u)+x_1 BEZ_{1,3}(u)+x_2 BEZ_{2,3}(u)+x_3 BEZ_{3,3}(u)$

$P_y(u) = y_0\, BEZ_{0,3}(u)+y_1 BEZ_{1,3}(u)+y_2 BEZ_{2,3}(u)+y_3 BEZ_{3,3}(u)$

$P_z(u) = z_0\, BEZ_{0,3}(u)+z_1 BEZ_{1,3}(u)+z_2 BEZ_{2,3}(u)+z_3 BEZ_{3,3}(u)$

where

$BEZ_{0,3}(u) = C(3,0)\, u^0(1-u)^3 = (1-u)^3$

$BEZ_{1,3}(u) = C(3,1)\, u^1(1-u)^2 = 3u(1-u)^2$

$BEZ_{2,3}(u) = C(3,2)u^2(1-u) = 3u^2(1-u)$

$BEZ_{3,3}(u) = C(3,3)u^3(1-u)^0 = u^3$

Properties of Bezier curve:

1. Starting and ending control points lie on the curve.
   $P(0) = P_0$
   $P(1) = P_n$

2. They generally follow the shape of the control polygon which consists of the segments joining the control points.
3. Multiple control points of same coordinate values give more weight to that point and as a result, the curve is pulled nearer to that point.
4. The curve lies within the convex hull of the region created by joining the control point. This is because $\sum_{k=0}^{n} BEZ_{k,n} = 1$.

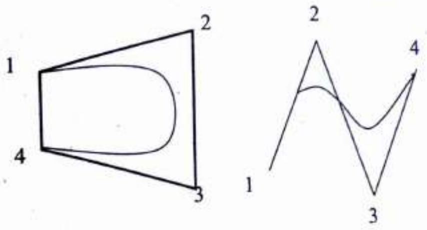


Figure 5.5: *Two dimensional Bezier curve*

5. The slope at the beginning of the curve is along the line joining the first two control points, and the slope at the end of the curve is along the line joining the last two endpoints.
6. Collinear control points with same coordinate values produce a point.
7. Control point do not have local control over the shape of the curve.
8. The degree of the polynomial defining the curve segment is one less than the number of defining control polygon points. For 4 control points the degree of polynomial is 3. i.e. cubic Bezier curve.

Drawback:

- The degree of Bezier curve depends on number of control points.
- Bezier curve exhibit global control property means moving a control point alters the shape of the whole curve.

Bezier surfaces

Two sets of orthogonal Bezier curves can be used to design an object surface by specifying by an input mesh of control points. The parametric vector functions for the Bezier surface is formed as the Cartesian product of Bezier blending functions

$$P(u,v) = \sum_{k=0}^{n}\sum_{j=0}^{n} p_{j,k} \cdot BEZ_{k,n}(u)$$

with $\mathbf{p}_{j,k}$ specifying the location of the (m+1) by (n+1) control points.

5.4 B-spline Curve

B-spline curve was developed to overcome the limitation or demerits of bezier curve. Demerits of Bezier curve are as follows:

1. Polynomial degree is decided by control points

   If C.P. = 5 then

   P.D. = 5–1 = 4

2. Blending function is non-zero for all parameter value over the entire curve. Due to this change in one vertex, changes the entire curve and this eliminates the ability to produce a local change within a curve.

Properties of B-spline curve

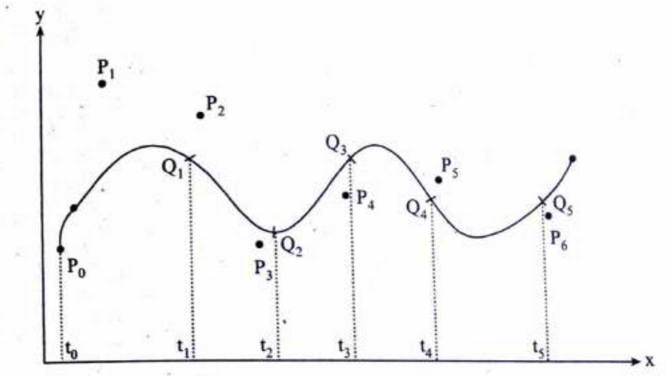
1. B-spline approximate spline curve with local effect. In this curve, each control point affects the shape of the curve only over range of parameter values where its associated basis function is non-zero.
2. B-spline curve made up of n+1 control point.
3. B-spline curve let us specify the order of basis (k) function and the degree of the resulting curve is independent on the no. of vertices.
4. It is possible to change the degree of the resulting curve without changing the no. of control points.
5. B-spline can be used to define both open & close curves.
6. Curve generally follows the shape of defining polygon. If we have order k = 4 then degree will be 3 $P(k) = x^3$.

7. The curve line within the convex hull of its defining polygon.

   In B-spline we segment out the whole curve which is decided by the order (k). By formula 'n–k+2'

For example:

If we have 7 control points and order of curve k=3 then n = 6 and this B-spline curve has segments 6 – 3 + 2 = 5



Five segments Q_1, Q_2, Q_3, Q_4, Q_5

Segment	Control points	Parameter
Q_1	$P_0\ P_1\ P_2$	$t_0 = 0, t_1 = 1$
Q_2	$P_1\ P_2\ P_3$	$t_1 = 1, t_2 = 2$
Q_3	$P_2\ P_3\ P_4$	$t_2 = 2, t_3 = 3$
Q_4	$P_3\ P_4\ P_5$	$t_3 = 3, t_4 = 4$
Q_5	$P_4\ P_5\ P_6$	$t_4 = 4, t_5 = 5$

There will be a join point or knot between Q_{i-1} & Q_i for $i \geq 3$ at the parameter value t_i know as KNOT VALUE [X].

If P(u) be the position vectors along the curve as a function of the parameter u, a B-spline curve is given by

$$P(u) = \sum_{i=0}^{n} P_i N_{i,k}(u) \quad 0 \leq u \leq n-k+2$$

$N_{i,k}(u)$ is B-spline basis function

$$N_{i,k}(u) = \frac{(u - X_i)\, N_{i,k-1}(u)}{X_{i+k-1} - X_i} + \frac{(X_{i+k} - u)\, N_{i+1,k-1}(u)}{X_{i+k} - X_{i+1}}$$

The values of X_i are the elements of a knot vector satisfying the relation $X_i \le X_{i+1}$.

The parameter u various form 0 to n–k+2 along the P(u).

So there are some conditions for finding the KNOT VALUES [X]

$X_i\,(0 \le i \le n + k)$: Knot values

$X_i = 0$ if $i < k$

$X_i = i - k + 1$ if $k \le i \le n$

$X_i = n–k+2$ if $i > n$

So as B-spline curve has Recursive Equation. So we stop at

$N_i, K(u) = 1$ if $X_i \le u\, x_{i+1}$

$= 0$ otherwise

Example:

$n = 5, k = 3$

then $X_i(0 \le i \le 8)$ knot values

$X_i\{0,0,0,1,2,3,4,4,4,4,\}$

$N_{0,3}(u) = (1–u)^2.N_{2,1}(u)$

When $i = 0$, $k = 3$ so $i < k$ is true

$X_0 = 0$

$i = 1, k = 3\ X_1 = 0$

$i = 2, k = 3\ X_2 = 0$

$i = 3, k = 3\ X_3 = i–k+1 = 3–3+1$

$X_3 = 1$

$i = 4, k = 3\ X_4 = i–k+1 = 4–3+1$

$X_4 = 2$

$i = 8, k = 3\ X_8 = i > n\ n - k + 2 = 5 - 3 + 2 = 4$

In this way we will calculate

SOLVED NUMERICALS

1. ***Find the Bezier curve which passes through (0,0,0) and (-2, 1, 1) and is controlled by (7,5,2) and (2,0,1).***

[2076 Ashwin Back]

Solution:

In Bezier curve the starting and ending control points lie on the curve so (0,0,0) and (–2,1,1) are starting and ending control points and (7, 5, 2) and (2, 0, 1) are intermediate control points.

We know,

Position vector $P(u) = \sum_{k=0}^{n} P_k\, BEZ_{k,n}(u)$ for $0 \le u \le 1$

where,

P_k is control point position

We have 4 control points so P_k varies $k = 0$ to 3

The Bezier blending function $BEZ_{k,n}(u)$ are the Bernstein polynomials,

$BEZ_{k,n}(u) = C(n, k)\, u^k(1 - r)^{n-k}$

where, C(n, k) are binomial coefficient

$$C(n, k) = \frac{n!}{k!(n - k)!}$$

So,

$P_x(u)=x_0BEZ_{0,3}(u) + x_1BEZ_{1,3}(u) + x_2BEZ_{2,3}(u) + x_3BEZ_{3,3}(u)$

$P_y(u)=y_0BEZ_{0,3}(u) + y_1BEZ_{1,3}(u) + y_2BEZ_{2,3}(u) + y_3BEZ_{3,3}(u)$

$P_z(u)= z_0BEZ_{0,3}(u) + z_1BEZ_{1,3}(u) + z_2BEZ_{2,3}(u) + z_3BEZ_{3,3}(u)$

Where,

$BEZ_{0,3}(u) = C(3,0)u^0\,(1 - u)^3 = (1 - u)^3$

$BEZ_{1,3}(u) = C(3,1)u^1(1 - u)^2 = 3u(1 –u)^2$

$BEZ_{2,3}(u) = C(3,2)u^2(1 - u) = 3u^2(1 - u)$

$BEZ_{3,3}(u) = C(3,3)u^3(1 - u)^0 = u^3$

Now,

$P_x(u)=x_0BEZ_{0,3}(u) + x_1BEZ_{1,3}(u) + x_2BEZ_{2,3}(u) + x_3BEZ_{3,3}(u)$

$=0*(1\text{-}u)+ 7*3u(1 –u)^2+2*3u^2(1 - u)+ (-2)*u^3$

$P_y(u)=y_0BEZ_{0,3}(u) + y_1BEZ_{1,3}(u) + y_2BEZ_{2,3}(u) + y_3BEZ_{3,3}(u)$

$=0*(1\text{-}u)+ 5*3u(1 –u)^2+0*3u^2(1 - u)+ 1*u^3$

$P_z(u)= z_0BEZ_{0,3}(u) + z_1BEZ_{1,3}(u) + z_2BEZ_{2,3}(u) + z_3BEZ_{3,3}(u)$

$=0*(1\text{-}u)+ 2*3u(1 –u)^2+1*3u^2(1 - u)+ 1*u^3$

2. ***A parametric cubic curve passes through the point (0,0), (2,4), (4,3) and (5,-2) which are parameterized at t = 0, ¼, ¾ and 1 respectively. Determine the geometric coefficient matrix and the slope of the curve when t = 0.5***

Solution:

The points on the curve are

(0,0) at t = 0

(2,4) at t = 1/4

(4,3) at t = 3/4

(5,-2) at t = 1

P(t) = $[2t^3-3t^2+1]P(0)+[(-2t^3+3t^2)]P(1)+[(t^3-2t^2+t)]$ P'(0) $+[(t^3-t^2)]P'(1)$

In matrix form the equation can be written as.

$$P(t) = [t^3\ t^2\ t\ 1]\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 2 & 4 \\ 4 & 3 \\ 5 & -2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \left(\frac{1}{4}\right)^3 & \left(\frac{1}{4}\right)^2 & \left(\frac{1}{4}\right) & 1 \\ \left(\frac{3}{4}\right)^3 & \left(\frac{3}{4}\right)^2 & \left(\frac{3}{4}\right) & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0.0156 & 0.0625 & 0.25 & 1 \\ 0.4218 & 0.5625 & 0.75 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix}$$

$$\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 5 & -2 \\ 10.33 & 22 \\ 4.99 & -26 \end{bmatrix}$$

The geometric coefficients are:

p(0) = (0,0)

p(1) = (5, −2)

$$p'(0) = \frac{22}{10.33}$$

$$p'(1) = \frac{-26}{4.99}$$

The slope at t = 0.5 is found by taking the first derivative of the above equation as follows.

$$p'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix}\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 5 & -2 \\ 10.33 & 22 \\ 4.95 & -26 \end{bmatrix}$$

Therefore

P'(0.5) = [3.67 -2.0]

$$\text{Slope} = \frac{-2.0}{3.67} = -0.545$$

3. ***Find equation of Bezier curve whose control points are P_0 (2, 6), P_1 (6, 8) and P_2 (9, 12). Also find co-ordinate of point at u = 0.8.*** *[2071 Chaitra]*

Solution:

Control points are $P_0(2, 6)$, $P_1(6, 8)$ and $P_2(9, 12)$

Number of control points = 3

A Bezier curve is a polynomial of degree one less than the number of control points.

