Lab on Computer Graphics

DCAP313

LOVELY PROFESSIONAL UNIVERSITY

LOVELY
PROFESSIONAL
UNIVERSITY

LAB ON COMPUTER GRAPHICS

Produced & Printed by
EXCEL BOOKS PRIVATE LIMITED
A-45, Naraina, Phase-I,
New Delhi-110028
for
Lovely Professional University
Phagwara

SYLLABUS

Lab on Computer Graphics

Objectives:

- To enable the student to understand various technicalities of hardware devices used in computer graphics.
- To enable the student to learn various techniques used to draw images.
- To enable the student to understand two dimensional and three dimensional operations on objects.
- To enable the student to understand technicalities to draw polygons.
- To enable the student to learn graphics programming.

Sr. No.	Topics
1.	**Fundamentals of Computer Graphics:** Applications of computer graphics in various fields, evolution of computer graphics.
2.	**Graphics Systems:** Video Display Unit, Random Scan Displays, raster scan displays, Displaying Colours, Frame Buffer, Digitization, Persistence, Resolution
3.	Implementing Line Algorithm
4.	Implementing Circle Algorithm
5.	Implementing Ellipse Algorithm
6.	Implementing polygon filling algorithm
7.	Implementation of Hidden Surface in 2D
8.	Implementing of Scaling in 2D Transformation
9.	Translation
10.	Shearing, Rotation, Reflection

CONTENTS

Unit 1: Fundamentals of Computer Graphics

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the concept of computer graphics
- Discuss the types of computer graphics
- Elaborate applications of computer graphics in different fields
- Explain the evolution of computer graphics over the time

Introduction

Computer graphics is a wonderful invention in the field of Computers. As per William A. Fetter "Perhaps the best way to define Computer graphics is to find out what it is not. It is not a group of computer programs. It is not the know-how of a graphic designer, a programmer, a writer, a motion picture specialist, or a reproduction specialist. Computer graphics is all these "a consciously managed and documented technology directed toward communicating information accurately and descriptively." Computer graphics find its use in diverse areas such as producing television commercials and feature films, simulation and analysis of real world problems, computer aided design, graphical user interfaces that increases the communication bandwidth between humans and machines, etc. The purpose of this Unit is to introduce the fundamentals of Computer graphics to the students. In this Unit, you will learn about concept of computer graphics, its applications and evolution of computer graphics over the time.

1.1 Concept of Computer Graphics

Computer graphics is concerned with all aspects of producing images using a computer. It comprises first creating simple graphical elements and then representing them.

The most common use of computer graphics is in the creation of an image which looks like a photograph but portrays something which we cannot really capture in a picture. This type of computer graphics is known as photo-realistic.

Example: An animation movie showing an alien is photo-realistic approach to computer graphics.

Another common use of computer graphics is in the form of real-time or interactive process.

Example: Computer games which require real-time action by the user and based on that interaction different action takes place in the game. Generally, a user can grab images the rate of 24 images per second. This means for portraying a real-time action by images one need to generally pass on 24 images (or frames) per second.

Pixel

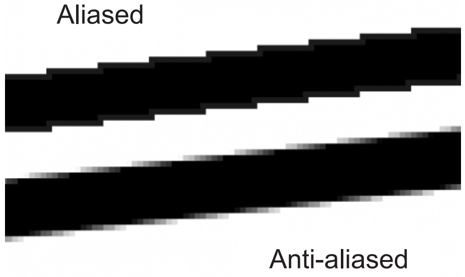
A computer image is usually represented as a discrete grid of picture elements a.k.a. pixels. The number of pixels determines the resolution of the image. Typical resolutions range from 320*200 to 2000*1500. For a black and white image, a number describes the intensity of each pixel. It can be expressed between 0.0 (black) and 1.0 (white).

For a color image, each pixel is described by a triple of numbers representing the intensity of red, green and blue.

Example: Pure red is (255, 0, 0) and purple is (255, 0, 255).

Because the image is represented by a discrete array of pixels, aliasing problems may occur. The most classical form of aliasing is the jaggy aspect of lines.

Figure: 1.1: Alias and Anti-alias Problem in Images



Source: http://i.stack.imgur.com/pA7uy.png

1.1.1 Types of Computer Graphics

Based on our discussion above with examples we can divide Computer graphics into two types:

1. ***Non-interactive Computer Graphics:*** It refers to the computer graphics where the observer has no control over the image and their movements. Familiar example of this type is cartoons shown on TV.

2. ***Interactive Computer Graphics:*** It refers to a two way communication between computer and user. Here, the user is given some control over the image and its movement which he does with an input device like a computer mouse. Using the input device, the user sends signal to the system based on which the images change their action.

Example: In shooting the hen game when the user press the mouse button at the hen it appears to him that he has killed the hen.

Notes Though non-interactive and interactive computer graphics both have their own merits but it is interactive computer graphics that finds its much usage these days and is expected to increase in near future.

1.1.2 Basic Components of Interactive Computer Graphics

An interactive Computer Graphic System consists of three components:

- ***Input Device:*** This refers to the devices used to input data or give signal to the system.

  Example: mouse, scanner, jockey, etc.

- ***Processing and Storage:*** This refers to the components that does processing work as well as stores the data.

  Example: Processor and memory.

- ***Display/Output:*** This refers to the component on which the output is displayed after the processing.

  Example: Monitor/Screen, Printer, etc.

Did u know? Sketchpad, first truly interactive graphics system was developed by Ivan Sutherland at MIT for his 1963 Ph.D. thesis.

Self Assessment

State whether the following statements are true or false:

1. Computer graphics is concerned with all aspects of producing images using a computer.
2. The most common use of computer graphics is in the creation of a song.
3. Computer video game is a type of non-interactive computer graphics.
4. Monitor is an example of display component of a computer graphic system.

Notes

1.2 Application of Computer Graphics

Computer Graphics find their use in various fields which are given below:

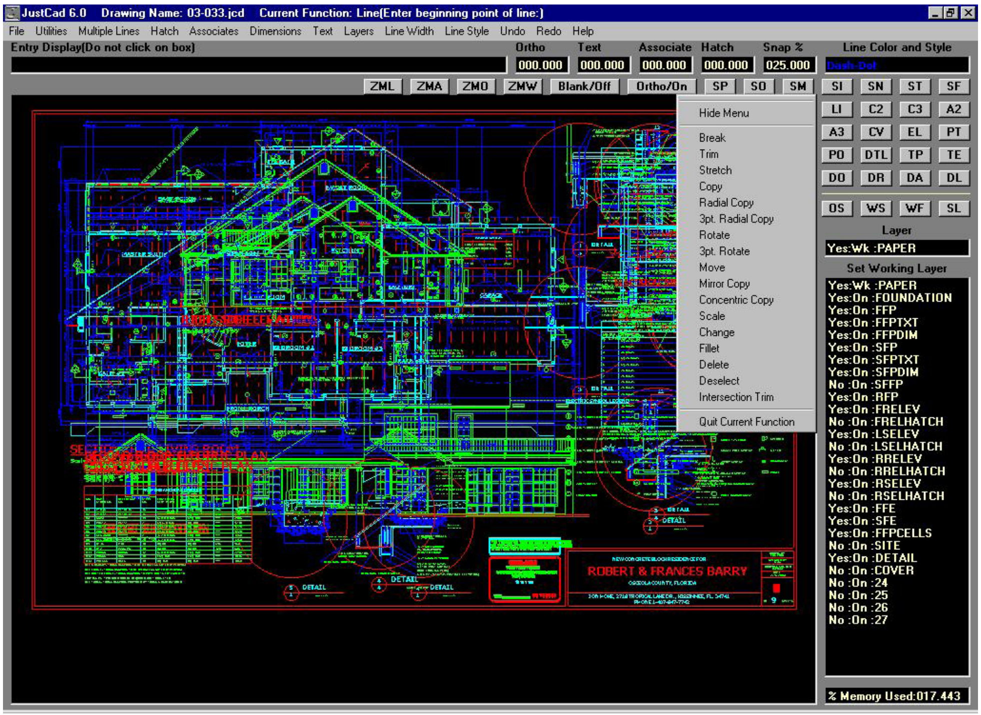
- Computer Aided Design (CAD)
- Image Processing
- Graphical User Interface (GUI)
- Presentation Graphics
- Computer Art
- Entertainment
- Education and training

1.2.1 Computer Aided Design (CAD)

Computer graphics are majorly used in design processes, particularly for engineering and architectural system. For some of the design applications, objects are first displayed in a Wireframe (outline) form. Wireframe (outline) form shows the overall shape and internal features of objects.

Software package for CAD applications like JustCad generally provide the designer with a multi-window environment as shown in Figure 1.2. Circuits are designed by placing the components into the layout and then the graphics package automatically provides the connections between components.

Figure 1.2: Multi-window Environment of JustCad



JustCad 6.0   Drawing Name: 03-033.jcd   Current Function: Line(Enter beginning point of line:)

Entry Display(Do not click on box)

Ortho	Text	Associate	Hatch	Snap %	Line Color and Style
000.000	000.000	000.000	000.000	025.000	Dash-Dot

Layer

Yes:Wk :PAPER

Set Working Layer

```
Yes:Wk :PAPER
Yes:On :FOUNDATION
Yes:On :FFP
Yes:On :FFPTXT
Yes:On :FFPDIM
Yes:On :SFP
Yes:On :SFPTXT
Yes:On :SFPDIM
No :On :SFFP
Yes:On :RFP
Yes:On :FRELEV
No :On :FRELHATCH
Yes:On :LSELEV
No :On :LSELHATCH
Yes:On :RRELEV
No :On :RRELHATCH
Yes:On :RSELEV
No :On :RSELHATCH
Yes:On :FFE
Yes:On :SFE
Yes:On :FFPCELLS
No :On :SITE
Yes:On :DETAIL
No :On :COVER
No :On :24
No :On :25
No :On :26
No :On :27
```

% Memory Used:017.443

Source: http://www.solidswiki.com/images/e/e9/Cad_software.jpg

CAD methods are used in the design of Building, Automobiles, Aircraft, and Textiles etc.