Degree of polynomial = 3−1 = 2

u = 0.8

$$P(u) = \sum_{k=0}^{n} p_k \qquad BEZ_{k,n}(u)$$

n = 2, so,

$$P(u) = \sum_{k=0}^{2} p_k \qquad BEZ_{k,2}(u)$$

2. ***A parametric cubic curve passes through the point (0,0), (2,4), (4,3) and (5,-2) which are parameterized at t = 0, ¼, ¾ and 1 respectively. Determine the geometric coefficient matrix and the slope of the curve when t = 0.5***

Solution:

The points on the curve are

(0,0) at t = 0

(2,4) at t = 1/4

(4,3) at t = 3/4

(5,-2) at t = 1

$$P(t) = [2t^3-3t^2+1]P(0)+[(-2t^3+3t^2)]P(1)+[(t^3-2t^2+t)] \quad P'(0) +[(t^3-t^2)]P'(1)$$

In matrix form the equation can be written as

$$P(t) = [t^3\ t^2\ t\ 1]\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 2 & 4 \\ 4 & 3 \\ 5 & -2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \left(\frac{1}{4}\right)^3 & \left(\frac{1}{4}\right)^2 & \left(\frac{1}{4}\right) & 1 \\ \left(\frac{3}{4}\right)^3 & \left(\frac{3}{4}\right)^2 & \left(\frac{3}{4}\right) & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0.0156 & 0.0625 & 0.25 & 1 \\ 0.4218 & 0.5625 & 0.75 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix}$$

$$\begin{bmatrix} p(0) \\ p(1) \\ p'(0) \\ p'(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 5 & -2 \\ 10.33 & 22 \\ 4.99 & -26 \end{bmatrix}$$

The geometric coefficients are:

$p(0) = (0,0)$

$p(1) = (5, -2)$

$p'(0) = \frac{22}{10.33}$

$p'(1) = \frac{-26}{4.99}$

The slope at t = 0.5 is found by taking the first derivative of the above equation as follows.

$$p'(t) = [3t^2 \quad 2t \quad 1 \quad 0]\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 5 & -2 \\ 10.33 & 22 \\ 4.95 & -26 \end{bmatrix}$$

Therefore

$P'(0.5) = [3.67 \quad -2.0]$

$$\text{Slope} = \frac{-2.0}{3.67} = -0.545$$

3. ***Find equation of Bezier curve whose control points are P_0 (2, 6), P_1 (6, 8) and P_2 (9, 12). Also find co-ordinate of point at u = 0.8.*** *[2071 Chaitra]*

Solution:

Control points are $P_0(2, 6)$, $P_1(6, 8)$ and $P_2(9, 12)$

Number of control points = 3

A Bezier curve is a polynomial of degree one less than the number of control points.

Degree of polynomial = 3–1 = 2

u = 0.8

$$P(u) = \sum_{k=0}^{n} p_k \qquad BEZ_{k,n}(u)$$

n = 2, so,

$$P(u) = \sum_{k=0}^{2} p_k \qquad BEZ_{k,2}(u)$$

$BEZ_{k,2}(u) = C(2,k)u^k(1-u)^{2-k}$

$C(2,k) = \frac{2!}{k!(2-k)!}$

$P_x(u) = \sum_{k=0}^{2} x_k BEK_{k,2}(u)$

$BEZ_{0,2}{}^{(u)} = C(2,0)u^0(1-u)^2 = (1-0.8)^2$

$BEZ_{1,2}{}^{(u)} = C(2,1)u^1(1-u)^1 = 2*(0.8)^1*(1-0.8)^1$

$BEZ_{0,2}{}^{(u)} = C(2,0)u^2(1-u)^0 = (0.8)^2*(1-0.8)^0$

$P_x(u) = x_0\,BEZ_{0,2}(u)+x_1\,BEZ_{1,2}(u)+x_2\,BEZ_{2,2}(u)$

$=2*(1-0.8)^2+6*2*(0.8)^1*(1-0.8)^1+9*(0.8)^2*(1-0.8)^0$

$= 7.76$

$P_y(u) = y_0\,BEZ_{0,2}(u)+y_1\,BEZ_{1,2}(u)+y_2\,BEZ_{2,2}(u)$

$=6*(1-0.8)^2+8*2*(0.8)^1*(1-0.8)^1+12*(0.8)^2*(1-0.8)^0$

$=10.48$

The coordinate point is (7.76, 10.48)

4. ***A cubic Bezier curve is described by the four control points. (0,0) , (2,1) , (5,2) and (6,1) Find the tangent to the curve at t = 0.5***

Solution:

Here, we know the Bezier cubic polynomial equation.

$P(t) = (1-t)^3P_0+3t(1-t)^2P_1+3t^2(1-t)P_2+t^3P_3$

The tangent is given by the derivative of the general equation above,

$P'(t) = 3*(1-t)^2P_0+6t(1-t)P_1-6tP_1+3t(1-t)^2P_1-3t^2P_2+6t(1-t)P_2 + 3t^2P_3$

$x'(t) = 3*(1-t)^2x_0+6t(1-t)x_1-6tx_1+3t(1-t)^2x_1-3t^2x_2+6t(1-t)x_2 + 3t^2x_3$

$= 3*(1-0.5)^2(0)+6(0.5)(1-0.5)(2)-6(0.5)(2)+3(0.5)(1-0.5)^2(2)-3(0.5)^2(5)+6(0.5)(1-0.5)(5)+3(0.5)^2(6)$

$= 6.75$

$y'(t) = 3*(1-t)^2y_0+6t(1-t)y_1-6ty_1+3t(1-t)^2y_1-3t^2y_2+6t(1-t)y_2+3t^2y_3$

$= 1.5$

Or, In matrix form this equation is written as

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Where,

$V_0 = (0,0)$

$V_1 = (2,1)$

$V_3 = (5,2)$

$V_4 = (6,1)$

The tangent is given by the derivative of the general equation above,

$$P'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

At t = 0.5, we get

$$P'(t) = \begin{bmatrix} 3(0.5)^2 & 2(0.5) & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$= [6.75 \quad 1.5 \quad 0 \quad 1]$

5. ***Design a Bezier curve controlled by points A(1,1), B(2, 3), C(4, 3) and D(6, 4)***

Solution:

Control point = 4

Degree of polynomial = n – 1 = 3

A(1, 1) = p_0

B(2, 3) = p_1

C(4, 3) = p_2

D(6, 4) = p_3

Equation of Bezier curve

$$p(u) = \sum_{k=0}^{n} p_k . B_{k,n}(u)$$

where p(u) = position vector

p_k = control point

$$B_{k,n} = \frac{n!}{k!(n-k)!} u^k (1-u)^{n-k}$$

$$B_{0,3}(u) = ((3, 0)\ u^0 . (1-u)^{3-0}$$

$$= \frac{3!}{0!3!}(1-u)^3$$

$$= (1-u)^3$$

$$B_{1,3}(u) = C(3,1)\ u^1\ (1-u)^{3-1}$$

$$= 3u.(1-u)^2$$

$$B_{2,3}(u) = C(3,2).u^2(1-u)^{3-2}$$

$$= 3u^2(1-u)$$

$$B_{3,3}(u) = C(3, 3)\ u^3(1-u)^{3-3} = u^3$$

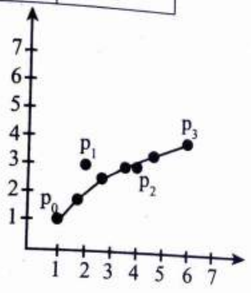
$$p_0 = p_0(1-u)^3 + p_1.3u(1-u)^2 + p_2 3u^2(1-u) + p_3 u^3$$

To find x(u) and y(u)

$$x_0 = x_0(1-u)^3 + x_1.3u(1-u)^2 + x_2 3u^2(1-u) + x_3 u^3$$

$$y_0 = y_0(1-u)^3 + y_1.3u(1-u)^2 + y_2 3u^2(1-u) + y_3 u^3$$

u	x(u)	y(u)
0	1	1
0.2	1.712	1.984
0.4	2.616	2.632
0.6	3.664	3.088
0.8	4.808	3.496
1	1	1



Surface Modeling

6.1 Three-Dimensional Object Representations

Graphics scenes can contain many different kinds of objects: trees, flowers, clouds, rocks, water, bricks, rubber, paper, marble, steel, glass, plastic, and so on. So, there is no one method that can be used to describe objects that will include all characteristics of these different materials. To provide realistic displays of scenes, we need to use representations that accurately model object characteristics.

Polygon and quadric surfaces provide precise descriptions for simple objects like polyhedrons and ellipsoids. Spline surfaces are useful for designing aircraft wings, gears, and other engineering structure with curved surfaces. Procedural methods such as fractal construction and particle systems are useful for accurately representing clouds, clumps of grass, and other natural objects. Octree encodings are used to represent internal features of object, such as those obtained from medical CT images.

Representation schemes for solid objects are divided into two broad categories: Boundary representations and space-partitioning representations; although not all representations fall neatly into one or other these categories.

1) **Boundary representations**

   Boundary representation method is used to describe a three-dimensional object as a set of surfaces that separate the object interior from the environment. E.g., polygon surfaces, curved surfaces.

2) **Space-partitioning representations**

   Space partitioning representation method is used to describe interior properties by partitioning the region containing an object into a set of small, non overlapping contiguous solids (usually cubes). E.g., octree representation

6.2 Polygon Surfaces

The most commonly used boundary representation for a three-dimensional object is a set of surface polygons that enclose the object interior. Many graphics systems store all object descriptions as sets of surface polygons. This makes surface rendering and display of objects simple and fast because all surfaces are described with linear equations. Almost all graphics packages provide this type of object representations. This is the reason why polygon descriptions are often referred to as "standard graphics objects".

Polygon surface representation of an object is created by dividing the boundary surface into a number of interconnected polygons. But for some objects, surfaces are simply tiled to produce the polygon mesh approximation. Such representations are common in design and modeling applications, since the wireframe outline can be displayed quickly to give a general indication of the surface structure. Realistic renderings are produced by interpolating shading patterns across the polygon surfaces to eliminate or reduce the presence of polygon edge boundaries.

6.3 Polygon Table

A polygon surface is specified with a set of vertex coordinates and associated attribute parameters. Information of each polygon is stored in polygon data tables that are used to process, display, and manipulate the objects in a scene.

Polygon data tables store the coordinate description and parameters that specify the spatial orientation of polygon surfaces as well as the attribute parameters that specify the surface characteristics such as surface reflectivity, degree of transparency, surface texture, etc.

Polygon data tables are organized into two groups: geometric tables and attribute tables.

1. **Geometric table**

   This table contains vertex coordinates and parameters that specify the spatial orientation of the polygon surfaces.

Geometric data tables are usually organized into three lists:

i. Vertex table

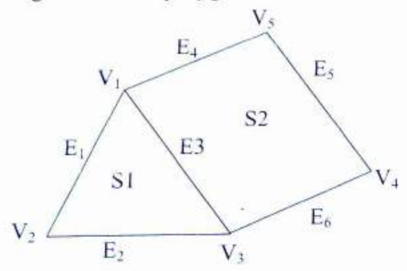
It stores coordinate values for each vertex in the object.

ii. Edge table

It contains pointers back into the vertex table to identify the vertices for each polygon edge.

iii. Polygon-surface table

It contains pointers back into the edge table to identify the edges for each polygon.



Vertex Table
$V_1: x_1, y_1, z_1$
$V_2: x_2, y_2, z_2$
$V_3: x_3, y_3, z_3$
$V_4: x_4, y_4, z_4$
$V_5: x_5, y_5, z_5$

Edge Table
$E_1: V_1, V_2$
$E_2: V_2, V_3$
$E_3: V_3, V1$
$E_4: V_3, V_4$
$E_5: V_4, V_5$
$E_6: V_5, V_1$

Polygon- Surface Table
$S_1: E_1, E_2, E_3$
$S_2: E_3, E_4, E_5, E_6$

Figure 6.1: *Geometric data table representation for two adjacent polygon surfaces formed with six edges and five vertices.*

Listing the geometric data in three tables provides a convenient reference to the individual components (vertices, edges, and polygons) of each object.

2) Attribute table

This table contains attribute information that includes parameters that specify the degree of transparency of the object, its surface reflectivity, and texture characteristics.

Some tests (guidelines) to generate error free table:

i) Check if every vertex is listed as an endpoint for at least two edges.

ii) Check is every edge is part of at least one polygon.

iii) Check if each polygon is closed.

iv) Check if each polygon has at least one shared edge and

v) Check if the edge table contains pointers to polygons, every edge referenced by a polygon pointer has a reciprocal pointer back to be the polygon.

6.4 Plane Equations

To produce a display of three-dimensional object, we must process the input data representation for the object through several procedures. These processing steps include:

i. Transformation of the modeling and world coordinate descriptions to viewing coordinates, then to device coordinates.

ii. Identification of visible surfaces, and

iii. The application of surface rendering procedures.

For some of these processes, we need information about the spatial orientation of the individual surface components of the object.

This information is obtained from the vertex co-ordinate values and the equations that describe the polygon planes.

Equation for a plane surface can be expressed as

$Ax + By + Cz + D = 0$.....................(i)

where (x, y, z) is a point on the plane, and coefficients A,B,C and D are constant that describe the spatial properties of the plane.

We can obtain the values for A,B, C and D by solving a set of three plane equations using the coordinate values for three non-collinear points. We select 3 successive polygon vertices $(x_1, y_1,$

z_1), (x_2,y_2, z_2) and (x_3, y_3, z_3) and solve the following set of simultaneous linear plane equations for the ratios $\frac{A}{D}$, $\frac{B}{D}$, and $\frac{C}{D}$.

$$\frac{A}{D}x_1 + \frac{B}{D}y_1 + \frac{C}{D}z_1 = -1$$

$$\frac{A}{D}x_2 + \frac{B}{D}y_2 + \frac{C}{D}z_2 = -1$$

$$\frac{A}{D}x_3 + \frac{B}{D}y_3 + \frac{C}{D}z_3 = -1$$

Using Cramer's rule, we get

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

After expanding the determinants, we get,

$$A = y_1(z_2-z_3)+y_2(z_3-z_1)+y_3(z_1-z_2)$$

$$B = x_1(x_2-x_3)+z_2(x_3-x_1)+z_3(x_1-x_2)$$

$$C = x_1(y_2-y_3)+x_2(y_3-y_1)+x_3(y_1-y_2)$$

$$D = -x_1(y_2z_3-y_3z_2)-x_2(y_3z_1-y_1z_3)-x_3(y_1z_2-y_2z_1)$$

As vertex values and other information are entered into the polygon structure, values for A, B, C, D are computed for each polygon and stored with the other polygon data.

Orientation of a plane surface in space can be described with the normal vector to the plane.

The normal vector has Cartesian components (A, B, C) where A,B,C are the plane coefficients calculated above.

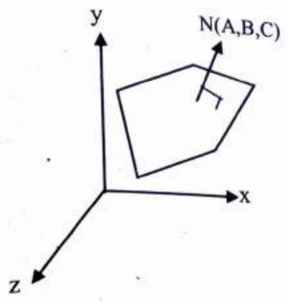


Figure 6.2: The vector N, normal to the surface of a plane described by the equation Ax + By +Cz +D=0, has Cartesian components (A, B, C)

Usually we deal with polygon surface that enclose the object interior in that case we need to distinguish between the "inside" and "outside" faces.

If polygon vertices are specified in a counter clockwise direction when viewing from the outer side of the plane, in a right-handed coordinate system, the direction of the normal vector will be from inside to outside.

To determine the components of the normal vector for the shaded surface, we select three of the four vertices along the boundary of the polygon, while viewing from outside in counter clockwise direction.

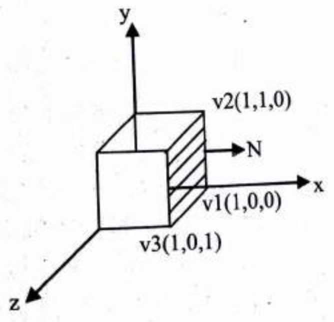


Figure 6.3: The shaded polygon surface of the unit cube has plane equation x - 1 = 0 and normal vector N = (1, 0, 0)

Now using the co-ordinate values for these vertices, we solve the equation for A, B, C and D are given.

$A = 1, B = 0, C = 0, D = -1$

This shows the normal vector is in positive x-direction, we can obtain the element of normal vector by calculating the vector cross product.

We again select any three vertices in counter clockwise direction while viewing from outside and calculate the normal vector as

$$\vec{N} = (\vec{v_2} - \vec{v_1}) \times (\vec{v_3} - \vec{v_1})$$

This will give us the direction of $\vec{N}$

For points not on the polygon surface

$Ax + By + Cz + D \neq 0$

So,

If $Ax + By + Cz + D < 0$, the point (x, y, z) is inside the surface.

If $Ax + By + Cz + D > 0$, the points (x, y, z) is outside the surface.

Plane can also be represented as $\vec{N}.\vec{P} = -D$, where $\vec{P}$ is a position vector of any given point on the plane.

Polygon meshes

Some graphics packages provide several polygon functions for modeling objects.

But when object surfaces are to be tiled, it is more convenient to specify the surface facets with a mesh functions.

A polygon mesh is a collection of vertices, edges and polygon connected in such a manner that at least two polygon share an edge and hence bounded the planner surface.

Common types

i. **Triangular mesh**

With n vertices, produce n-2 triangles

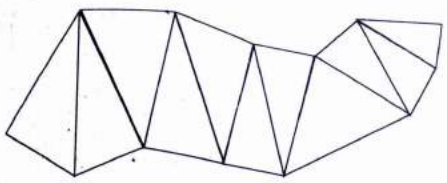


Figure 6.4: *A triangle strip formed with 11 triangles 13 vertices*

ii. **Quadrilateral mesh**

m = 5

n = 4

Quadrilaterals = (m–1)(n–1) = 12

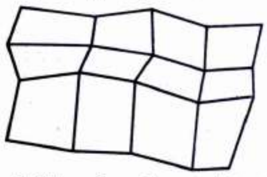


Figure 6.3: *A quadrilateral mesh containing 12 quadrilaterals constructed from a 5 by 4 input vertex array*

Chapter 7

Visible Surface Determination

7.1 Visible Surface Determination (Hidden Surface Elimination)

Visible surface determination is a process of identifying those parts of a scene that are visible from a chosen viewing position. There are numerous algorithms: some require more memory, some involve more processing time, and some apply only to special types of objects. The choice of a particular algorithm depends on factors like the complexity of the scene, type of objects to be displayed, available equipment, and whether static or animated displays are to be generated. Visible-surface detection algorithms are broadly classified into two categories:

i. **Object-space method**

   This method deals with object definitions directly. 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.

ii. **Image-space method**

   This method deals with the projected images of the objects. In this method, visibility is decided point by point at each pixel position on the projection plane. Most visible-surface algorithms use image-space methods.

7.2 Back-Face Detection

It is a fast and simple object-space method for identifying the back faces of a polyhedron (a solid in three dimensions with flat polygonal faces, straight edges and sharp corners or vertices) and is based on the "inside-outside" tests.

A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if

$$Ax + By + Cz + D < 0$$

When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and cannot see the front of it from our viewing position).

To simplify this test, consider the normal vector $\vec{N}$ to a polygon surface. If $\vec{V}$ is a vector in the viewing direction from the eye (or camera) position, then this polygon is a back face if

$$\vec{V}.\vec{N} > 0$$

If object description have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then $\vec{V} = (0, 0, V_z)$ and

$$\vec{V}.\vec{N} = VzC$$

so that we only need to consider the sign of C, the z component of the normal vector $\vec{N}$.

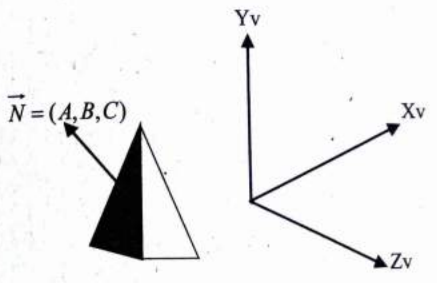


Figure 7.1: A polygon surface with plane parameter C < 0 in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z_v axis.

In a right-handed viewing system with viewing direction along the negative z_v axis, the polygon is a backface if C<0. Also, we cannot see any face whose normal has z component C = 0, since our viewing direction is grazing that polygon. Thus, in general, we can label any polygon as a backface if its normal vector has a z component value

$$C \le 0.$$

7.3 Depth-Buffer Method (Z-Buffer)

It is a commonly used image-space method for detecting visible surfaces. This method compares surface depth at each pixel position on the projection plane. This procedure is also called z-buffer method, as object depth is usually measured form the view plane along the z axis of a viewing system. Each surface of a scene is processed separately, one point at a time across the surface. This method is suitable for scenes containing only polygon surfaces. With object descriptions converted to projection coordinates, each (x, y, z) position on a polygon surface corresponds to the orthographic projection point (x, y) on the view plane. For each pixel position (x, y) on the view plane, object depths can be compared by comparing z values. Along the projection line from the position (x, y) in view plane taken as x_vy_v plane. Surface S_1 is closest at this position, so its surface intensity value at (x, y) is saved.

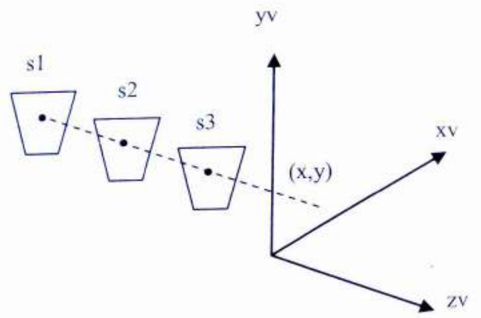


***Figure** 7.2: At view-plane position (x, y), surface S1, has the smallest depth from the view plane and so is visible at that position*

Depth-buffer method requires two buffer areas:

i. *Depth buffer* that stores depth values for each (x,y) position.

ii. *Refresh buffer* that stores the intensity values for each position.

Initially, all positions in the depth buffer are set to 0 (minimum depth) and the refresh buffer is initialized to background intensity. Each surface listed in the polygon table is then processed, one scan line at a time, calculating the depth (z value) at each (x, y) fixed position. The calculated depth is compared to the value previously stored in the depth buffer at that position. If the calculated depth is greater than the value stored in the depth buffer, the new depth is stored, and the surface intensity at that position is determined and placed in the same (x, y) location in the refresh buffer.

Depth values for a surface position (x, y) can be calculated from the plane equation for each surface as

$$z = \frac{-Ax - By - D}{C} \quad \text{...............(i)}$$

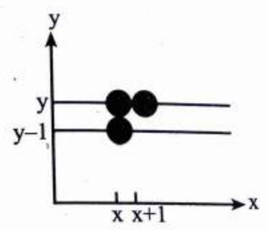


***Figure 7.3 :** From position (x, y) on a scan line, the next position across the line has coordinates (x+1, y), and position immediately below on the next line has coordinates (x, y-1)*

For any scan line, adjacent x and y values differs by 1. If the depth of position (x, y) has been determined to be z, then the depth z′ of the next position (x+1, y) along the scan line is obtained from equation (i) as

$$z' = \frac{-A(x+1) - By - D}{C} \quad \text{...............(ii)}$$

$$z' = z - \frac{A}{C} \quad \text{...............(iii)}$$

The ratio −A/C is constant for each surface, so succeeding depth values across a scan line are obtained from proceeding values with a single addition.

On each scan line, we start by calculating the depth on a left edge of the polygon that intersects that scan line. Depth values at each successive position across the scan line are then calculated by equation (iii).

We first determine the y-coordinate extents of each polygon and process the surface from the topmost scan line to the bottom scan line as shown in figure.

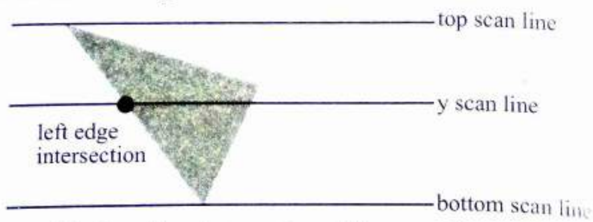


Figure 7.4: Scan lines intersecting a Polygen surfaces

Starting at a top vertex, we can recursively calculate x positions down a left edge of the polygon as x'= $x - \frac{1}{m}$, where m is the slope of the edge (see Figure 13.7)

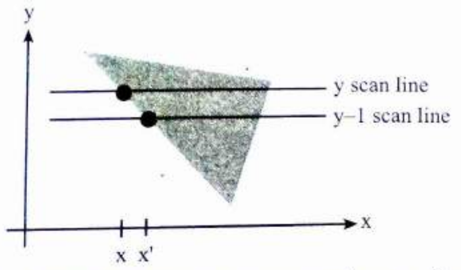


Figure 7.5: Intersection positions on successive scan lines along a left polygon edge.

Depth values down the edge are then obtained as

$$z' = \frac{-Ax' - By' - D}{C}$$

$$= \frac{-A(x - \frac{1}{m}) - B(y-1) - D}{C}$$

$$= \frac{-Ax - By - D}{C} + \frac{A(m+B)}{C}$$

$$= Z + \frac{A/m + B}{C}$$

For vertical edge, the slope is infinite, so $z' = z + \frac{B}{C}$

For polygon surfaces, depth-buffer method is very easy to implement and it requires no sorting of the surfaces in a scene. However, it requires second buffer (depth buffer) in addition to the refresh buffer. Another drawback is that depth-buffer method can only find one visible surface at each pixel position. That is, it deals with only opaque surfaces and cannot accumulate intensity values for more than one surface, as is necessary if transparent surfaces are to be displayed.

7.4 A-Buffer Method

This method is an extension of depth-buffer method. The A-buffer method represents an antialiased, area-averaged, accumulation-buffer method. The A-buffer method expands the depth-buffer so that each position in the buffer can reference a linked list of surfaces. Thus, more than one surface intensity can be taken into consideration at each pixel position, and object edges can be antialiased.

Each position in the A-buffer has two fields:

- **Depth field** – stores a positive or negative real number
- **Intensity field** – stores surface-intensity information or a pointer value.

If the depth field is positive, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RGB components of the surface color at that point.

If the depth field is negative, this indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. Data for each surface in the linked list includes

- RGB intensity components
- Opacity parameter (percent of transparency)
- Depth
- Percent of area coverage

- Surface identifier
- Other surface-rendering parameters
- Pointer to next surface

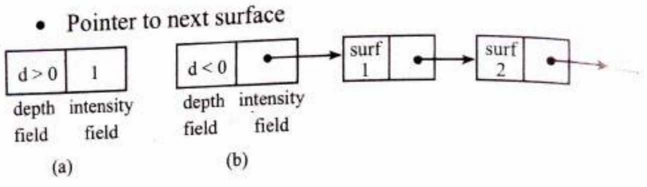


Figure 7.6: *Organization of an A-buffer pixel position: (a) single-surface overlap of the corresponding pixel area (b) multiple-surface overlap.*

7.5 Scan-Line Method

Scan-line method is the image-space method for removing multiple hidden surfaces. As each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are made for each overlapping surface to determine which as nearest to the view plane. When the visible surface is determined, the intensity value of that position is entered into the refresh buffer.