1.2.2 Image Processing

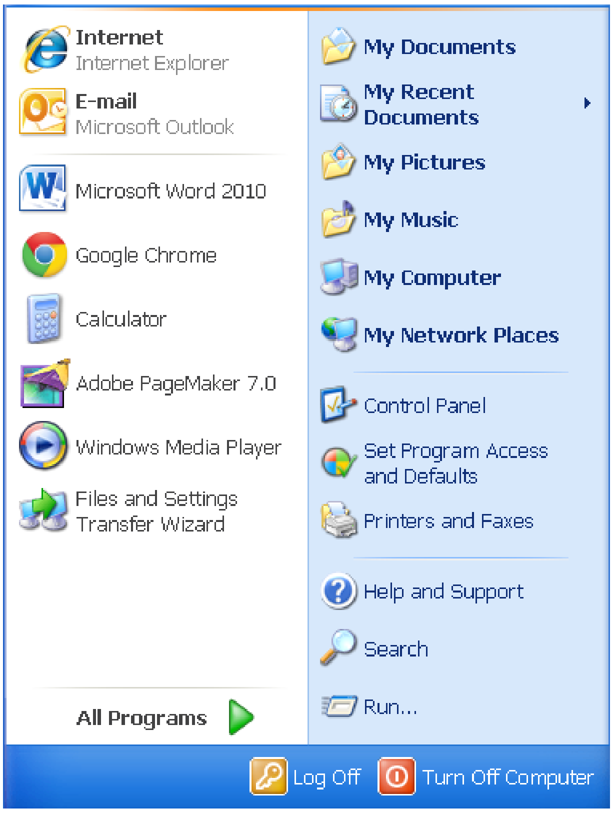
Image processing refers to the techniques used to modify existing pictures. Interpreting an existing image is also a type of image processing. Image processing can be any of the following:

- Image enhancement: It deals with sharpening of image features such as boundaries or contrast to make a graphic display much better than the original one.
- Image restoration: It deals with minimizing the effect of degradations from the existing image.
- Image compression: It deals with minimizing the number of bits required to represent an image without degrading the quality of the image to an unacceptable level.

1.2.3 Graphical User Interface

A Graphical User Interface (GUI) usually allows users to interact with electronic devices using images rather than text commands. It also displays menus and icons for fast processing. Common example of GUI is your Windows Operating System.

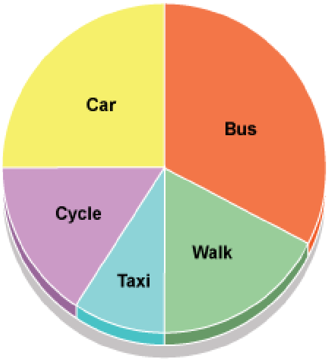
Figure 1.3: Windows Graphical User Interface



1.2.4 Presentation Graphics

Presentation graphics are used to generate 35-mm slides or transparent sheets for use with projectors. In presentation graphics we generally use Bar Graphs, Line Graphs and Pie Charts. Figure 1.4 shows a pie chart.

Figure 1.4: A Pie Chart Example



Source: http://www.bbc.co.uk/bitesize/ks3/maths/images/pie_chart1.gif

1.2.5 Computer Art

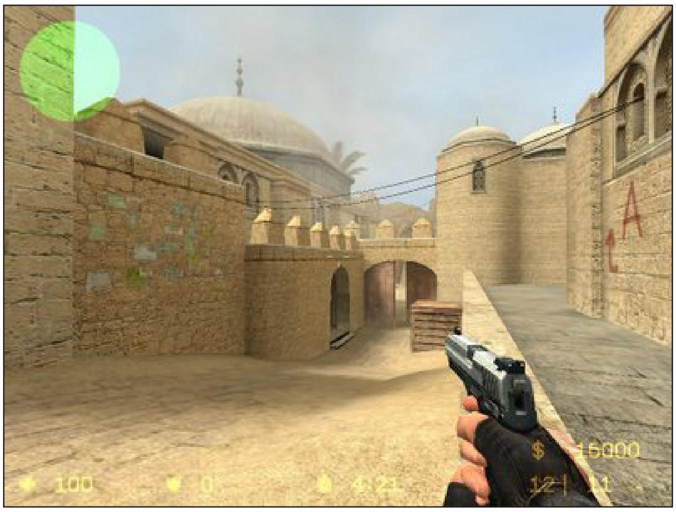
Computer graphics are widely used in fine art and commercial art applications. In fine art, using paintbrush an artist paints pictures on the screen of a video monitor. Commercial art refers to activities like creation of logos, page layout, morphing, etc.

Example: The first movie to employ detailed morphing was Willow, in 1988. A similar process was used a year later in Indiana Jones and the Last Crusade to create Walter Donovan's gruesome demise. Both effects were created by Industrial Light & Magic using grid warping techniques developed by Tom Brigham and Doug Smythe (AMPAS).

1.2.6 Entertainment

Computer graphics are commonly used in making motion pictures, games and television shows. Figure 1.5 shows a screenshot of a video game which depicts use of computer graphics.

Figure 1.5: Screenshot of a Video Game

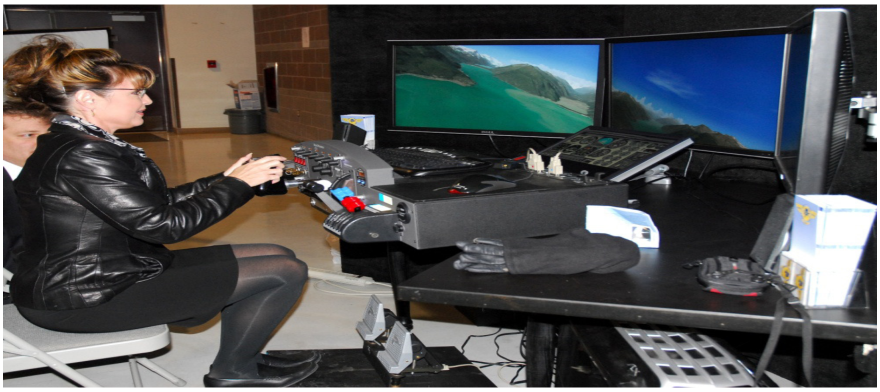


Source: http://3.bp.blogspot.com/_OzwVAnut5gA/SejmPlVuaUI/AAAAAAAAAMU/k1T0H jNxLY8/s400/Foto CounterStrikeSource.jpg

1.2.7 Education and Training

Certain systems are used to help trainees understand the operation of the system. Simulators are a kind of special system which is used for training of professionals like pilot, ship captains, etc. Figure 1.6 shows a Flight Simulator.

Figure 1.6: Flight Simulator



Source: http://upload.wikimedia.org/wikipedia/commons/d/df/Sarah_Palin_Flight_Simulator.jpg

Task Make a list of at least three important applications of computer graphics.

Self Assessment

Fill in the blanks:

5. For some of the design applications, objects are first displayed in a .... form.

6. Image processing refers to the techniques used to .... existing pictures.

7. A Graphical User Interface (GUI) usually allows users to interact with electronic devices using .... rather than .... commands.

8. .... are used to generate 35-mm slides or transparent sheets for use with projectors.

1.3 The Evolution of Computer Graphics

Computer graphics is a wonderful invention in the field of Computers. As we saw earlier in the unit it is used in diverse areas. For example, we no longer need to depend on the time consuming punching cards or the difficult commands.

Caution Computer graphics might be taken for granted today but to be at this stage it has been a long and painful struggle because hardware rarely was keeping up with the demand for better images.

Computer graphics is application oriented. First application was Sketchpad by Ivan Sutherland. It allowed user to draw on the screen. Then came the concept of paint systems which made

Notes

the basic communication between human and computer more pictorial. This lead to people's attention towards Word processing, Desktop publishing, Computer-aided design and simulation of real world problems over the time. Computer games got more popular later on with use of extensive graphics in them.

1.3.1 From Blinking Lights to Plotters

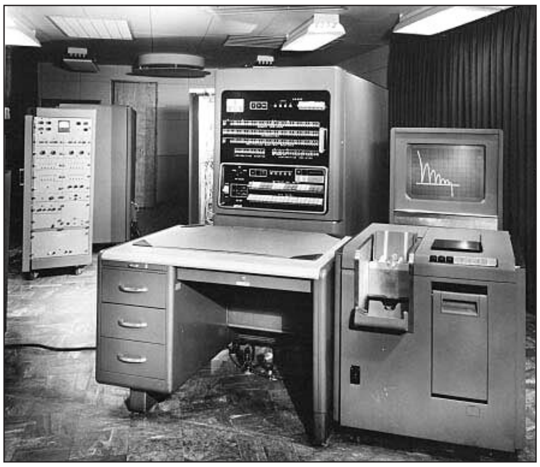
In the early days of computing, there were existing devices which could translate a simple binary pattern into text. Thus, getting the computer to type was not that much difficult task.

Example: The military used teletype machines for many years. Earlier computers mostly used flashing lights with punched cards for input and output. Generally, when there is scope of only a few hundred instructions, you take input and output in its simplest form.

But sometimes, with time and the need to do something new becomes an inventor of new technology. Same happened when the need to produce a picture was raised. Then need increased from simple image to an image through which a user could interact with system.

IBM offered an output printer on its 701 model in 1952. It also offered a primitive graphics solution (the model 740 "Cathode Ray Tube Output Recorder") in 1954. Figure 1.7 shows IBM model 740.

Figure 1.7: IBM Model 740



Source: http://cdn.arstechnica.net/01-19-2011/ibm_model_740.jpg

The Figure 1.7 shows how big the demand for graphics was that time. The IBM Model 740 was a cathode ray tube to which a camera could be attached. Digital-to-analog converters used to drive the cathode ray tube to slowly draw the lines based on the digital outputs of the computer. This method was later called as "vector graphics".

Line plotting takes place one point at a time. IBM had bragged that points were plotted at a rate of 8,000 per second, with a display accuracy of a given point of only 3%, but with good repeatability. Typically, the camera shutter was opened when the drawing started, and closed when it finished. At that time, the film could be developed, and the image could be viewed later the same day.