Different data structures or tables are set up for various surfaces which includes both an edge table and a polygon table.

i. **Edge table:** It contains coordinate endpoints of each line in the scene, inverse slope of each line, and pointers into the polygon table to identify the surfaces bounded by each line.

ii. **Surface table:** It contains coefficient of the plane equation for each surface, surface intensity information, and pointers to the edge table.

iii. **Active edge list:** It contains only edges that cross the current scan line, sorted in order of increasing x. It facilitates the search for surface crossing a given scan line.

iv. **Surface flag:** It indicates whether a position along a scan line is inside or outside of the surface. At the leftmost boundary of a surface, the surface flag is turned on and at the right most boundary it is turned off.

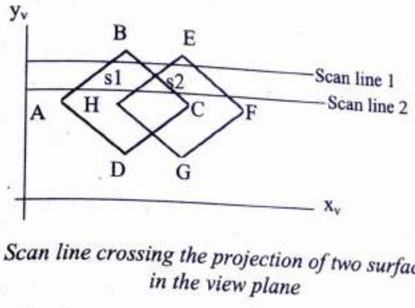


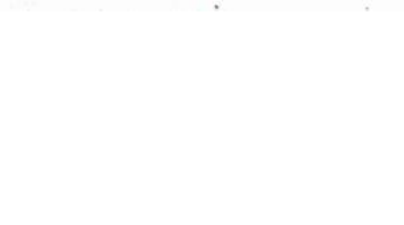
Figure 7.7: *Scan line crossing the projection of two surfaces, S1 and S2, in the view plane*

Active list for scan line 1 contains information from the edge table for edges AB, BC, EH and EF. For positions along this scan line between edges AB and BC only the flag for surface S_1 is on. So, on depth calculations are necessary and the intensity information for surface S_1 is entered from the polygon table into the refresh butter. Between edges EH and EF, only the flag for surface S_2 is entered into the refresh buffer, while for all other positions the intensity values are set to the background intensity. Similarly, active list for scan line 2 contains the edges AB, EH, BC and EF. Between edges AB and EH, only the flag for surface S_1 is set on and intensity value for S_1 is stored into refresh buffer.

Between edges EH and BC, the flags for both the surfaces S_1 and S_2 are set on. For this interval depth calculation must be alone (using the plane coefficients) and depending upon which surface is closer to the view plane, its intensity value is stored into the refresh buffer. Between edges BC and EF, the flag for surface S_1 goes off and the intensity value for surface S_2 is stored into the refresh buffer.

SOLVED NUMERICALS

1. Represent the following surfaces by polygon table method. Find the normal of surface S_1. *[2076 Ashwin Back]*

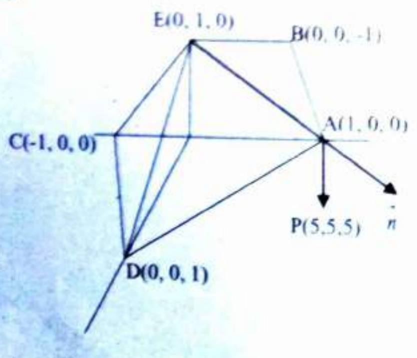


Vertex table	Edge table	Polygon-surface table
A: 1,2,3	e_1: A, B	S_1: e_1, e_2, e_3
B: 5,8,6	e_2: A, C	S_2: e_3, e_4, e_5, e_6
C: 2,5,8	e_3: C, B	
D: 8,−3,2	e_4: C, D	
E: 12,6,9	e_5: D, E	
	e_6: E, B	

We can find the normal of surface S by selecting any vertices in counter clockwise direction. While viewing outside

$$\vec{N} = \overrightarrow{AC} \times \overrightarrow{AB}$$
$$= [(2-1)i + (5-2)j + (8-3)k] \times [(5-1)i + (8-2)j + (6-3)k]$$
$$= (i + 3j + 5k) \times (4i + 6j + 3k)$$
$$= \begin{bmatrix} i & j & k \\ 1 & 3 & 5 \\ 4 & 6 & 3 \end{bmatrix}$$
$$= (9 - 30)i - (20 - 3)j + (6 - 12)k$$
$$= -27i - 17j - 6k$$

2. ***Find the visibility for the surface AED where observer at P(5,5,5)***



Solution:

Step 1:

Find the normal vector n for AED surface (always take anti-clockwise direction convention)

i.e. AE*AD, NOT AD*AE

AE = E − A

= (0 - 1)i + (0 - 0)j + (0 - 0)k

= -i + j

N = AE*AD

= (-i + j) × (-i + k)

$$\begin{bmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

-i(1 - 0) - j(-1 + 0) + k(0 + 1)

= i + j + k

Step 2:

The observer is at P(5,5,5) so we can construct the view vector V from surface to view point A(1,0,0) as

V = PA = (1 - 5)i + (0 - 5)j + (0 - 5)K

= -4i - 5j - 5k

Steps 3:

To find the visibility of the object, we use dot product of view vector and normal vector N as

$$\vec{V}.\vec{N} = (-4i - 5j - 5k).(i + j + k)$$
$$= -4 - 5 - 5$$
$$= -14 < 0$$

This shows that the surface is visible for observe

Chapter 8

Illumination and Surface Rendering Method

8.1 Illumination Models and Surface Rendering Technique

8.1.1 Illumination model/lighting model/shading model

Realistic displays of a scene are obtained by generating perspective projections of objects and by applying natural lighting effect to the visible surfaces.

Illumination model is used to calculate the intensity of light that we should see at a given point on the surface of an object. An illumination model (equation) expresses the components of light reflected from or transmitted (refracted) through a surface. There are three basic light components: ambient, diffuse and specular.

8.1.2 Surface rendering algorithms

Use the intensity calculations from an illumination model to determine the light intensity for all projected pixel positions for the various surfaces in a scene. Rendering can be performed by applying the illumination model to every visible surface point, or the rendering can be accomplished by interpolating intensities across the surfaces from a small set of illumination model calculations.

Illumination models involve number of factors like

i. Optical properties of the surfaces (transparency, reflectivity, surface texture)

ii. Relative positions of the surfaces in a scene.

iii. Color and position of the light sources and

iv. Position and orientation of the viewing plane.

8.2 Light Source

Total reflected light from an opaque non luminous object, is the sum of the contributions from light source and other reflecting surfaces in the scene. So, a surface that is not directly exposed to a light source may still be visible if nearby objects are illuminated. Light emitting sources are light bulbs, sun etc. Light reflecting sources are walls of a room, other reflecting surfaces etc. A luminous object, in general, can be both a light source and a light reflector, e.g. a plastic globe with a light bulb, etc.

1. **Point source**
   - Simplest model for a light emitter
   - Rays from the source then follow radially diverging paths from the source position

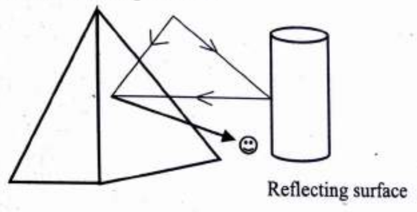
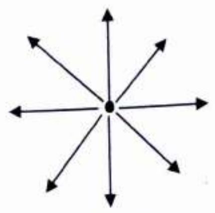
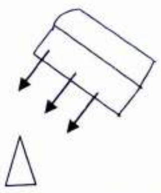


Figure 8.1: *Light viewed from an opaque non luminous surface is in general a combination of reflected light from a light source and reflections from other surfaces*

   - Reasonable approximation for sources whose dimensions are small compared to the size of objects in the scene. E.g. sun.

2. **Distributed light source**
   - Area of the source is not small compared to the surface in the scene.
   - A nearby source, such as the long fluorescent light is more accurately modeled as a distributed light source.


point source


distributed light source

Light reflection

When light is incident on an opaque surface, past of it is reflected and past is absorbed. Amount of incident light reflected by a surface depends on the type of nature. shining materials reflect more of the incident light, and dull surfaces absorb more of the incident light. Similarly, for an illuminated transparent surface some of the incident light will be reflected and some will be transmitted through the material.

1. **Diffuse reflection**

Diffuse reflection is the reflection of light from a surface such that a ray incident on the surface is scattered at many angles rather than at just one angle as in the case of specular reflection. An ideal diffuse reflecting surface is said to exhibit Lambertian reflection, meaning that there is equal luminance when viewed from all directions lying in the half-space adjacent to the surface.

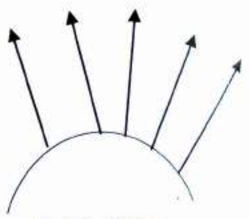


Fig 8.2: Diffuse reflection

2. **Specular reflection**

Specular or regular reflection, is the mirror-like reflection of waves, such as light, from a surface. In this process, each incident ray is reflected at the same angle to the surface normal as the incident ray, but on the opposing side of the surface normal in the plane formed by incident and reflected rays. The result is that an image reflected by the surface is reproduced in mirror-like (specular) fashion.

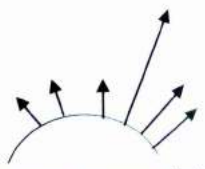


Fig 8.3: Specular reflection super imposed on diffuse reflections vector

The law of reflection states that for each incident ray the angle of incidence equals the angle of reflection, and the incident, normal, and reflected directions are coplanar.

8.3 Basic Illumination Models

Illumination model is method for calculating light intensities. Light calculations are based on the optical properties of the surface, the back ground lighting conditions and the light-source specifications. Optical parameters are used to set surface properties such as transparency, opacity etc, and these control the amount of reflection and absorption of incident light. All light sources are considered to be point source specified with a co-ordinate position an intensity value (color).

1. **Ambient light**

- A surface that is not directly exposed to a light source will still be visible if nearby objects are illuminated.
- It is non directional light source that is the product of multiple reflections from the surrounding environment.
- It is a basic illumination model, where we set a general level of brightness for a scene.
- It is a simple way to model the combination of light reflections from various surfaces to produce a uniform illumination called the ambient light or background light.
- Ambient light has no spatial or directional characteristics
- Amount of ambient light incident on each object is a constant for all surfaces and over all directions, but the intensity of the reflected light for each surface depends on the optical properties of the surface.
- So if I_a is the amount of ambient light incident on any surface, the ambient light reflection is given by ambient illumination equation,

$I = K_a * I_a$

We have K_a = ambient reflectivity or ambient reflection coefficient which ranges from 0 to 1. It is a material property.

2. **Diffuse reflection**

- Ambient light reflection is an approximation of global diffuse lighting effects.
- Diffuse reflections are constant over each surface in a scene, independent of viewing direction.
- Amount of incident light that is diffusively reflected is defined with a surface parameter K_d called diffuse-reflection coefficient on diffuse-reflectivity.
- K_d is assigned a constant value in the interval 0 to 1, according to the reflecting properties we want the surface to have.
- For highly reflecting surface, $K_d \rightarrow 1$ and for a very dull surface $K_d \rightarrow 0$

  If a surface is exposed only to ambient light, the intensity of diffuse reflection at any point the surface is

  $I_{ambdiff} = K_d * I_a$
- Ambient light produces flat uninteresting shading for each surface so scenes are rarely rendered with ambient light alone.
- At least one light source is included in a scene, often as a point source at the viewing position.

Ideal diffuse reflector

It scatters diffuse reflections from the surface with equal intensity in all directions. Since radiant light energy from any point on the surface is governed by Lambert's cosine law, ideal diffuse reflector is often known as Lambertian reflector. Lambert's cosine law states that the radial energy from any small surface area A in any direction N relative to the surface normal is proportional to $\cos\phi_N$ and the light intensity depends on the radial energy per projected area perpendicular to direction ϕ_N.

i.e. $I = dA \cos\phi_N$

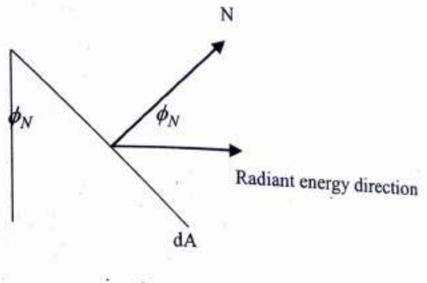


Fig 8.4: Radiant energy from a surface area dA in direction ϕ_N relative to the surface normal direction

Thus, for Lambertian reflection the intensity of light is the same over all viewing direction. Even though there is equal light scattering in all directions from a perfect diffuse reflector, the brightness of the surface does depend on the orientation of the surface relative to the light source.

Surface perpendicular to the direction of incident light appears brighter than the one with some oblique angle to the direction of the incoming light.

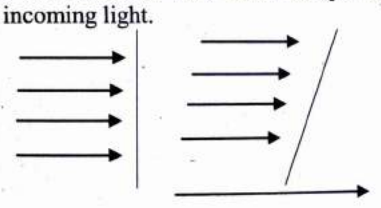


Fig 8.5: Specular reflection super imposed on diffuse reflections

If θ is angle of incidence between the incoming light direction and the surface normal then the projected area of a surface path perpendicular to light direction is proportional to cosθ

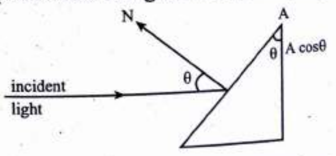


Fig 8.6: An illuminated area projected perpendicular to the path of the incoming light rays

Thus, the amount of illumination (or the number of incident light rays cutting the projected surface path) depends on cos θ.

If the incoming light from the source is perpendicular to the surface at a particular point, that point is fully illuminated.

As the angle of illumination moves away from the surface normal the brightness of the point drops off.

If I_l is the intensity of the point light source, then the diffuse reflection equation for a point on the surface is,

$I_{l,diff} = K_d I_l \cos\theta$.

A surface is illuminated by a point source only if the angle of incidence is in the range 0^0 to 90^0 for which cosθ is in the range 0 to 1. When cosθ is negative, the light source is behind the surface.

If $\vec{N}$ is the unit normal vector to a surface and $\vec{L}$ is the direction vector to the point source from a position on the surface then,

$\vec{N}.\vec{L} = \cos\theta$

$\cos\theta = \frac{\vec{N}.\vec{L}}{/\vec{N}/./\vec{L}/}$

and

$I_{l,diff} = K_d I_l(\vec{N}.\vec{L})$

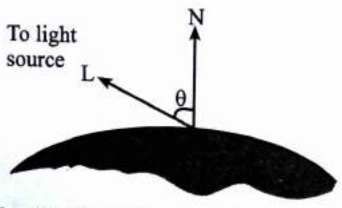


Figure 8.7: Angle of incident θ between the unit light-source direction vector $\vec{L}$ and the unit surface normal $\vec{N}$.

We can combine the ambient and point source intensity calculations to obtain an expression for the total diffuse reflection

$I_{diff} = K_a I_a + K_d I_l (\vec{N}.\vec{L})$

Whose both I_a and K_d depends upon surface material properties and are assigned values in the range from 0 to 1.

Specular Reflection and the Phong model

- Highlight or bright sports seen on shiny surfaces as a result of total or hear total reflection of the incident light in a concentrated region around the specular reflective angle.

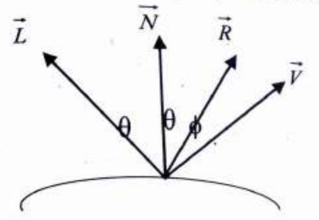


Figure 8.8: Specular-reflection angle equals angle of incident θ.

$\vec{L}$ = unit vector directed

$\vec{V}$ = unit vector pointing to the viewer

$\vec{R}$ = unit vector in the direction of ideal specula reflection

$\vec{N}$ = unit normal vector of the surface point ion.

- For an ideal reflector (perfect mirror), incident light is reflected only in the specula- reflection direction, i.e. $\vec{V}$ and $\vec{R}$ vectors coincide and $\phi = 0$
- Objects other than ideal reflectors exhibit specula reflections over a finite range of viewing positions around vector $\vec{R}$
- Shiny surfaces have a narrow specula-reflection range, where as dull surfaces have a wider reflection range.
- Phong model, developed by phong Bui-Tuong, is used to calculate the specula reflected range, which sets the intensity of specula reflection proportional to $\text{Cos}^{n_s}\phi$.
- ϕ can be assigned values in the range 0° to 90° (cosϕ= 0 to 1)
- n_s (specular- reflection parameter) is determined by the type of surface that we want to display.

- a very shiny surface is modeled with a large value for n, (say 100), and duller surface is assigned smaller values (say 1)
- for a perfect reflector, ns is infinite.
- Phong model calculate the specular reflection light intensity as

$I_{spec} = W(\theta)\, I_l \cos^{n_s}\phi$

Where I_l = incident light intensity

$W(\theta) = 1$ at $\theta = 90°$, and all of the incident light is reflected.

$W(\theta)$ = Specular-reflection coefficient

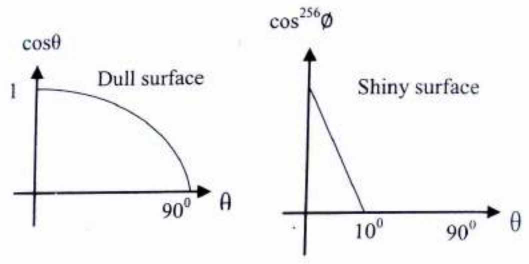


Figure 8.9: Plots of $\cos^{n_s}\phi$ for several values of specular parameter n_s

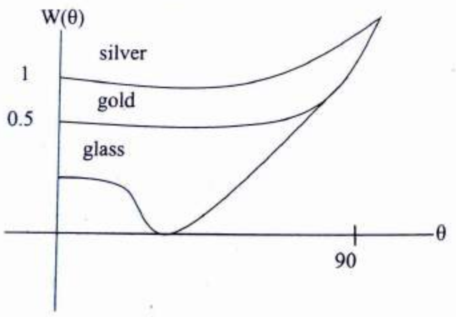


Figure 8.10: *Approximate variation of the specular-reflection coefficient as a function of angle of incidence for different materials.*

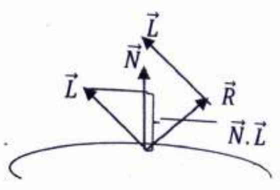


Figure 8.11: *Calculation of vector $\vec{R}$ by considering projections onto the direction of the normal vector $\vec{N}$.*

The projection of $\vec{L}$ onto the direction of the normal vector is obtained with the dot product $\vec{N}.\vec{L}$.

(The vector projection of $\vec{L}$ onto $\vec{N}$ is $\frac{\vec{N}.\vec{L}}{|\vec{N}|^2}\vec{N}$

The scalar projection $\vec{L}$ onto $\vec{N}$ is $\frac{\vec{N}.\vec{L}}{|\vec{N}|}$)

So,

$\vec{R} + \vec{L} = (2\vec{N}.\vec{L})\,\vec{N}$

$\vec{R} = (2\vec{N}.\vec{L})\,\vec{N} - \vec{L}$

- As seen from the figure for W(θ) vs θ, transparent materials, such as glass, only exhibit appreciate specula reflections as θ approaches 90°.
- But many opaque objects exhibits almost constant specular reflection for all incidence angles. For this case, we can replace W(θ), with a constant specular-reflection coefficient ks, whose value can be assigned in the range 0 to 1.

Also,

$\cos\phi = \frac{\vec{V}.\vec{R}}{|\vec{V}||\vec{R}|} = \vec{V}.\vec{R}$

so, $I_{spec} = K_s I_l (\vec{V}.\vec{R})^{n_s}$

Now,

Combined diffuse and specular reflection,

$I = I_{diff} + I_{spec}$

$= K_a I_a + k_d I_l (\vec{N}.\vec{L}) + k_s I_l (\vec{V}.\vec{R})^{n_s}$

If there are more than one light sources in the scene, then

$$I = K_aI_a + \sum_{i=1}^{n} I_{1i}[K_d(\vec{N}.\vec{L_i}) + K_s(\vec{V}.\vec{R_i})^{n_s}]$$

Intensity Attenuation

- As radiant energy from a point source travels through space, its amplitude is attenuated by the factor $1/d^2$, where d is the distance that the light has traveled.
- Which means a surface close to the light source (small d) receives a higher incident intensity from the source than a distant surface (large d).
- So, for realistic lighting effects, we should take into account the intensity attenuation, otherwise, it produces unrealistic effect as we will be illuminating all surfaces with the same intensity, no matter how for they might be from the light source.
- But simple point source illumination model does not always produce realistic pictures, if we use the factor $1/d^2$ to attenuate intensities, as it produces very little variation when d is large.
- Graphics package have compensated this problem by using universal quadratic attenuation function as the attenuation factor.

  $f(d) = 1/(a_0+a_1d+a_2d^2)$

  where d is the distance to the light source. And a, b and c are properties of the light . The numbers a, b, and c are called the "constant attenuation" , "linear attenuation", and " quadratic attenuation" of the light source. OpenGL1.1 supports attenuation. By default, a is one and b and c are zero, which means that there is no attenuation.
- Required effects can be obtained by varying the values of a_0, a_1, and a_2

  or

  $f_{att} = 1/(K_c+K_ld+K_qd^2)$

  d = distance between the light and the surface being shaded

  K_c = constant attenuation factor

K_l = Linear attenuation factor

K_q = quadratic attenuation factor

- Using the attenuation function
- $$I = K_aI_a + \sum_{i=1}^{n} f(d_i)I_{1i}[K_d(\vec{N}.\vec{L_i}) + K_s(\vec{V}.\vec{R_i})^{n_s}]$$

8.4 Surface Rendering Methods

The standard objects that are formed with polygon surfaces are rendered by the application of an illumination model.

1. Constant shading
2. Giraud shading
3. Phong shading
4. Fast Phong shading

1. Constant Shading

- Constant shading is also called flat shading
- It is fast and simple
- A single intensity is calculated for each polygon and all points over the surface of the polygon are then displayed with the same intensity value
- Useful for quickly displaying the general appearance of a curved surface.
- Provides an accurate rendering for an object if all of the following assumptions are valid
  - i. The object is a polyhedron and is not an approximation of an object with a curved surface.
  - ii. All light source illuminating the object are sufficiently far from the surface so that $\vec{N}.\vec{L}$ and the attenuation function are constant over the surface.
  - iii. The viewing position is constant over the surface.
- The sharp intensity discontinuation is seen in the border between two polygon.

2. Gouraud Shading (Intensity Interpolation Method)

- Renders a polygon surface by linearly interpolating intensity values across the surface.
- Intensity values for each polygon are matched with the values, of adjacent polygons along the common edges, thus eliminating the intensity dis continuation that can occur in flat shading.

- Produces more realistic results, but requires considerably more calculations.

Each polygon surface is rendered with Gouraud shading by performing the following calculations:

i. Determine the average unit vector at each polygon vertex

$$\vec{N}v = \frac{\sum_{k=1}^{n} \vec{N}_k}{\left|\sum_{k=1}^{n} \vec{N}_k\right|}$$

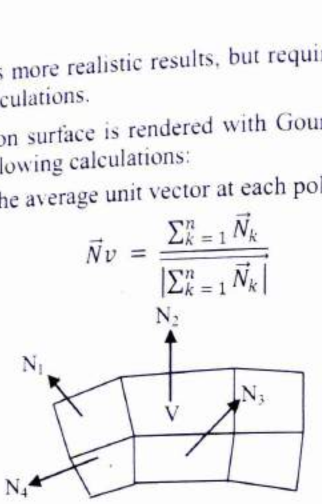


Figure 8.12: The normal vector at vertex V is calculated as the average of the surface normals for each polygon sharing that vertex.

ii. Apply an illumination model to each vertex to calculate the vertex intensity.

$I = K_aI_a + k_dI_l\,(\vec{N}.\vec{L}) + k_sI_l\,(\vec{V}.\vec{R})^{n_s}$

iii. Linearly interpolate the vertex intensities over the surface of the polygon.

For each scan line, the intensity at the intersection of the scan line with a polygon edge is linearly interpolated from the intensities at the edge endpoints.

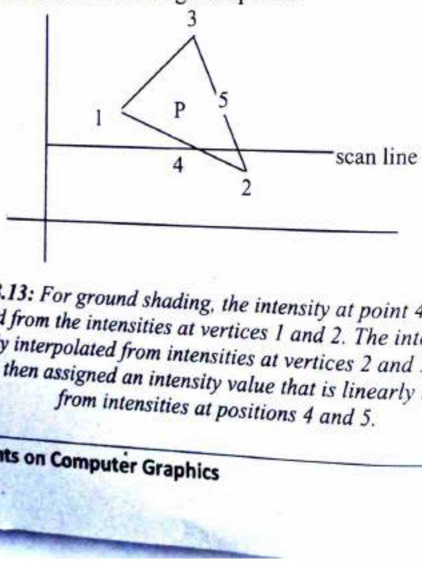


Figure 8.13: *For ground shading, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point P is then assigned an intensity value that is linearly interpolated from intensities at positions 4 and 5.*

As shown in figure, a scan line intersects two edge 12 and 23 now, obtain intensity value at any point P along the scan line, we do calculation in the following manner,

$I_4 = \frac{y4-y2}{y1-y2}*I_1 + \frac{y1-y4}{y1-y2}*I_2$

Similarly,

$I5 = \frac{y5-y3}{y2-y3}*I2 + \frac{y2-y5}{y2-y3}*I3$

then, $Ip = \frac{xp-x5}{x4-x5}*I4 + \frac{x4-xp}{x4-y5}*I5$

And, along the edge we make incremental calculations for intensity values

$I = \frac{y-y2}{y1-y2}*I_1 + \frac{y1-y}{y1-y2}*I_2$

$$\begin{aligned}
I' &= \frac{(y-1)-y_2}{y_1-y_2} \times I_1 + \frac{y_1-(y-1)}{y_1-y_2} \times I_2 \\
&= \frac{y-y_2-1}{y_1-y_2} \times I_1 + \frac{y_1-y+1}{y_1-y_2} \times I_2 \\
&= \frac{y-y_2}{y_1-y_2} \times I_1 - \frac{I_1}{y-y_2} + \frac{y_1-y}{y_1-y_2} \times I_2 + \frac{I_2}{y_1-y_2} \\
&= I - \frac{I_1}{y_1-y_2} + \frac{I_2}{y_1-y_2} \\
&= I + \frac{I_2-I_1}{y_1-y_2}
\end{aligned}$$

then, $I' = I + \frac{I2-I2}{y1-y2}$

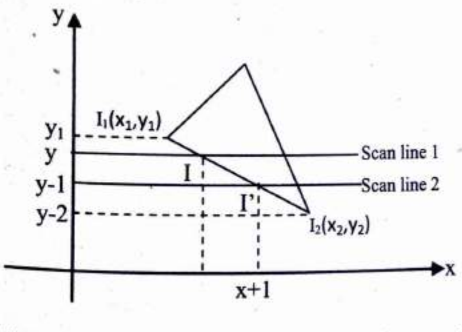


Figure 8.14: *Incremental interpolation of intensity values along polygon edge for successive scan lines.*

We make, similar calculations to obtain successive intensity values along horizontal line.