From the above description we can realize that this technology was not suitable for playing video games.

The IBM Model 740 also had another similar model 780, which had a comparatively long-persistence phosphor (20 seconds). Though it was not as precise as 740, but it allowed the operator to verify the image desirability. Importance of this can be analyzed when one has to wait several hours for the film image to be developed.

Gene Seid and Robert Morton, two of the founders of CalComp, developed an idea in 1953, that there is another way to get an image with a slow computer and that is using a plotter. But lack of funding kept the device off the market until 1959.

The idea of plotter is simple: drive a pen on two axes. That takes a pair of stepping motors, and something to put the pen down at the start of a line, and lift it again at the end. Software can calculate when the pen should be stepped in either axis to draw straight lines between two points, curved lines, or whatever.

With time, a better software packages was developed which allowed users to describe the image in more human-friendly terms such as "a=1, plot y=ax+b for x=1 to 5," and so on. If the plotter was used to draw a shape, it could be filled with solid, dashed, or dotted lines to simulate black or grayscale. With time, plotters had additional pens added to do drawings in color. But the number of colors was limited, and the drawings were still vectors.

Self Assessment

State whether the following statements are true or false:

9. IBM bragged that points were plotted at a rate of 5,000 per second.
10. Computer graphics is object oriented.
11. Earlier computers mostly used flashing lights with punched cards for input and output.
12. Line plotting was not suitable for playing video games.
13. IBM Model 780 produce more desirable image than IBM Mode 740.
14. Idea of plotter was developed by Gene Seid and Robert Morton.
15. Plotter works on a very complex principle.

1.4 Summary

- Computer graphics is concerned with all aspects of producing images using a computer. It comprises first creating simple graphical elements and then representing them.
- The most common use of computer graphics is in the creation of an image which looks like a photograph but portrays something which we cannot really capture in a picture. This type of computer graphics is known as photo-realistic.
- Computer graphics are divided into two types: Non-interactive Computer Graphics refer to the computer graphics where the observer has no control over the image and their movements, and Interactive Computer Graphics refer to a two way communication between computer and user.
- Computer Graphics find their use in various fields which are given below:
  - Computer Aided Design (CAD)
  - Image Processing
  - Graphical User Interface (GUI)
  - Presentation Graphics
  - Computer Art
  - Entertainment
  - Education and training

Notes

- Computer graphics is application oriented. First application Sketchpad allowed user to draw on the screen. Then came the concept of paint systems which made the basic communication between human and computer more pictorial.
- Earlier computers mostly used flashing lights with punched cards for input and output. IBM offered an output printer on its 701 model in 1952.
- Gene Seid and Robert Morton, two of the founders of CalComp, developed idea of plotter in 1953, but lack of funding kept the device off the market until 1959. With time, a better software packages was developed which allowed users to describe the image in more human-friendly terms such as "a=1, plot y=ax+b for x=1 to 5," and so on. Later on, plotters had additional pens added to do drawings in color. But the number of colors was limited, and the drawings were still vectors.

1.5 Keywords

Graphical User Interface: A system which usually allows users to interact with electronic devices using images rather than text commands.

Image Compression: It deals with minimizing the number of bits required to represent an image without degrading the quality of the image to an unacceptable level.

Image Enhancement: It deals with sharpening of image features such as boundaries or contrast to make a graphic display much better than the original one.

Image Restoration: It deals with minimizing the effect of degradations from the existing image.

Interactive Computer Graphics: It refers to a two way communication between computer and user.

Non-interactive Computer Graphics: It refers to the computer graphics where the observer has no control over the image and their movements.

Photo-realistic: It refers to portray something which we cannot really capture in a picture.

1.6 Review Questions

1. What are two types of computer graphics?
2. What are basic components of Interactive Computer Graphics?
3. Explain use of computer graphics in image processing field.
4. Discuss the role computer graphic plays in computer-aided design.
5. Why was line plotting not suitable for playing video games?
6. Write short note on evolution of computer graphics over the years.

Answers: Self Assessment

1.	True	2.	False
3.	False	4.	True
5.	Wireframe (outline)	6.	Modify
7.	Images, text	8.	Presentation graphics
9.	False	10.	False

Notes

11. True

12. True

13. True

14. True

15. False

1.7 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. "*Computer Graphics*" Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Angel, Edward. "*Interactive Computer Graphics*" Pearson Education.

Krishnamurthy, N. "*Introduction to Computer Graphics*" Tata McGraw Hill.

Online links

http://cs.lmu.edu/~ray/notes/graphicsintro/

http://ecomputernotes.com/computer-graphics/basic-of-computer-graphics/introduction-to-computer-graphics

https://users.cs.jmu.edu/bernstdh/web/common/lectures/slides_graphics-basics.php

http://graphicssoft.about.com/od/aboutgraphics/Learn_About_Computer_Graphics.htm

Unit 2: Graphics Systems

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss computer graphics system
- Explain video display devices
- Describe vector scan/random scan display
- Explain raster scan systems
- Define resolution
- Discuss about the colour displays
- Explain frame buffer

Introduction

Computers have become a powerful tool for the rapid and economical production of pictures and models. Computer graphics is a wonderful invention in the field of Computers. It is used in diverse areas such as displaying the results of engineering and scientific computations and visualization, producing television commercials and feature films, simulation and analysis of real world problems, computer aided design, graphical user interfaces that increases the communication bandwidth between humans and machines, etc. The art of creating pictures with a computer has got numerous applications, that it is of great importance to explore the intrinsic of the world of computer graphics. The advent of CRTs brought about a major change in the world of computers. No longer do we have to depend upon the awkward and time consuming punching cards or the cryptic commands. There is no area in which graphics displays cannot be used to some advantage. In this unit, you will learn about computer graphics systems. As the unit progress, different video display devices will be discussed. Lastly, the topics like resolution, color display and frame buffer will be introduced.

2.1 Computer Graphics System

Computer graphics are graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software. Interactive computer graphics involves some capability for user interaction. Although there are graphic applications in which user interaction is minimal or unnecessary, such as final rendering of an animation.

Did u know? Many programming languages provide libraries for creating GUI's. When combined with a display window, these devices can be an effective way to interact with a graphical system.

Though the term computer graphics generally is used in a broad sense to describe almost everything on computers that is not text or sound but it actually refers to:

- representation and manipulation of image by a computer
- a field of computer science that studies methods for digitally synthesizing and manipulating visual content

Computer generated images can be categorized into three different types:

- 2D (two dimensional) computer graphics is the computer-bottomed generation of digital images mostly from two-dimensional models (such as 2D geometric models) and by techniques specific to them.
- 3D (three dimensional) computer graphics are graphics that use a three-dimensional representation of geometric data (often Cartesian) that is stored in the computer for the purposes of performing calculations and rendering 2D images. With time, 3D computer graphics have become more common, but 2D computer graphics have not vanished.
- Computer animation refers to the art of creating moving images with the use of computers.

If we talk about Interactive Systems, a picture may take the form of a static image or that of an animated image. In any case there needs to be an interaction between the user and the system.

Caution For interaction to take place, the system needs to arrange tools to facilitate input and output devices.

Example: Computer Graphics Devices:

- CRT, VGA/SVGA monitors
- Flat panel devices, video digitizers, LCD panels
- Plotters, laser printers
- Keyboard, joystick, mouse, scanners
- Touch screen, track ball, etc.

Notes

Self Assessment

Fill in the blanks:

1. .... computer graphics is the computer-bottomed generation of digital images mostly from two-dimensional models and by techniques specific to them.
2. .... computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images.
3. .... refers to the art of creating moving images with the use of computers.

2.2 Video Display Devices

We mentioned different examples of Computer Graphic device above. Different types of display devices are:

- Raster (or Refresh) Scan Displays
- Random Scan or Calligraphic Displays
- Color CRT Monitors
- Direct View Storage Tube (DVST).
- Flat panel Displays
- Three Dimensional Viewing Devices
- Stereoscopic and Virtual Reality System
- Computer Graphics Display Devices

Did u know? The display systems are often referred to as Video Monitor Unit (VMU) or Video Display Unit (VDU).

2.2.1 Cathode Ray Tube (CRT)

The cathode ray tube (CRT) is a vacuum tube containing an electron gun (a source of electrons) and a fluorescent screen used to view images. It has a means to accelerate and deflect the electron beam(s) to the fluorescent screen to create the images. It is a vacuum tube made up of evacuated glass which is large, deep and heavy. Schematic representation of CRT is shown in Figure 2.1.

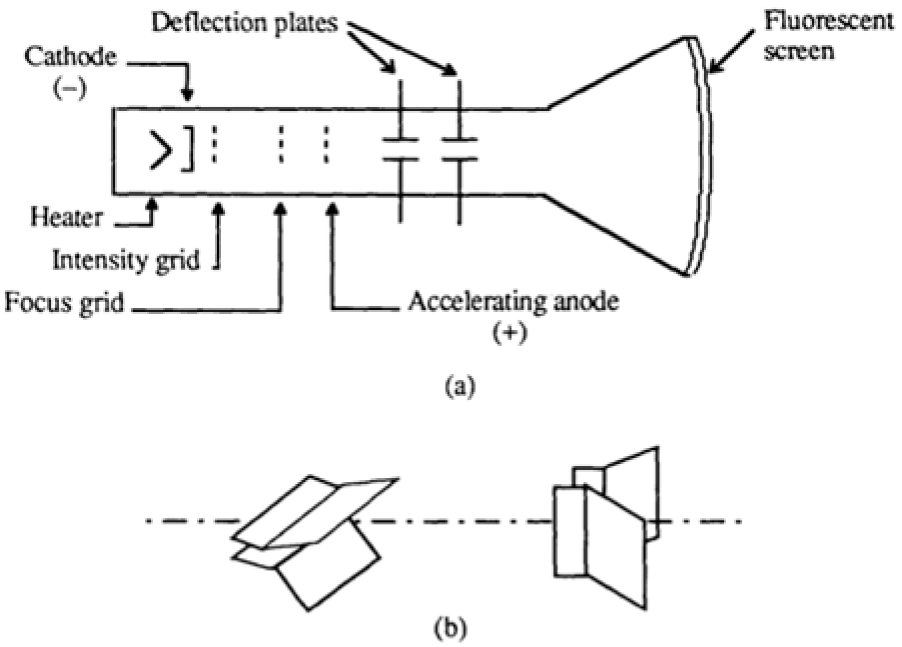
The cathode ray is a beam of electrons which are emitted by the heated cathode (negative electrode) and accelerated toward the fluorescent screen. Combination of cathode, intensity grid, focus grid, and accelerating anode is called an electron gun. Purpose of electron gun is to generate the electron beam and control its intensity and focus. There are two deflection plates between the electron gun and the fluorescent screen. One is to provide horizontal deflection of the beam and one to give vertical deflection to the beam.