Advantages

- Removes the intensity discontinuities associated with the constant shading model.

Disadvantages

- Linear intensity interpolation can cause bright or dark intensity streaks, called mach bands, to appear on the surface. This effect can be reduced by increasing the number of polygons while representing the object.

3. **Phong shading (Normal Vector Interpolation shading):**
   - It is normal vector interpolation method
   - Greatly reduced mach band effect
   - More accurate interpolation method which interpolates the vector normal over the surface of the polygon.
   - Displays more realistic surface reducing mach band effect.

Steps:

i. Determine the average unit normal vector at each polygon vertex.

ii. Linearly interpolate the vertex normal's over the surface of the polygon

iii. Apply an illumination model along each scan line to calculate projected pixel intensities for the surface points.

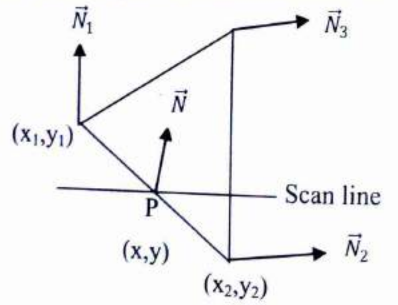


Figure 8.15: *Interpolation of surface normals along a polygon edge*

As shown in figure, intensity at point along an edge is calculated first by interpolating the normal vectors for the end points of the edge, and finally applying the illumination model.

$$\vec{N} = \frac{y-y2}{y1-y2} \times \vec{N}_1 + \frac{y1-y}{y1-y2} \times \vec{N}_2$$

Advantages

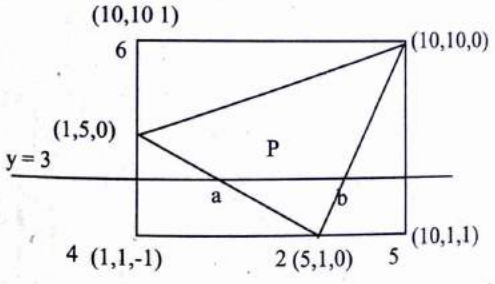
- Displays more realistic highlights in a surface.
- Reduce mach band effect.

Disadvantages

- Requires more computation than the Gouraud shading method.
- It is expensive rendering method.

SOLVED NUMERICALS

1. ***Find out the intensity of light reflected from the midpoint P on scan line y = 3 in the above given figure using Gouraud shading model. Consider a single point light source located at positive intensity on z-axis and assume vector to eye as (1,1,1). Given d = 0, k = 1, la = 1, l_l = 10, K_s = 2, K_a = K_d = 0.8 for use in a simple illumination model.***



Solution:

Step 1

Calculate unit normal vectors $\hat{N}_1$, $\hat{N}_2$, $\hat{N}_3$ at vertices 1, 2 and 3 respectively.

The normal vectors at the vertices can be approximated by averaging the cross product of all the edges that terminate at the vertices. It is important that the order of vectors should be so chosen that the cross product yields outward normal vectors only.

The normal vectors at 1,

$$\vec{N}_1' = V_1V_2 \times V_1V_3 + V_1V_3 \times V_1V_6 + V_1V_4 \times V_1V_2$$

$$= (4\hat{i}-4\hat{j})\times(9\hat{i}+5\hat{j}) + (9\hat{i}+5\hat{j}) \times (5\hat{j}+\hat{k}) + (-4\hat{j}-\hat{k}) \times (4\hat{i}-4\hat{j})$$

$$= \hat{i}-13\hat{j}+117\hat{k}$$

The unit normal at 1,

$$\hat{N}_1 = \frac{\vec{N}_1'}{|\vec{N}_1'|}$$

$$= \frac{\hat{i}-13\hat{j}-117\hat{k}}{\sqrt{1^2+(-13)^2+117^2}}$$

$$= 0.01\hat{i}+0.11\hat{j}+0.99\hat{k}$$

Similar at 2,

$$\vec{N}_2' = V_2V_3 \times V_2V_1 + V_2V_1 \times V_2V_4 + V_2V_5 \times V_2V_3$$

$$= 13\hat{i}+\hat{j}+117\hat{k}$$

$$\hat{N}_2 = \frac{\vec{N}_2'}{|\vec{N}_2'|}$$

$$= -0.11\hat{i}+0.001\hat{j}+0.99\hat{k}$$

Similarly at 3,

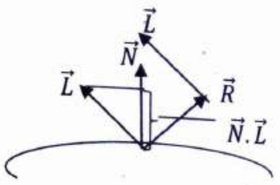
$$\vec{N}_3' = -4\hat{i}-4\hat{j}+146\hat{k}$$

$$\hat{N}_3 = \frac{\vec{N}_3'}{|\vec{N}_3'|}$$

$$= -0.03\hat{i}-0.03\hat{j}-0.99\hat{k}$$

Step 2,

Calculate unit reflection vectors $\vec{R}_1$, $\vec{R}_2$, $\vec{R}_3$ at vertices 1, 2, and 3 respectively



The projection of $\vec{L}$ onto the direction of the normal vector is obtained with the dot product $\vec{N}.\vec{L}$.

So,

$$\vec{R} + \vec{L} = (2\vec{N}.\vec{L})\vec{N}$$

$$\vec{R} = (2\vec{N}.\vec{L})\vec{N} - \vec{L}$$

$$\vec{R}_1 = 2(\vec{N}_1.\vec{L})\,\vec{N}_1 - \vec{L}$$

$$= 2((0.01\hat{i} - 0.11\hat{j} + 0.99\hat{k}).\hat{k})(0.01\hat{i} - 0.11\hat{j} + 0.99\hat{k}) - \hat{k}$$

$$= (0.02\hat{i} - 0.22\hat{j} + 0.96\hat{k})$$

$$\vec{R}_2 = 2(\vec{N}_2.\,\vec{L})\vec{N}_2 - \vec{L}$$

$$= (0.02\hat{i} - 0.22\hat{j} + 0.96\hat{k})$$

$$\vec{R}_3 = 2(\vec{N}_3.\vec{L})\,\vec{N}_3 - \vec{L}$$

$$= (-0.06\hat{i} - 0.06\hat{j} + 0.96\hat{k})$$

Step 3,

Calculate intensities I_1, I_2, I_3 at vertices 1, 2, and 3 respectively

$$I_1 = I_aK_a + \frac{Il(K_d(\vec{N}_1.\vec{L})+K_s)\vec{R}_1.V)^{ns})}{K+d}$$

$\vec{N}_1.\vec{L} = (0.01\hat{i} - 0.11\hat{j} + 0.99\hat{k}).\ \hat{k}$

$= 0.99$

$\vec{R}_1.\vec{V} = (0.02\hat{i} - 0.22\hat{j} + 0.96\hat{k}).(0.58\hat{i} + 0.58\hat{j} + 0.58\hat{k})$

$= 0.44$

$I_1 = (1)(0.10)+(10.1)((0.10)(0.99)+(0.80)(0.44)^2)$

$= 2.64$

$I_2 = I_aK_a + \frac{Il(Kd(N2L)+Ks)R2.V)^{ns)}}{K+d}$

$\vec{N}_2.\vec{L} = (0.01\hat{i} - 0.11\hat{j} + 0.99\hat{k}).\ \hat{k}$

$= 0.99$

$\vec{R}_2.\vec{V} = (0.02\hat{i} - 0.22\hat{j} + 0.96\hat{k}).(0.58\hat{i} + 0.58\hat{j} + 0.58\hat{k})$

$= 0.44$

$I_2 = (1)(0.10)+(10.1)((0.10)(0.99)+(0.80)(0.44)^2)$

$= 2.64$

Similarly, calculate $\vec{N}_3.\vec{L}$, $\vec{R}_3.\vec{V}$ and we get

$I_3 = 3.09$

Step 4

Interpolate intensities I_a, I_b and I_p at a, b, p respectively

Referring the figure,

The scan line y = 3 containing point p intersects the edges 1-2 and 3-2 respectively at a and b.

$$\frac{x_1 - x_a}{x_1 - x_2} = \frac{y_1 - y}{y_1 - y_2}$$

$$\frac{x_2 - x_b}{x_2 - x_3} = \frac{y_2 - y}{y_2 - y_3}$$

Using the slope of the edges the co-ordinates of a and b are found to be (3,3,0) and (6.11,3,0) respectively .

The coordinates of p, the midpoint of a b is found (4.56, 3,0). Now we have apply 3 stage interpolation technique to determine I_p

$$\frac{I_1 - I_a}{I_1 - I_2} = \frac{y_1 - y}{y_1 - y_2}$$

$I_a = 2.64$ $\quad I_b = 2.74$

$x_a = 3$ $\quad x_b = 6.11$

$x_p = 4.555$

$$\frac{I_a - I_p}{I_a - I_b} = \frac{x_a - x_p}{x_a - x_b}$$

$I_p = 2.69$

Chapter 9

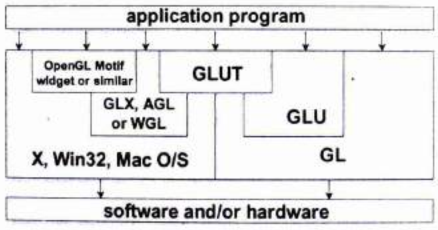
Introduction to OpenGL

9.1 Introduction

Open Graphics Library (OpenGL) is cross-language, cross-OS, cross-platform application programming interface (API) with large set of function to create and manipulate 2D and 3D graphics images. It is also known as graphics rendering API which generates high-quality color images composed of geometric and image primitives.

This interface consists of 250 distinct commands (200 in core OpenGL and 50 in OpenGL Utility Library) to produce interactive 3D applications.

OpenGL and Related APIs



- **AGL, GLX, WGL**

  It's glue between OpenGL and windowing systems

- **GLU (OpenGL Utility Library)**

  The OpenGL Utility Library (GLU) is a part of OpenGL. It contains several routines for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces.

- **GLUT (OpenGL Utility Toolkit)**

  GLUT an auxiliary library that provides several routines for opening windows, detecting input and creating complicated 3D objects like sphere, torus, and teapot.

9.2 OpenGL Libraries

A number of libraries exist like OpenGL, OpenGL Utility Library, OpenGL Utility Toolkit, and OpenGL Extension to the X Window System and Wrappers for X functions to simplify programming tasks,.

i. GL or OpenGL functions

OpenGL (gl or GL) command uses the prefix gl and initial capital letters for each word making up the command name. Some GL functions or commands are as follows:

- glClearColor(…);
- glBegin(…);
- glColor3f(…);
- glVertex3f(…);
- glEnd(…);
- glEnable(…);
- glDisable(…);

Constants:

OpenGL defined constants begin with GL_ and use all capital letters and underscores to separate words.

e.g. GL_COLOR_BUFFER_BIT

Data types

Data type	Corresponding C Language	OpenGL Type Definition
8 bit integer	signed char	GLbyte
16 bit integer	short	GLshort
32 bit integer	int or long	GLint, GLsizei
32 bit float	float	GLfloat
64 bit float	double	GLdouble

ii. GLU or OpenGL Utility Library

The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. Some GLU functions or commands are as follows:

- gluPerspective(…);
- gluLookAt(…);

iii. GLUT or OpenGL Utility Toolkit

The OpenGL Utility Toolkit (GLUT) is a window system independent toolkit written by Mark Kilgard to hide the complexities of differing window system APIs. OpenGL contains only rendering commands and no commands for opening windows or reading events from keyboard or mouse. GLUT provides several routines for opening windows, detecting input and creating complicated 3D objects like sphere, torus, and teapot. GLUT is an auxiliary library that makes easy to show the output of OpenGL application. It handles window creation, OS system calls like Mouse buttons, movement, keyboard, Callbacks etc.

GLUT routines use the prefix glut

```
glutInit(&argc, argv);
glutInitDisplayMode(…);
glutInitWindowPosition(…);
glutInitWindowSize(…);
glutCreateWindow(…);
glutDisplayFunc(…);
glutKeyboardFunc(…);
glutMouseFunc();
glutMainLoop(…);
```

9.3 OpenGL Program Structure

The basic OpenGL Programs have a similar structure as follows.

main()

- Define the callback functions
- Opens one or more windows with the required properties

- Enter event loops
- In main(), we should setup GL and GLUT stuff

Init ( )

It Initialize any OpenGL state and other program variable.

- Viewings
- Attributes

Callback Functions

Initialize the registered Callback functions

The basic OpenGL application structure is as follow

(i) Configure and open window and Initialize OpenGL state

We do this in main ()

E. g.,

```
/* OpenGL initialization code */
   glutInit(&argc, argv);
/* Specify the display Mode – RGB or color Index, single or
   double Buffer */
   glutInitDisplayMode(
   GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);
/* Create a window Named "simple GLUT Application"
   with starting point (100,100) with resolution 640 x 480 */
   glutInitWindowPosition(100, 100);
   glutInitWindowSize(640,480);
   glutCreateWindow("Simple GLUT Application");
```

(ii) Register input callback functions

We do this in main()

- render
- resize
- input: keyboard, mouse, etc.

E.g.,

```
/*Register the call back functions */
   glutDisplayFunc(render);
```

```
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);
```

(iii) The callback function code

We do this before main() function, E.g.,

```
void render(void)
 {
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFE
 R_BIT);
 glBegin(GL_TRIANGLES);
 glColor3f(1,0,0);
 glVertex2f(-0.5,-0.5);
 glColor3f(0,1,0);
 glVertex2f(0.5,-0.5);
 glColor3f(0,0,1);
 glVertex2f(0.0,0.5);
glEnd();
 }
void keyboard( unsigned char c, int x, int y)
 {
if(c = = 'a')
 {
exit(0);
 }
}
void mouse(int button, int state, int x, int y)
 {
if(button = = GLUT_RIGHT_BUTTON)
 {
exit(0);
 }
}
```

(iv) Enter event processing loop

We do this in main()

E. g.,

```
/*The program goes into an infinite loop waiting for events
   */
   glutMainLoop();
```

Simple OpenGL Program to draw a Triangle

```
/* program to draw a triangle*/
#include<windows.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <stdlib.h>
void render(void)
  {
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFE
   R_BIT);
  glBegin(GL_TRIANGLES);
  glColor3f(1,0,0);
  glVertex2f(-0.5,-0.5);
  glColor3f(0,1,0);
  glVertex2f(0.5,-0.5);
  glColor3f(0,0,1);
  glVertex2f(0.0,0.5);
glEnd();
  }
void keyboard( unsigned char c, int x, int y)
  {
if(c = = 'a')
  {
```

```
exit(0);
  }
}
void mouse(int button, int state, int x, int y)
  {
if(button = = GLUT_RIGHT_BUTTON)
  {
exit(0);
  }
}
int main( int argc, char* argv[])
{
/* OpenGL initialization code (Optional) */
   glutInit(&argc, argv);
/* Specify the display Mode – RGB or color Index, single or
   double Buffer */
   glutInitDisplayMode(
   GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);
/* Create a window Named "simple GLUT Application"
   with starting point (100,100) with resolution 640 x 480 */
   glutInitWindowPosition(100, 100);
   glutInitWindowSize(640,480);
   glutCreateWindow("Simple GLUT Application");

   /*Register the call back functions */
   glutDisplayFunc(render);
   glutKeyboardFunc(keyboard);
   glutMouseFunc(mouse);
/*The program goes into an infinite loop waiting for events
   */
   glutMainLoop();
}
```

9.3.1 Window Management

Five routines that perform necessary tasks to initialize a window are as follows:

- glutInit(int *argc, char **argv) or glutInit(&argc, argv) initializes GLUT and processes. glutInit() should be called before any other GLUT routine.
- glutInitDisplayMode(unsigned int mode), e.g., glutInit DisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RG BA) specifies whether to use an RGBA or color-index color model. We can also specify whether we want a single- or double-buffered window. (If we're working in color-index mode, we'll want to load certain colors into the color map; we use glutSetColor() to do this.) Finally, we can use this routine to indicate that we want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if we want a window with double buffering, the RGBA color model, and a depth buffer, we might call glutInitDisplay Mode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH).
- glutInitWindowPosition(int x, int y), e.g., glutInitWindow Position(100, 100) specifies the screen location for the upper-left corner of the window.
- glutInitWindowSize(int width, int size), eg., glutInitWindow Size(640,480) specifies the size, in pixels, of the window.
- int glutCreateWindow(char *string) or eg., glutCreate Window("Simple GLUT Application") creates a window with an OpenGL context. It returns a unique identifier for the new window. Until glutMainLoop() is called, the window is not yet displayed. The glutMainLoop() makes the program goes into an infinite loop waiting for events.

9.3.2 Callback Function

Callback function is user-defined function used to react on specific event like to redraw window, to react on keyboard, to handle mouse motions etc. We have to register callback function before to use it.

A callback function is a function which the library (GLUT) calls when it needs to know how to process something. E.g. when glut gets a key down event it uses the glutKeyboardFunc() callback routine to find out what to do with a key press.

The glutKeyboardFunc() deals with events generated by keys which have an ASCII code, for instance 'a', '1', or even ' '. glutSpecialFunc() deals with the "special keys", like F1-F12, Home, Up, etc.

The Display Callback function

- glutDisplayFunc(void (*func)(void)) or eg., glutDisplayFunc (render) is most important event callback function. Here function render is registered. We write the code in render() function for the graphics we want to display. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by glutDisplayFunc() is executed. Therefore, we should put all the routines we need to redraw the scene in the display callback function.
- If our program changes the contents of the window, sometimes we will have to call glutPostRedisplay(void), which gives glutMainLoop() a nudge to call the registered display callback at its next opportunity.
- We can register and call the function for action when any key is pressed or mouse button is pressed using glutKeyboardFunc() and glutMouseFunc() respectively. Allow us to link a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released.
- glutReshapeFunc(void (*func)(int w, int h)) indicates what action would be taken when the window is resized.
- **Managing a Background Process:** glutIdleFunc((*func)(void)) is used to specify a function that's to be executed if no other

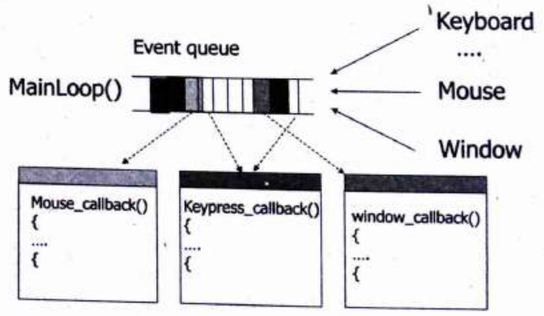
events are pending. This routine takes a pointer to a function as its only argument

GLUT Callback function

- **Event-driven:** Programs that use windows
  - Input/Output
  - Wait until an event happens and then execute some pre-defined functions according to the user's input
  - **Events** – key press, mouse button press and release, window resize, etc.
- **Callback function** : Routine to call when an event happens
  - Window resize or redraw
  - User input (mouse, keyboard)
  - Animation (render many frames)

  "Register" callbacks with GLUT
  - glutDisplayFunc(my_display_func);
  - glutIdleFunc(my_idle_func);
  - glutKeyboardFunc(my_key_events_func);
  - glutMouseFunc (my_mouse_events_func);

Event Queue


Keyboard
....
Event queue
MainLoop()
Mouse
Window
Mouse_callback() { .... {
Keypress_callback() { .... {
window_callback() { .... {

- glutKeyboardFunc() – register the callback that will be called when a key is pressed
- glutMouseFunc() – register the callback that will be called when a mouse button is pressed

- glutMotionFunc() – register the callback that will be called when the mouse is in motion while a buton is pressed
- glutIdleFunc() – register the callback that will be called when nothing is going on (no event)

Rendering Callback

- Callback function where all our drawing is done
- Every GLUT program must have a display callback
- glutDisplayFunc(***my_display_func***); /* this part is in main.c */

```
void my_display_func (void )
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_TRIANGLE);
        glVertex3fv( v[0] );
        glVertex3fv( v[1] );
        glVertex3fv( v[2] );
glEnd();
glFlush();
}
```

Idle Callback

- Use for animation and continuous update
  Can use *glutTimerFunc* or *timed callbacks* for animations
- glutIdleFunc(***idle***);

```
void idle( void )
{
        /* change something */
t += dt;
glutPostRedisplay();
}
```

User Input Callbacks

- Process user input
- glutKeyboardFunc(my_key_events);

```
void my_key_events (char key, int x, int y )
{
  switch ( key ) {
          case 'q' : case 'Q' :
            exit ( EXIT_SUCCESS);
            break;
          case 'r' : case 'R' :
          rotate = GL_TRUE;
            break;
  }
}
```

Mouse Callback

- Captures mouse press and release events
- glutMouseFunc(my_mouse);

```
void myMouse(int button, int state, int x, int y)
{if (button = = GLUT_LEFT_BUTTON && state = =
   GLUT_DOWN)
{...}
}
```

Events in OpenGL

Event	Example	OpenGL Callback Function
Keypress	KeyDown KeyUp	glutKeyboardFunc
Mouse	leftButtonDown leftButtonUp	glutMouseFunc
Motion	With mouse press Without	glutMotionFunc glutPassiveMotionFunc
Window	Moving Resizing	glutReshapeFunc
System	Idle Timer	glutIdleFunc glutTimerFunc
Software	What to draw	glutDisplayFunc

9.3.3 Running the Program

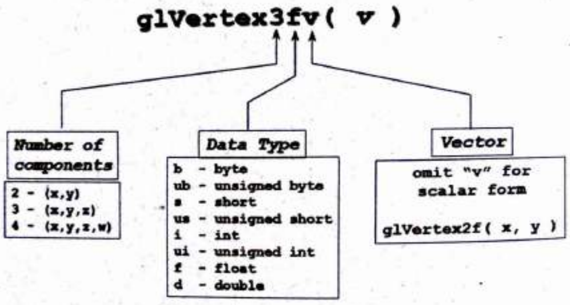
- At the last we call glutMainLoop(void). The glutMainLoop() makes the program goes into an infinite loop waiting for events. All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once this loop is entered, it is never exited!

9.4 Open GL Geometric Primitives

The geometry is specified by vertices. There are different primitive types:

GL_POINTS
GL_LINES
GL_TRIANGLES
GL_POLYGONS

OpenGL Command Format



9.4.1 Vertices and Primitives

Primitives are specified using

glBegin(*primType*);

...

glEnd();

➤ *primType* determines how vertices are combined

GLfloat red, green, blue;

Glfloat coords[nVerts][3];

*/*Initialize coords and colors somewhere in program*/*

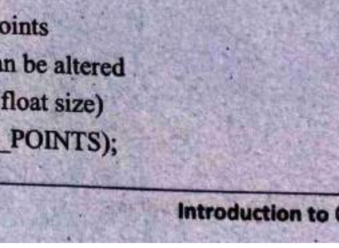
glBegin(*primType*);

for(i = 0; i < nVerts; ++i) {
  glColor3f(red, green, blue);
  glVertex3fv(coords[i]);
}

glEnd();

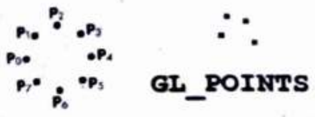
An Example to draw a Parallelogram

```
void drawParallelogram( GLfloat color[] )
{
glBegin( GL_QUADS );
glColor3fv( color );
glVertex2f( 0.0, 0.0 );
glVertex2f( 1.0, 0.0 );
glVertex2f( 1.5, 1.118 );
glVertex2f( 0.5, 1.118 );
glEnd();
}
```



i. **Points, GL_POINTS**
   - ➤ Individual points
   - ➤ Point size can be altered
   - ■ glPointSize(float size)
     glBegin(GL_POINTS);

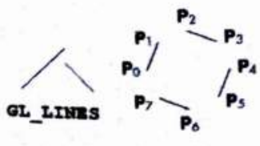
```
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



ii. **Lines, GL_LINES**

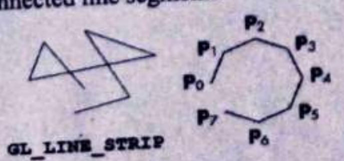
- Pairs of vertices interpreted as individual line segments
- Can specify line width using:
- glLineWidth(float width)

```
glBegin(GL_LINES);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



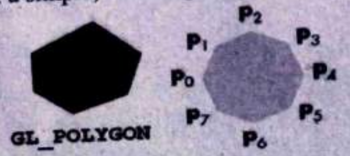
iii. **Line Strip, GL_LINE_STRIP**

- series of connected line segments



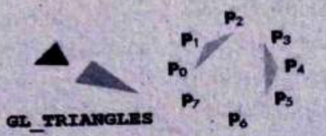
iv. **Polygon ,GL_POLYGON**

- boundary of a simple, convex polygon



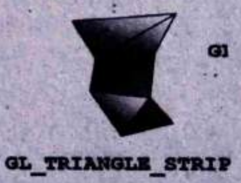
v. **Triangles ,GL_TRIANGLES**

- triples of vertices interpreted as triangles



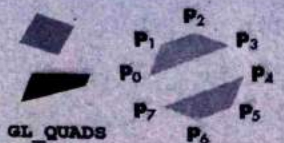
vi. **Triangle Strip ,GL_TRIANGLE_STRIP**

- linked strip of triangles



vii. **Quads ,GL_QUADS**

- quadruples of vertices interpreted as four-sided polygons



While writing program between glBegin and glEnd, those opengl commands are allowed:

- glVertex*(): set vertex coordinates
- glColor*() : set current color
- glIndex*() : set current color index
- glNormal*() : set normal vector coordinates (Light.)
- glTexCoord*() : set texture coordinates (Texture)

9.5 Color Command

OpenGL supports both RGBA and color index mode. In general, an OpenGL programmer first sets the color or coloring scheme and then draws the objects. Until the color or coloring scheme is changed, all objects are drawn in that color or using that coloring scheme.

To set a color, use the command glColor3f(). It takes three parameters, all of which are floating-point numbers between 0.0 and 1.0. The parameters are, in order, the red, green, and blue components of the color. You can think of these three values as specifying a "mix" of colors: 0.0 means don't use any of that component, and 1.0 means use all you can of that component. Thus, the code

glColor3f(1.0, 0.0, 0.0); makes the brightest red the system can draw, with no green or blue components. All zeros makes black; in contrast, all ones makes white. Setting all three components to 0.5 yields gray (halfway between black and white). Here are eight commands and the colors they would set.

glColor3f(0.0, 0.0, 0.0);	black
glColor3f(1.0, 0.0, 0.0);	red
glColor3f(0.0, 1.0, 0.0);	green
glColor3f(1.0, 1.0, 0.0);	yellow
glColor3f(0.0, 0.0, 1.0);	blue
glColor3f(1.0, 0.0, 1.0);	magenta
glColor3f(0.0, 1.0, 1.0);	cyan
glColor3f(1.0, 1.0, 1.0);	white

glClearColor(), takes four parameters, the first three of which match the parameters for glColor3f(). The fourth parameter is the alpha value. For now, set the fourth parameter of glClearColor() to 0.0, which is its default value.