Various components of CRT (Figure 2.2) are:

1. Electron gun
2. Electron beam
3. Focusing Coils

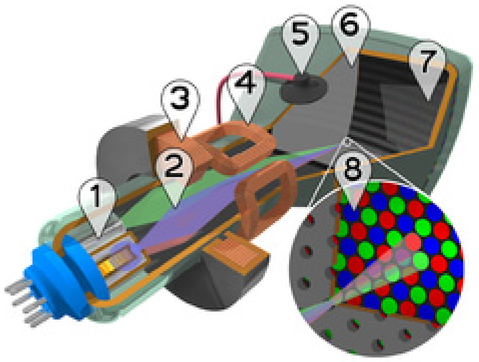
Notes

Figure 2.1: Cathode Ray Tube (a) Schematic (b) Detail of Deflection Plate



4. Deflection Coils
5. Anode Connection
6. Shadow mask
7. Phosphor layer
8. Close-up of the phosphor coated inner side of the screen

Figure: 2.2: A Rendered CRT



Source: http://upload.wikimedia.org/wikipedia/commons/thumb/9/9b/CRT_color enhanced. png/250px-CRT_color_ enhanced.png

There are two techniques used for producing image on the CRT screen: Vector scan/random scan and Raster scan. We will cover them one by one later in this unit.

Notes

Notes The first commercially made electronic television sets with cathode ray tubes were manufactured by Telefunken in Germany in 1934.

Self Assessment

Fill in the blanks:

4. The display systems are often referred to as ....

5. The .... is a vacuum tube containing an electron gun (a source of electrons) and a fluorescent screen used to view images.

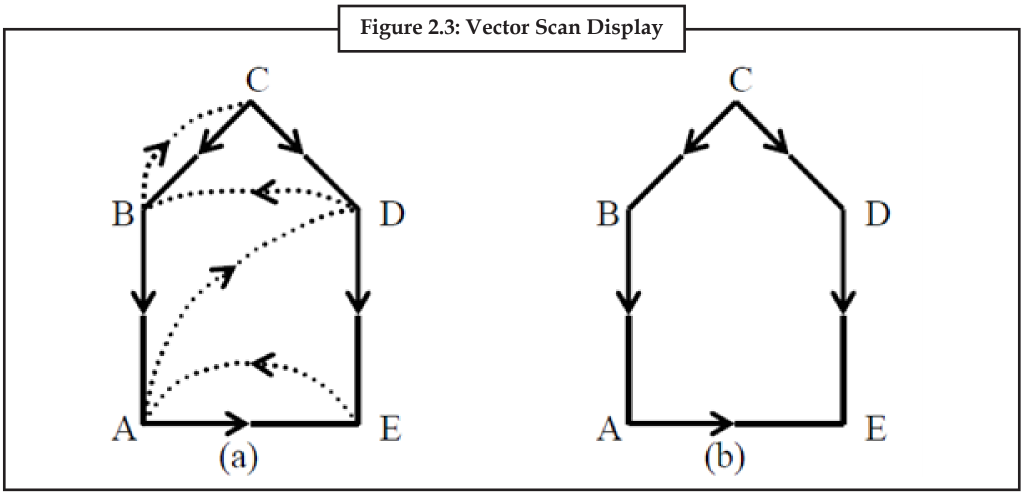
6. There are two techniques used for producing image on the CRT screen: Vector scan/random scan and .... scan.

2.3 Vector Scan/Random Scan Display

Vector scan display directly traces out only the desired lines on the CRT tube i.e. a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn. Random scan monitors draw a picture with one line at a time and for this reason it is also called Vector Scan Displays. Thus, in a vector display, the image is composed of drawn lines rather than a grid of glowing pixels as in raster graphics. An arbitrary path is followed by the electron beam to trace the connected sloped lines, rather than following the same horizontal raster path for all images. Dark areas of images are skipped by the beam.

Did u know? Figure 2.3 shows five line segments, the right and left roof slopes, the right and left walls and the floor line. Effectiveness can be achieved by minimizing wasted motion. Hence, it would be better to trace the path like ABCDE, than as CB, CD, BA, DE, and AE.

Figure 2.3: Vector Scan Display

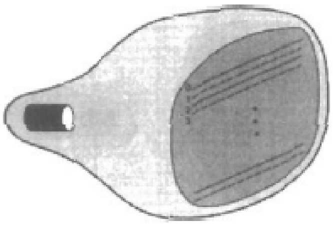


Source: http://www.expertsmind.com/CMSImages/808_Line%20Drawing%20Display%20-%20Random%20Scan%20Display%20Device%202.png

When functioned as a random-scan display unit, a CRT has the electron beam directed only to the components of the computer display where a picture is to be drawn. The constituent lines of an image can be drawn and refreshed by a random-scan.

Notes

Figure 2.4: Random-Scan Display Unit



Source: http://2.bp.blogspot.com/_hlVhUoVqhz8/TVUsQwHetRI/AAAAAAAAATA/wtxx9cZWVjI/s400/Interlacing+scan+lines+on+a+raster-scan+display.png

Self Assessment

Fill in the blanks:

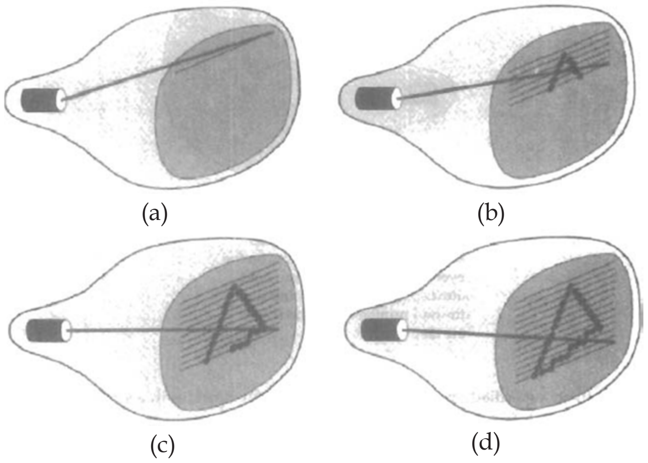
7. Random scan monitors draw a picture with one line at a time and for this reason it is also called ....

8. When functioned as a ...., a CRT has the electron beam directed only to the components of the computer display where a picture is to be drawn.

2.4 Raster-Scan Displays

The most common kind of graphics using a CRT is the raster-scan display. It is bottomed on TV technology. In a raster-scan system, the electron beam is cleared over the screen, one row at a time from top to bottom. As the electron beam moves over each row, the beam intensity is turned on and off to create a pattern of illuminated locations. Picture definition is stored in a memory area called the refresh buffer or frame buffer. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and displayed on the screen one row (scan line) at a time. Each computer display screen point is referred to as a pixel. The capability of a raster-scan system to store intensity for each computer display point makes it well matched for the realistic display of scenes including subtle shadowing and color patterns. Home television groups and printers are examples of other systems using raster-scan methods.

Figure 2.5: A Raster-scan System Displays an Object as a Set of Discrete Points across Each Scan Line



Source: http://1.bp.blogspot.com/_hlVhUoVqhz8/TVUsRKnGqaI/AAAAAAAAATM/ioYKjIRx2As/s1600/raster-scan+system+displays+an+object+as+a+set+of+discrete+points+across+each+scan+line.png

Notes

On some raster-scan systems (and in TV sets), each frame is displayed in two passes using an interlaced refresh procedure. In the first pass, the beam sweeps over every other scan line from top to bottom. Then after the vertical retrace, the beam clears out the residual scan lines. Interlacing of the scan lines in this way permits us to see the entire computer display displayed in one-half the time it would have taken to clear over all the lines at one time from top to bottom. Interlacing is mainly used with slower refreshing rates. On an older, 30 frames-per-second, non-interlaced display, for example, some flicker is obvious. But with interlacing, each of the two passes can be carried out in 1/60th of a second, which brings the refresh rate nearer to 60 frames per second. This is a productive technique for avoiding flicker, supplying that adjacent scan lines comprise similar display information.

Task Make a difference between raster scan and random scan technique.

Self Assessment

Fill in the blanks:

9. In a ...., the electron beam is cleared over the screen, one row at a time from top to bottom.

10. Picture definition is stored in a memory area called the ....

2.5 Resolution

The resolution of a digital television or computer monitor display device is the number of distinct pixels in each dimension that can be displayed. It can be an ambiguous term particularly as the displayed resolution is controlled by different components in cathode ray tube (CRT), Flat panel display which includes liquid crystal displays, or projection displays using fixed pixel arrays.

It is usually given as width × height, with the units in pixels: for example, “1024 × 768” means the width is 1024 pixels and the height is 768 pixels. This example would commonly be called as “ten twenty-four by seven sixty-eight” or “ten twenty-four by seven six eight”.

Notes Printers, monitors, scanners etc. are frequently known as high resolution, medium resolution, or low resolution.

Televisions are of the following resolutions:

1. Standard definition television (SDTV):
   (a) 480i (NTSC standard uses an analog system of 486i split into two interlaced fields of 243 lines)
   (b) 576i (PAL, 720 × 576 split into two interlaced fields of 288 lines)
2. Enhanced definition television (EDTV):
   (a) 480p (720 × 480 progressive scan)
   (b) 576p (720 × 576 progressive scan)
3. High definition television (HDTV):
   (a) 720p (1280 × 720 progressive scan)

(b) 1080i (1920 × 1080 split into two interlaced fields of 540 lines)

(c) 1080p (1920 × 1080 progressive scan)

4. Ultra-high definition television (UHDTV)

(a) 2160p (3840 × 2160 progressive scan)

(b) 4320p (7680 × 4320 progressive scan)

(c) 8640p (15360 × 8640 progressive scan)

Following Table 2.1 shows some common display resolutions for computer monitor display.

Table 2.1: Some Common Display Resolutions for Computer Monitor Display

Most common display resolutions in first half of 2012

Acronym	Aspect ratio	Width (px)	Height (px)	% of Steam users	% of web users
VGA	4:3	640	480	00.02	n/a
SVGA	4:3	800	600	00.17	01.03
WSVGA	~17:10	1024	600	00.31	02.25
XGA	4:3	1024	768	05.53	18.69
XGA+	4:3	1152	864	00.87	01.55
WXGA	16:9	1280	720	01.51	01.54
WXGA	5:3	1280	768	n/a	01.54
WXGA	16:10	1280	800	04.25	12.97
SXGA–(UVGA)	4:3	1280	960	00.72	00.72
SXGA	5:4	1280	1024	10.66	07.49
HD	~16:9	1360	768	02.36	02.28
HD	~16:9	1366	768	17.19	19.14
SXGA+	4:3	1400	1050	00.18	n/a
WXGA+	16:10	1440	900	07.60	06.61
HD+	16:9	1600	900	06.82	03.82
UXGA	4:3	1600	1200	00.53	n/a
WSXGA+	16:10	1680	1050	10.26	03.66
FHD	16:9	1920	1080	25.04	05.09
WUXGA	16:10	1920	1200	03.65	01.11
QWXGA	16:9	2048	1152	00.13	n/a
WQHD	16:9	2560	1440	00.72	00.36
WQXGA	16:10	2560	1600	00.19	n/a
	3:4	768	1024	n/a	01.93
	16:9	1093	614	n/a	00.63
	~16:9	1311	737	n/a	00.35
Other				01.29	07.25

Source: http://en.wikipedia.org/wiki/Display_resolution

Self Assessment

State whether the following statements are true or false:

11. The resolution of a digital television or computer monitor display device is the number of distinct pixels in each dimension.

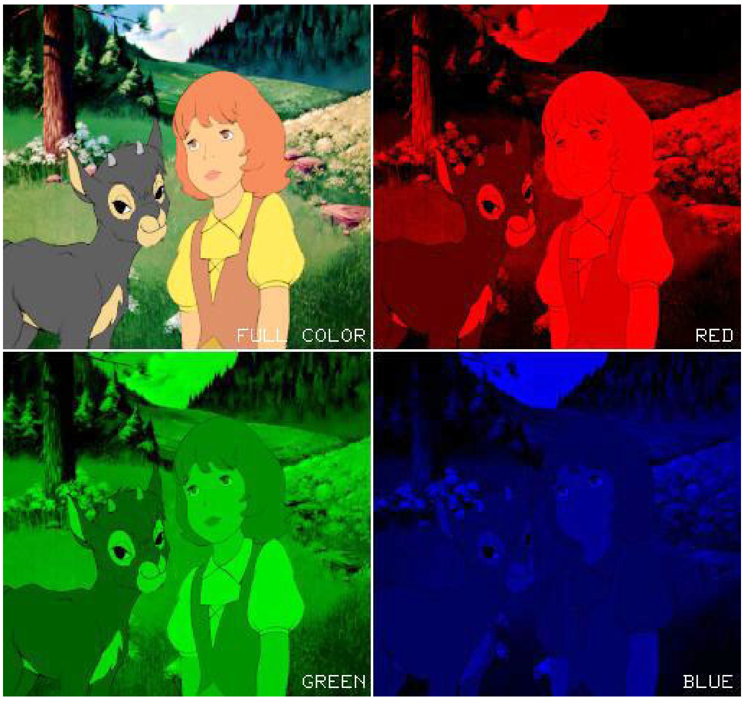
12. Printers, monitors, scanners etc. are frequently known as high resolution, medium resolution, or low resolution.

Notes

2.6 Color Display

Computers typically display color in three components red, green, and blue. When combined, these three colors make the full-color image as seen in Figure 2.6.

Figure 2.6 Color Image



Source: http://www.graphics.cornell.edu/online/tutorial/color/heidirgb.jpg

The RGB color model is color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers.

Did u know? RGB is a device-dependent color model which means different devices detect or reproduce a given RGB value differently.

Display technologies influence how colors are represented and thus can depend on:

- Physical differences between inks
- Dot gain – ink can tend to spread slightly depending on how it lays on or is absorbed by the paper being used
- LCD versus RGB displays – both are different technologies so have different color sets
- NTSC versus RGB displays – depends on resolution, interlacing, color encoding

Self Assessment

Fill in the blanks:

13. Computers typically display color in three components – red, ..., and ....

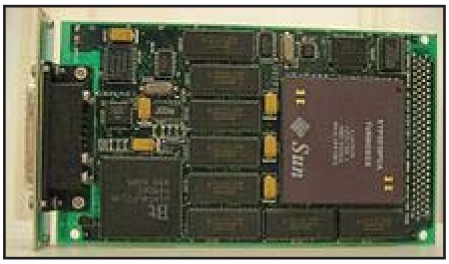
14. The .... is color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors.

2.7 Frame Buffer

A Framebuffer (or occasionally called Framestore) is a video output device that drives a video display from a memory buffer containing an entire frame of data.

The data in the memory buffer normally comprises of color values for every pixel (point that can be displayed) on the computer display. Color standards are commonly retained in 1-bit binary (monochrome), 4-bit palettized, 8-bit palettized, 16-bit highcolor and 24-bit truecolor formats. An additional alpha channel is occasionally used to retain data about pixel transparency. The total allowance of the memory required to propel the framebuffer depends on the resolution of the output signal, and on the color deepness and palette size.

Figure 2.7: Overhead Photo of a Sun TGX Framebuffer

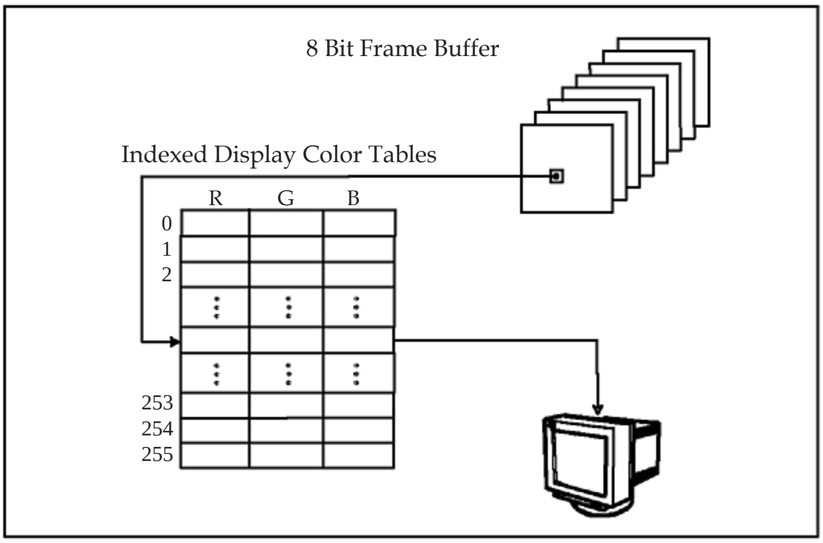


Source: en.wikipedia.org/wiki/File:Sun_sbus_cgsix_framebuffer.jpg

Framebuffers used in personal and home computing often had sets of defined modes under which the framebuffer could operate. They would automatically reconfigure the hardware to output different resolutions, color depths, memory layouts and refresh rate timings.

An indexed frame buffer contains only one display color table index for each pixel on the display surface as shown in the Figure 2.8, as opposed to the three color indexes provided by a component buffer.

Figure 2.8: Indexed Frame Buffer



Source: http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.graPHIGS/doc/phigsund/figures/AFM24rac.jpg

Component frame buffers provide millions more simultaneously displayable colors than do indexed frame buffers due to the separate indexing of the red, green, and blue components. For example, a workstation with a display color table of 512 entries and a component frame buffer with a bit depth of 24 bits (8 bits for each color component) will be capable of displaying from a palette of 16.7 million colors simultaneously. The same workstation would only be able to display 512 colors simultaneously using an index frame buffer.

Self Assessment

Fill in the blanks:

15. A .... is a video output device that drives a video display from a memory buffer containing an entire frame of data.

16. An .... frame buffer contains only one display color table index for each pixel on the display surface.

2.8 Summary

- Many programming languages provide libraries for creating GUI's. When combined with a display window, these devices can be an effective way to interact with a graphical system.
- The cathode ray tube (CRT) is a vacuum tube containing an electron gun (a source of electrons) and a fluorescent screen used to view images.
- Vector scan display directly traces out only the desired lines on the CRT tube i.e. a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn.
- The most common kind of graphics using a CRT is the raster-scan display. It is bottomed on TV technology. In a raster-scan system, the electron beam is cleared over the screen, one row at a time from top to bottom.
- On some raster-scan systems (and in TV sets), each frame is displayed in two passes using an interlaced refresh procedure. In the first pass, the beam sweeps over every other scan line from top to bottom.
- The resolution of a digital television or computer monitor display device is the number of distinct pixels in each dimension that can be displayed.
- Computers typically display color in three components – red, green, and blue. When combined, these three colors make the full-color image
- A Framebuffer (or occasionally called Framestore) is a video output device that drives a video display from a memory buffer containing an entire frame of data.

2.9 Keywords

2D (two dimensional): 2D computer graphics is the computer-bottomed generation of digital images mostly from two-dimensional models (such as 2D geometric models) and by techniques specific to them.