For example, the pseudocode

```
set_current_color(red);
draw_object(A);
draw_object(B);
set_current_color(green);
set_current_color(blue);
draw_object(C);
```

The above code draws objects A and B in red, and object C in blue. The command on the fourth line that sets the current color to green is wasted.

OpenGL Program to draw a Polygon

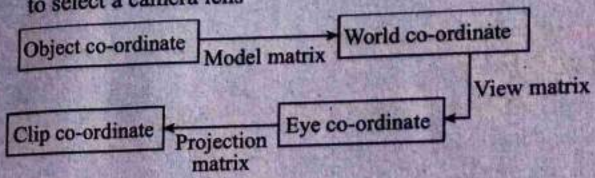
```
#include <GL/gl.h>
#include <GL/glut.h>

void display(void)
{ /* clear all pixels */
glClear (GL_COLOR_BUFFER_BIT);
/* draw white polygon (rectangle) with corners at * (0.25,
   0.25, 0.0) and (0.75, 0.75,0.0) */
glColor3f (1.0, 1.0, 1.0);
glBegin(GL_POLYGON);
glVertex3f (0.25, 0.25, 0.0);
glVertex3f (0.75, 0.25, 0.0);
glVertex3f (0.75, 0.75, 0.0);
glVertex3f (0.25, 0.75, 0.0);
glEnd();
/* don't wait! * start processing buffered OpenGL routines
   */
glFlush ();
}
void init (void)
{ /* select clearing (background) color      */
glClearColor (0.0, 0.0, 0.0, 0.0);
/* initialize viewing values */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0); }
/*
* Declare initial window size, position, and display mode
* (single buffer and RGBA). Open window with "hello"
* in its title bar. Call initialization routines.
* Register callback function to display graphics.
* Enter main loop and process events. */
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (250, 250);
glutInitWindowPosition (100, 100);
glutCreateWindow ("hello");
init ();
glutDisplayFunc(display);
glutMainLoop();
return 0; /* ISO C requires main to return int. */
}
```

9.6 OpenGL viewing

- In OpenGL the model – View matrix is used to position the camera (i.e. viewing), it can be done by rotations and translation but is easier to use gluLookAt ( ), and to build models of objects.

- The projection matrix is used to define the view volume and to select a camera lens



Model matrix

The model matrix transforms a position in a model to the position in the world. This position is affected by the position, scale and rotation of the model that is being drawn. If the vertices in world coordinates (common when drawing a simple test scene) is specified already, then this matrix can simply be set to the identity matrix.

View matrix

In real life to alter the view of a certain scene we can move camera, in OpenGL it's the other way around. The camera in OpenGL cannot move and is defined to be located at (0,0,0) facing the negative Z direction. That means that instead of moving and rotating the camera, the world is moved and rotated around the camera to construct the appropriate view.

In older versions of OpenGL we use *Model View* and *Projection* transformations. The ModelView matrix combines the model and view transformations into one. It is easier to separate the two, so the view transformation can be modified independently of the model matrix.

That means to simulate a camera transformation; we actually have to transform the world with the inverse of that transformation. If we want to move the camera up, you have to move the world down instead.

Projection matrix

After the world has been aligned with our camera using the view transformation, the projection transformation can be applied, resulting in the clip coordinates. If we're doing a perspective transformation, these clip coordinates are not ready to be used as normalized device coordinates just yet.

Transformation command

glMatrixMode( )

- It specifies the model view, projection or texture matrix modification passing the argument GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE
- Only one matrix can be modified at a time

glLoadIdentity ( ),

- It sets the currently modifiable matrix to identity matrix.

Viewing transformation

- gluLookAt (eye.x, eye.y, eye.z, center.x, center.y, center.z, up.x, up.y, up.z)
- The argument of this command indicates where the camera (or eye position) is placed, where it is aimed and which way is up.
- Viewing direction is center to eye.
- Up is the upward direction.
- Viewing direction and up vector indicates eye co-ordinate system.

  x-axis points to the right of viewer.

  y-axis points to upward.

  z-axis points to the back of viewer.

Viewport transformation

- The viewport transformation in openGL is controlled by fucntion glviewport (GLint x, GLint y, GLintsizei width, GLintsizei height)
- It is used to set the size and position of the viewport.
- The (x, y) defines the lower left corner of the viewport and width and height are the size of the viewport rectangle.

The Viewing Transformation example

The viewing transformation is analogous to positioning and aiming a camera. In this code example, before the viewing transformation can be specified, the current matrix is set to the identity matrix with glLoadIdentity(). This step is necessary since most of the transformation commands multiply the current matrix by the specified matrix and then set the result to be the current matrix. If you don't clear the current matrix by loading it with the identity matrix, you continue to combine previous transformation matrices with the new one you supply.

In the following example, after the matrix is initialized, the viewing transformation is specified with gluLookAt(). The arguments for this command indicate where the camera (or eye position) is placed, where it is aimed, and which way is up. The arguments used here place the camera at (0, 0, 5), aim the camera lens towards (0, 0, 0), and specify the up-vector as (0, 1, 0). The up-vector defines a unique orientation for the camera.

If gluLookAt() was not called, the camera has a default position and orientation. By default, the camera is situated at the origin, points down the negative z-axis, and has an up-vector of (0, 1, 0). So in this example, the overall effect is that gluLookAt() moves the camera 5 units along the z-axis.

OpenGL program to draw a Transformed Cube:

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

voidinit(void)
{  glClearColor (0.0, 0.0, 0.0, 0.0);
glShadeModel (GL_FLAT);
}
void display(void)
{  glClear (GL_COLOR_BUFFER_BIT);
glColor3f (1.0, 1.0, 1.0);
glLoadIdentity ();          /* clear the matrix */
/* viewing transformation */
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glScalef (1.0, 2.0, 1.0);    /* modeling transformation */
glutWireCube (1.0);
glFlush ();
}
void reshape (int w, int h)
{
glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
glMatrixMode (GL_MODELVIEW);
}
int main(intargc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();
return 0;
}
```

Q. Transformation function in open GL

i. Translation
   glTranslatef (tx, ty, tz),
   where tx, ty, tz are translation co-ordinate
ii. Rotation
   glRotatef(A, x, y, z),
   where A is angle of rotation
iii. Scaling
   glscalef (float x, float y, float z)

9.7 Lighting in OpenGL

To describe light in an OpenGL application, first, we have to enable the lighting system by using glEnable(GL_LIGHTING) and glEnable(GL_LIGHTn) then we need to perform two steps: set the lighting and the shading models.

```
// Enable the lighting system
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

// Create light components
Glfloat ambientLight[] = { 0.2f, 0.2f, 0.2f, 1.0f };
Glfloat diffuseLight[] = { 0.8f, 0.8f, 0.8, 1.0f };
Glfloat specularLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };
Glfloat position[] = { -1.5f, 1.0f, -4.0f, 1.0f };

// Assign created components to GL_LIGHT0
glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

Defining light source

OpenGL allows a maximum of 8 light sources in a scene. First of all, we have to enable the lighting system on the whole by calling glEnable(GL_LIGHTING) and glEnable(GL_LIGHTn); where n is the index number of the color you are enabling, ranging from 0 to 7. It should be obvious that to specify a light source number 1 we should use GL_LIGHT0. By order, the light source number 8 is specified as GL_LIGHT7.

glDisable(int *cap*) function disables whatever properties were previously set with glEnable.

To set a specific light type and enable it in 3D scene, for each one of the light sources we need to call the glLightfv function with parameters which specify the what, and more importantly the how. To add a component of specular light to a light source, you would make the following function call:

GLfloat specular[] = {1.0f, 1.0f, 1.0f , 1.0f};

glLightfv(GL_LIGHT0, GL_SPECULAR, specular);

GL_LIGHT0 is light source and the specular shading model is described by the GL_SPECULAR parameter and the GLfloat specular parameter defines properties of the specular reflection that are setting up for the GL_LIGHT0 light source. specular[] is a 4-parameter array. The parameters are described as:

specular[] = { floatRed, floatGreen, floatBlue, floatAlpha };

The first three parameters are the RGB values which can range from anywhere between 0.0f and 1.0f. 0.0f being no color, and 1.0f being full color. The fourth parameter 'floatAlpha' in the array is used for an EMISSIVE light component to modify an object's material's alpha value.

The same glLightfv mechanism can be used to specify any of the other three shading models. For example, to set and enable the Ambient Light component of a light source so that it emits Ambient Light of moderately pale white color (R=0.5f, G=0.5f, B=0.5f) we can use the following call and parameters:

GLfloat ambient[] = { 1.0f, 1.0f, 1.0f };

glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);

We also have to specify the position of the light source. And this can be done, similarly to defining light components, with the glLight function in the following way:

Glfloat position[] = { -1.5f, 1.0f, -4.0f, 1.0f };

glLightfv(GL_LIGHT0, GL_POSITION, position);

The lighting model

The lighting model is set up with a call to glLightModel. The visual behavior of the lighting model is specified by the function glLightModel. There are two types of this function. One that uses scalar values as parameters and one for use with vector valued as parameters. Here are the definitions of both functions:

glLightModelf(GLenum pname, GLfloat param); // scalar params

glLightModelfv(GLenum pname, const GLfloat *params); // vector params

To define a light source, in OpenGL for Ambient Light, Diffuse Light, Specular Light and Emissive Light, OpenGL uses four models: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR and GL_EMISSIVE respectively.

For enabling the global ambient light model for the whole scene GLOBAL_AMBIENT_LIGHT_MODEL is used, which tells OpenGL that we want a global ambient light model set. The color of the global ambient light is set as:

Glfloat global_ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f };

glLightModelfv(GL_LIGHT_MODEL_AMBIENT,global_ambient);

The shading model

After setting up light model the next step is usually setting up the shading model. This is done by calling glShadeModel. Here, we use the smooth shading model glShadeModel(GL_SMOOTH);

After this call, all of the polygons will be smoothly shaded by using the Gouraud-shading technique and according to the nearby light sources and polygon's material properties.

The shading model is set up with a call to glShadeModel and can be either set to SMOOTH or FLAT model. The SMOOTH shading model specifies the use of Gouraud-shaded polygons to describe light while the FLAT shading model specifies the use of single-colored polygons.

glShadeModel(int mode) selects the polygon shading model. mode is the flag representing the shading mode. This flag can be set to either GL_FLAT or GL_SMOOTH.

GL_SMOOTH shading is the default shading model, causes the computed colors of vertices to be interpolated as the primitive is rasterized, assigning different colors to each resulting pixel fragment. GL_FLAT shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.

In either case, the computed color of a vertex is the result of lighting, if lighting is enabled, or it is the current color at the time the vertex was specified, if lighting is disabled.

The lighting model of OpenGL is based on the Gouraud Shading implementation. A specific color is assigned to each of the vertices in a polygon. This color is calculated according to the object's material properties and surrounding light sources. Then the colors at each of the 3 vertices are taken and interpolated across the whole polygon.

❖❖❖

Bibliography:

1. Donald D. Hearn and M. Pauline Baker, "Computer Graphics", (Second Edition)
2. Foley, Van Dam, Feiner, Hughes "Computer Graphics Principles and Practice", (Second Edition in C)
3. Udit Agrawal, "COMPUTER GRAPHICS", (Fourth Edition)
4. Rajiv Chopra, "COMPUTER GRAPHICS(A PRACTICAL APPROACH)", (Third Edition)
5. Neeta Nain, "COMPUTER GRAPHICS", (First Edition)
6. G. S. Baluja, "COMPUTER GRAPHICS", (Revised Edition 2008)

Lab Plan

Subject Teacher: Shree Krishna Sulu	**Subject: Computer Graphics**	Full Mark:50
Year/Part: III/I	**Program: Computer and ELX.**	Pass mark:20

SN	**Title**	**Objectives**	**Lab**	**Requirements**
1	Introduction to graphics mode, library functions, initializing graphics mode	After the completion of Lab: Students would have the concept of graphics mode and initialize it	Computer Lab	Turbo C
2	DDA Algorithm	After the completion of Lab: Students would be able to draw lines using DDA algorithm in C language	Computer Lab	Turbo C
3.	Bresenham's Algorithm	After the completion of Lab: Student would be able to draw a line using Bresenham's algorithm	Computer Lab	Turbo C
4	Mid-Point Circle Algorithm	After the completion of Lab: Student would be able to draw a mid point circle using the algorithms	Computer Lab	Turbo C
5	Mid-Point Ellipse Algorithm	After the completion of Lab: Student would be able to draw a mid point ellipse using the algorithms	Computer Lab	Turbo C
6	Two Dimensional Transformations	After the completion of Lab: Students would be able to program about two dimensional algorithm	Computer Lab	Turbo C
7.	Basic Drawing Technique in OpenGL	After the completion of Lab: Students would be able to draw lines, polygons in OpenGL	Computer Lab	Code Block

'Silence' and 'smile' are two powerful words. Smile' is the way to solve many problems and 'silence' is the way to avoid many problems.