3D (three dimensional): 3D computer graphics are graphics that use a three-dimensional representation of geometric data (often Cartesian) that is stored in the computer for the purposes of performing calculations and rendering 2D images.

Cathode Ray Tube (CRT): The CRT is a vacuum tube containing an electron gun (a source of electrons) and a fluorescent screen used to view images.

Cathode Ray: The cathode ray is a beam of electrons which are emitted by the heated cathode (negative electrode) and accelerated toward the fluorescent screen.

Computer Animation: Computer animation refers to the art of creating moving images with the use of computers.

Computer Graphics: Computer graphics are graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software.

Framebuffer: A Framebuffer (or occasionally called Framestore) is a video output device that drives a video display from a memory buffer containing an entire frame of data.

Raster-scan System: In a raster-scan system, the electron beam is cleared over the screen, one row at a time from top to bottom.

RGB color Model: The RGB color model is color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors.

Vector Scan Displays: Random scan monitors draw a picture with one line at a time and for this reason it is also called Vector Scan Displays.

2.10 Review Questions

1. What is the computer graphics system?
2. Write down the different types of computer generated images. Also give examples.
3. Discuss the video display devices.
4. What is Cathode Ray Tube (CRT)?
5. Make a list of various components of CRT.
6. Explain the vector scan/random scan display.
7. "A raster-scan system displays an object as a set of discrete points across each scan line". Elaborate.
8. Explain the resolution. Give examples.
9. Explain the colour displays. How do we see colour display?
10. Write down the meaning of framebuffer.

Answers: Self Assessment

1. 2D (two dimensional)
2. 3D (three dimensional)
3. Computer animation
4. Video Monitor Unit (VMU) or Video Display Unit (VDU).
5. Cathode ray tube (CRT)
6. Raster
7. Vector Scan Displays
8. Random-scan display unit
9. Raster-scan system
10. Refresh buffer or frame buffer
11. True
12. True
13. Green and blue
14. RGB color model
15. Framebuffer
16. indexed

Notes

2.11 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. "*Computer Graphics*" Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Angel, Edward. "*Interactive Computer Graphics*" Pearson Education.

Krishnamurthy, N. "*Introduction to Computer Graphics*" Tata McGraw Hill.

Online links

http://www.dgp.toronto.edu/~hertzman/418notes.pdf

http://www.eazynotes.com/notes/computer-graphics/notes/short-answer-type-questions.pdf

http://www.inf.ed.ac.uk/teaching/courses/cg/Web/intro_graphics.pdf

Unit 3: Implementing Line Algorithm

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the concept of Implementing Line Algorithm
- Discuss the concept of Bresenham's Algorithm
- Define DDA Line Algorithm

Introduction

The Bresenham algorithm is probably the most efficient of all line drawing algorithms. It greatly simplifies line drawing by using only integer variables, and importantly removing that costly division operation for slope. Before we begin implementing the algorithm, it is advisable to revise the method for drawing line in an inefficient way. This can help lay out the fundamentals of line algorithm, and is very useful since Bresenham algorithm itself is an extension of the inefficient one anyway. In this unit, you will learn about implementing line algorithm. Concept of Bresenham algorithm will be discussed later in this unit. Finally, you will learn about DDA line algorithm.

3.1 Concept of Implementing Line Algorithm

A line drawing algorithm is a graphical algorithm for approximating a line segment on discrete graphical media. On discrete media, such as pixel-based displays and printers, line drawing needs such an approximation. On continuous media, by contrast, no algorithm is essential to draw a line.

Example: Oscilloscopes use natural phenomena to draw lines and curves.

The Cartesian slope-intercept equation for a straight line is $Y = mx + a$. Here m represents the slope of the line and a represent y intercept. If it is given that the two endpoints of the line

Notes

segment at positions (x_1,y_1) and (x_2,y_2), we can find values for the slope m and y intercept b with the following calculations,

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

so, $a = y_1 - m.x_1$

A naïve line-drawing algorithm

```
$dx = x_2 - x_1$
$dy = y_2 - y_1$
for $x$ from $x_1$ to $x_2$ {
$y = y_1 + (dy) * (x - x_1)/(dx)$
plot($x$, $y$)
}
```

It is assumed here that the points have already been ordered so that $x_2 > x_1$. This algorithm works just fine when $dx > = dy$ (i.e., slope is less than or equal to 1), but if $dx < dy$ (i.e., slope is greater than 1), the line get lots of gaps, and in the case of $dx = 0$, only a single point is plotted.

> *Notes* The naïve line drawing algorithm is inefficient and thus, works slow on a digital computer. Line drawing algorithms such as Bresenham's is preferred instead.

Self Assessment

Fill in the blanks:

1. A .... algorithm is a graphical algorithm for approximating a line segment on discrete graphical media.
2. On ...., no algorithm is essential to draw a line.
3. The Cartesian slope-intercept equation for a straight line is Y = ...
4. The .... drawing algorithm is inefficient and thus, works slow on a digital computer.

3.2 Concept of Bresenham's Algorithm

The Bresenham line algorithm is an algorithm which determines which order to form a close approximation to a straight line between two given points. It is routinely used to draw lines on a computer screen, as it benefits only integer addition, subtraction and bit shifting operations, all of which are very cheap procedures.

Did u know? It is one of the earliest algorithms evolved in the area of computer graphics.

3.2.1 Algorithm

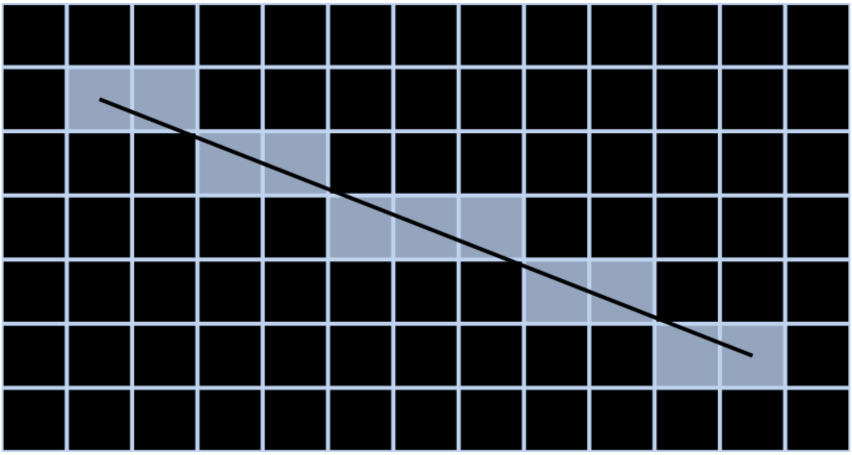
The common conventions will be used:

- the top-left is (0,0) such that pixel coordinates increase in the right and down directions [e.g. that the pixel at (7,4) is directly above the pixel at (7,5)], and
- that the pixel centers have integer coordinates.

Notes

The endpoints of the line are the pixels at (x_1, y_1) and (x_2, y_2), where the first coordinate of the pair is the column and the second is the row.

Figure 3.1: Illustration of the Result of Bresenham's Line Algorithm



Source: http://en.wikipedia.org/wiki/File:Bresenham.svg

Bresenham's algorithm chooses the integer y corresponding to the pixel center that is closest to the ideal (fractional) y for the same x; on successive columns y can remain the same or increase by 1. The general equation of the line through the endpoints is given by:

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}.$$

Since we know the column, x, the pixel's row, y, is given by rounding this quantity to the nearest integer:

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0.$$

The slope (y_1 – y_0)/ (x_1 – x_0) depends on the endpoint coordinates only and can be precomputed, and the ideal y for successive integer values of x can be computed starting from y_0 and repeatedly adding the slope.

We will use variables x_1, x_2, y_1, y_2 here in our algorithm.

In the following pseudocode sample plot(x,y) plots a point and abs returns absolute value:

```
function line($x_1$, $x_2$, $y_1$, $y_2$)
intdeltax := $x_2$ – $x_1$
intdeltay := $y_2$ – $y_1$
real error := 0
realdeltaerr := abs (deltay/deltax) // Assume deltax != 0 (line is not vertical),
//note that this division needs to be done in a way that preserves the fractional part
int $y$ : = $y_1$
for $x$ from $x_1$ to $x_2$
plot($x$, $y$)
```

```
        error := error + deltaerr
        if error ≥ 0.5 then
            $y$ := $y$ + 1
            error := error - 1.0
```

3.2.2 Optimization

Problem with this approach is that computers operate relatively slowly numbers which are in fraction like error and deltaerr. Working with integers is faster and more accurate. Here we use a trick to multiply all the fractional numbers (including the constant 0.5) in the code above by deltax, by which we can express them in form on integers. To make it work accordingly we modify the code as:

```
function line($x_1$, $y_1$, $x_2$, $y_2$)
    boolean steep := abs($y_2 - y_1$) > abs($x_2 - x_1$)
    if steep then
        swap($x_1$, $y_1$)
        swap($x_2$, $y_2$)
    if $x_1 > x_2$ then
        swap($x_1$, $x_2$)
        swap($y_1$, $y_2$)
    intdeltax: = $x_2 - x_1$
    intdeltay:= abs($y_2 - y_1$)
    int error := deltax/2
    intystep
    int $y$ := $y_1$
    if $y_1 < y_2$ thenystep := 1 elseystep := – 1
    for $x$ from $x_1$ to $x_2$
        if steep then plot($y$,$x$) else plot($x$,$y$)
        error := error – deltay
        if error < 0 then
            $y$ := $y$ + $y_{\text{step}}$
            error := error + deltax
```

Now, if you want to control the points in order of appearance (for example to print several consecutive dashed lines) you will have to again modify the code (actually skip the second swap):

```
function line($x_1$, $y_1$, $x_2$, $y_2$)
    boolean steep := abs($y_2 - y_1$) > abs($x_2 - x_1$)
    if steep then
        swap($x_1$, $y_1$)
swap($x_2$, $y_2$)
intdeltax := abs($x_2$ – $x_1$)
intdeltay := abs($y_2$ – $y_1$)
int error := deltax/2
intystep
int $y$ : = $y_1$
intinc REM added
if $x_1$ < $x_2$ theninc: = 1 elseinc := –1 REM added
if $y_1$ < $y_2$ thenystep: = 1 elseystep := – 1
for $x$ from $x_1$ to $x_2$ with incrementinc REM changed
if steep then plot($y$,$x$) else plot($x$,$y$)
    REM increment here a variable to control the progress of the line drawing
error := error – deltay
if error < 0 then
$y$ := $y$ + $y$step
error := error + deltax
```

3.2.3 Simplification

It is further possible to eliminate the swaps in the initial stage by considering the error calculation for both directions simultaneously:

```
function line ($x_1$, $y_1$, $x_2$, $y_2$)
$dx$ := abs($x_2$ – $x_1$)
$dy$ : = abs($y_2$ – $y_1$)
if $x_1$ < $x_2$ thensx := 1 elsesx := – 1
if $y_1$ < $y_2$ thensy := 1 elsesy := – 1
err := $dx$ – $dy$
loop
plot($x_1$,$y_1$)
if $x_1$ = $x_2$ and $y_1$ = $y_2$ exit loop
$e_2$ := 2*err
if $e_2$ > – $dy$ then
err := err – $dy$
$x_1$ := $x_1$ + $sx$
end if
if $x_1$ = $x_2$ and $y_1$ = $y_2$ then
plot($x_1$, $y_1$)
```

Notes

```
exit loop
end if
if $e_2 <\ dx$ then
err := err + $dx$
$y_1 := y_1 + sy$
end if
end loop
```

3.2.4 Derivation of Bresenham's Algorithm

To derive Bresenham's algorithm, two steps are required. The first step is transforming the equation of a line from the typical slope-intercept form into something different second step is using this new equation for a line to draw a line based on the idea of accumulation of error.

Line Equation

The slope-intercept form of a line can be written as:

$$y = f(x) = mx + b$$

where m is the slope and b is the y-intercept. This is a function of only x and it would be useful to make this equation written as a function of both x and y.

Using algebraic manipulation and recognition that the slope is the $\Delta y/\Delta$x then

$$\begin{aligned} y &= mx + b \\ y &= \frac{(\Delta y)}{(\Delta x)}x + b \\ (\Delta x)y &= (\Delta y)x + (\Delta x)b \\ 0 &= (\Delta y)x - (\Delta x)y + (\Delta x)b. \end{aligned}$$

Let the last equation be a function of x and y then it can be written as:

$$f(x, y) = 0 = Ax + By + C$$

where the constants are:

$$\begin{aligned} \mathrm{A} &= \Delta y \\ \mathrm{B} &= -\Delta x \\ \mathrm{C} &= (\Delta x)b \end{aligned}$$

The line is then defined for some constants A, B, and C and anywhere $f(x, y) = 0$.

Notes For any (x, y) not on the line then $(x, y) \neq 0$ it should be noted that everything about this form involves only integers if x and y are integers as the constants are necessarily integers.

For an example, the line $y = \frac{1}{2}x + 1$ could be written as $f(x, y) = x - 2y + 2$.

The point (2,2) is on the line

$$f(2, 2) = x - 2y + 2 = (2) - 2(2) + 2 = 2 - 4 + 2 = 0$$

and the point (2,3) is not on the line

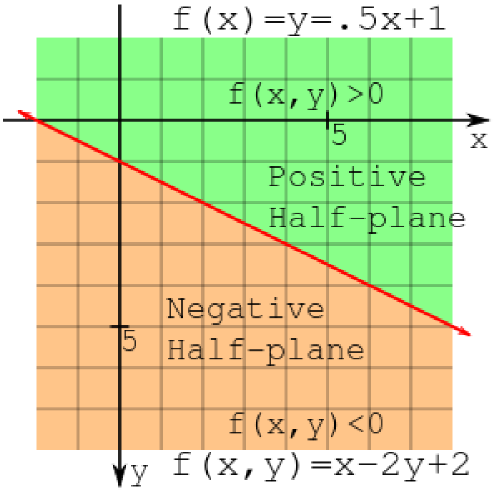
$$f(2, 3) = (2) - 2(3) + 2 = 2 - 6 + 2 = - 2$$

and neither is the point (2,1)

$$f(2, 1) = (2) - 2(1) + 2 = 2 - 2 + 2 = 2$$

Also note that the points (2,1) and (2,3) are on opposite sides of the line and $f(x,y)$ evaluates to positive or negative.

Figure 3.2: Positive and Negative Half-planes



Source: http://en.wikipedia.org/wiki/File:Line_1.5x%2B1_--_planes.svg

As you can see line splits a plane into halves and the half-plane that has a negative $f(x,y)$ can be called the negative half-plane, and the other half can called the positive half-plane.

Algorithm

Clearly, the starting point is on the line

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

only because the line is defined to start and end on integer coordinates.

⚠ *Caution* Keeping in mind that the slope is less-than-or-equal-to 1, the problem now presents itself as to whether the next point should be at $(x_0 + 1, y_0)$ or $(x_0 + 1, y_0 + 1)$.

The point should be chosen based upon which is closer to the line at $x_0 + 1$. If it is closer to the former then include the former point on the line, if the latter then the latter.

To answer this, let us evaluate the line function at the midpoint between these two points:

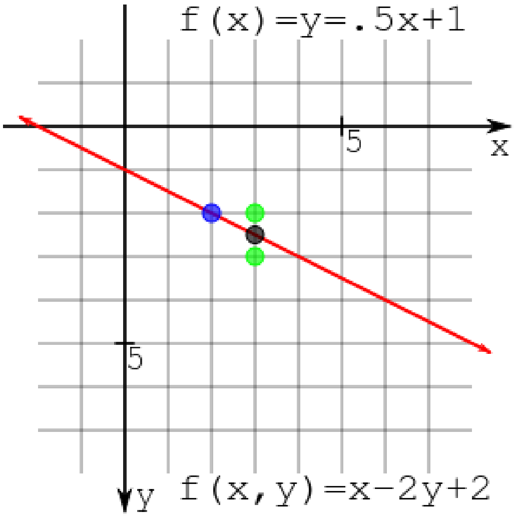
$$f(x_0 + 1, y_0 + 1/2)$$

If the value of this is positive then the ideal line is below the midpoint and closer to the candidate point $(x_0 + 1, y_0 + 1)$. Otherwise, the ideal line passes through or above the midpoint, and in this case chooses the point $(x_0 + 1, y_0)$.

Notes

The Figure 3.3 shows the blue point (2,2) chosen to be on the line with two candidate points in green (3,2) and (3,3). The black point (3, 2.5) is the midpoint between the two candidate points.

Figure 3.3: Candidate Point (2,2) in Blue and Two Candidate Points in Green (3,2) and (3,3)



Source: http://en.wikipedia.org/wiki/File:Line_1.5x%2B1_--_candidates.svg

Algorithm with Integer Arithmetic

Alternatively, the difference between points can be used instead of evaluating $f(x,y)$ at midpoints. This alternative method allows for integer-only arithmetic, which is generally considered faster than using floating-point arithmetic.

To derive the alternative method, define the difference to be as follows:

$$D = f(x_0 + 1, y_0 + 1/2) - f(x_0, y_0).$$

Simplifying this expression yields:

$$D = [A(x_0+1)+B(y_0+1/2)+C]-[Ax_0+By_0+C]$$

$$= \left[Ax_0+By_0+C+A+\frac{1}{2}B\right]-[Ax_0+By_0+C]$$

$$= A+\frac{1}{2}B$$

Now, if D is positive, then choose $(x_0 + 1, y_0 + 1)$, otherwise choose $(x_0 + 1, y_0)$.

The decision for the second point can be written as

$$f(x_0+2, y_0+1/2) - f(x_0+1, y_0+1/2) = A = \Delta y$$

$$f(x_0+2, y_0+3/2) - f(x_0+1, y_0+1/2) = A + B = \Delta y - \Delta x$$

If the difference is positive then $(x_0 + 2, y_0 + 1)$ is chosen, otherwise $(x_0 + 2, y_0)$.

So, all of the derivation for the algorithm is done.

plotLine (x_1, y_1, x_2, y_2)

$dx = x_2 - x_1$

$dy = y_2 - y_1$

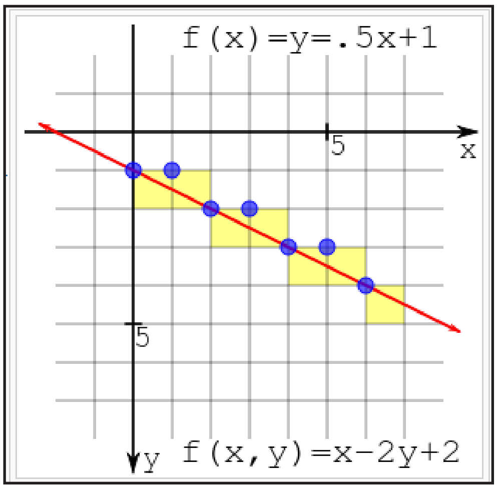
```
$D = 2 * dy - dx$
plot($x_1$,$y_1$)
$y = y_1$
for $x$ from $x_1 + 1$ to $x_2$
    if $D > 0$
        $y = y + 1$
        plot($x$, $y$)
        $D = D + (2 * dy - 2 * dx)$
    else
        plot($x$, $y$)
        $D = D + (2* dy)$
```

Now, let us take an example for this. Running this algorithm for $f(x, y) = x - 2y + 2$ from (0,1) to (6,4) yields the following differences with $dx = 6$ and $dy = 3$:

- D = 2*3-6=0
- plot(0,1)
- Loop from 1 to 6
    - $x = 1$: $D \leq 0$: plot(1,1), $D = 6$
    - $x = 2$: $D > 0$: $y = 2$, plot(2,2), $D = 6 + (6–12) = 0$
    - $x = 3$: $D \leq 0$: plot(3,2), $D = 6$
    - $x = 4$: $D > 0$: $y = 3$, plot(4,3), $D = 6 + (6 - 12) = 0$
    - $x = 5$: $D \leq 0$: plot(5,3), $D = 6$
    - $x = 6$: $D > 0$: $y = 4$, plot(6, 4), $D = 6 + (6 - 12) = 0$

The result of this plot is shown in Figure 3.4.

Figure 3.4: Plotting the Line from (0,1) to (6,4) Showing a Plot of Grid Lines and Pixels



Source: http://en.wikipedia.org/wiki/File:Line_1.5x%2B1_--_points.svg

Notes

The plotting can be viewed by plotting at the intersection of lines (blue circles) or filling in pixel boxes (yellow squares).

Bresenham's Line Drawing Algorithm in C

Following example shows how to draw a line with 0 <= slope <= 1, using the Bresenham's algorithm:

```
voiddraw_line(float x1,float y1,float x2,float y2)
{
floatdy,dx;
floatx,y;
float p, p0, dp1, dp2;
dy = y2 – y1;
dx = x2 – x1;
p0 = 2 * (dy – dx);
dp1 = 2 * dy;
dp2 = 2 * (dy – dx);
putpixel(x1, y1, EGA_WHITE);
p = p0;
for (x = x1 + 1, y = y1; x < x2; x++)
  {
if(p < 0)
   {
    p = p + dp1;
putpixel (x, y,EGA_WHITE);
   }
else
   {
    p = p + dp2;
y++;
putpixel (x, y,EGA_WHITE);
   }
   }
}
```

The Bresenham algorithm can be interpreted as slightly modified DDA.

Self Assessment

Fill in the blanks:

5. The .... algorithm is an algorithm which determines which order to form a close approximation to a straight line between two given points.

Notes

6. Working with .... is faster and more accurate.
7. A plane into halves and the half-plane that has a negative $f(x,y)$ can be called the ....
8. This alternative method allows for integer-only arithmetic, which is generally considered faster than using .... arithmetic.
9. The Bresenham algorithm can be interpreted as slightly modified ....

3.3 DDA Line Algorithm

In computer graphics, a hardware or software implementation of a digital differential analyzer (DDA) is used for linear interpolation of variables over an interval between start and end point.

DDAs are used for creation of lines, triangles and polygons. In its simplest implementation the DDA algorithm interpolates values in interval $[(x_{start}, y_{start}), (x_{end}, y_{end})]$ by computing for each x_i the equations $x_i = x_{i-1} + 1/m$, $y_i = y_{i-1} + m$, where $\Delta x = x_{end} - x_{start}$ and $\Delta y = y_{end} - y_{start}$ and $m = \Delta y/\Delta x$.

This method can be implemented using floating-point or integer arithmetic. The native floating-point implementation requires one addition and one rounding operation per interpolated value and output result.

> *Notes* This process is only efficient when a Floating point unit with fast add and rounding operation is available.

The fixed-point integer operation requires two additions per output cycle. In case of fractional value, one additional increment and subtraction is required. The probability of fractional part overflows is proportional to the ratio m of the interpolated start/end values.

Did u know? DDAs are well suited for hardware implementation and can be pipelined for maximized throughput.

3.3.1 Algorithm

DDA is a scan conversion line algorithm based on calculating either dy or dx. A line is sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for other coordinates.

Considering a line with positive slope, if the slope is less than or equal to 1, we sample at unit x intervals (dx = 1) and compute successive y values as

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

Subscript k takes integer values starting from 0, for the 1st point and increases by 1 until endpoint is reached. y value is rounded off to nearest integer to correspond to a screen pixel.

For lines with slope greater than 1, we reverse the role of x and y i.e. we sample at dy = 1 and calculate consecutive x values as

$$x_{k+1} = x_k + \frac{1}{m}.$$

Similar calculations are carried out to determine pixel positions along a line with negative slope. Thus, if the absolute value of the slope is less than 1, we set dx = 1 if $x_{start} < x_{end}$ i.e. the starting extreme point is at the left.

Self Assessment

State whether the following statements are true or false:

10. A hardware or software implementation of a digital differential analyzer (DDA) is used for linear interpolation of variables over an interval between start and end point.
11. DDAs are not used for creation of lines, triangles and polygons.
12. The fixed-point integer operation requires one addition per output cycle.

3.4 Summary

- A line drawing algorithm is a graphical algorithm for approximating a line segment on discrete graphical media.
- The Cartesian slope-intercept equation for a straight line is Y = mx + a. Here m represents the slope of the line and *a* represent *y* intercept.
- The Bresenham line algorithm is an algorithm which determines which order to form a close approximation to a straight line between two given points.
- Bresenham's algorithm chooses the integer *y* corresponding to the pixel center that is closest to the ideal (fractional) *y* for the same *x*; on successive columns *y* can remain the same or increase by 1.
- Problem with this approach is that computers operate relatively slowly numbers which are in fraction like error and deltaerr.
- To derive Bresenham's algorithm, two steps are required. The first step is transforming the equation of a line from the typical slope-intercept form into something different second step is using this new equation for a line to draw a line based on the idea of accumulation of error.
- In computer graphics, a hardware or software implementation of a digital differential analyzer (DDA) is used for linear interpolation of variables over an interval between start and end point.
- DDA is a scan conversion line algorithm based on calculating either *dy* or *dx*. A line is sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for other coordinates.

3.5 Keywords

Bresenham's Algorithm: The Bresenham line algorithm is an algorithm which determines which order to form a close approximation to a straight line between two given points.

Cartesian Slope-intercept: The Cartesian slope-intercept equation for a straight line is $Y= mx + a$.

Digital Differential Analyzer (DDA): In computer graphics, a hardware or software implementation of a digital differential analyzer (DDA) is used for linear interpolation of variables over an interval between start and end point.

Line Drawing Algorithm: A line drawing algorithm is a graphical algorithm for approximating a line segment on discrete graphical media.

Negative Half-plane: Line splits a plane into halves and the half-plane that has a negative $f(x,y)$ can be called the negative half-plane.

3.6 Review Questions

Notes

1. Briefly explain the concept of implementing line algorithm.
2. What is Bresenham's Algorithm?
3. Explain the algorithm for Bresenham. Also illustrate the result of Bresenham's line algorithm.
4. Explain the optimization for Bresenham.
5. How it is possible to eliminate the swaps in the initial stage?
6. What are the steps required for deriving Bresenham's algorithm?
7. Discuss the algorithm with integer arithmetic.
8. Explain the Bresenham's line drawing algorithm in C.
9. What is DDA line algorithm?
10. "DDA is a scan conversion line algorithm based on calculating either *dy* or *dx*". Discuss.

Answers: Self Assessment

1. Line drawing
2. Continuous media
3. $mx + a$
4. Naïve line
5. Bresenham line
6. Integers
7. Negative half-plane
8. Floating-point
9. DDA
10. True
11. False
12. False

3.7 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. "*Computer Graphics*"Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Angel, Edward. "*Interactive Computer Graphics*"Pearson Education.

Krishnamurthy, N. "Introduction to Computer Graphics" Tata McGraw Hill.

Online links

www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html

www.cs.princeton.edu/~rs/AlgsDS07/17GeometricSearch.pdf

tech-algorithm.com/articles/drawing-line-using-bresenham-algorithm/

Unit 4: Implementing Circle Algorithm

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss the concept of circle algorithm
- State the midpoint circle drawing algorithm
- Explain the Bresenham's circle drawing algorithm

Introduction

A circle is a round plane figure whose boundary (the circumference) consists of points equidistant from a fixed center. The diameter of a circle is the length of the line through the center and touching two points on its edge. In this unit, you will learn about implementing circle algorithm. After discussing concept of circle algorithm midpoint circle algorithm will be covered. Finally, Bresenham circle drawing algorithm will be covered in the unit.

4.1 Concept of Circle Algorithm

A circle is a simple shape of geometry that is the set of all points in a plane that are at a given distance from a given point, the centre. The distance between any of the points and the centre is called the radius.

The general equation of a circle is:

$x_2 + y_2 + Dx + Ey + F = 0$

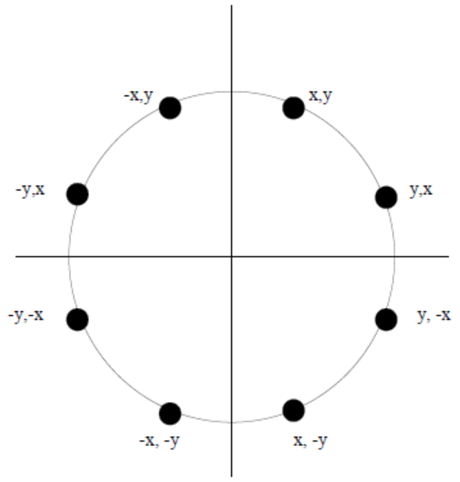
The "centre-radius" form of the equation is:

$(x - h)_2 + (y - k)_2 = r_2$

4.1.1 The Symmetry of a Circle

- Due to the eight way symmetry of a circle, it is necessary to calculate the positions of the pixels of only one octant.
- Up to eight points can thus be plotted simultaneously.

Figure 4.1: The 8-way Symmetry of a Circle



Source: staff.um.edu.mt/kurt//documents/grafx.pdfý

```
Procedure PlotCirclePoints (x, y: Integer);
Begin
    PlotPixel (xc + x, yc + y);
    PlotPixel (xc + x, yc – y);
    If x <> 0 then
    begin
        PlotPixel (xc – x, yc + y);
        PlotPixel (xc – x, yc – y)
    end;
    If x <> y then
    begin
        PlotPixel (xc + y, yc + x);
        PlotPixel (xc – y, yc + x);
        If x <> 0 then
        begin
            PlotPixel (xc + y, yc – x);
            PlotPixel (xc – y, yc – x)
        end
    end
End;
```

Notes

Self Assessment

Fill in the blanks:

1. The distance between any of the points and the centre is called the ....
2. A circle is a simple shape of geometry that is the set of all points in a plane that are at a given distance from a given point, the center ....
3. Due to the .... way symmetry of a circle, it is necessary to calculate the positions of the pixels of only one octant.

4.2 Midpoint Circle Algorithm

In computer graphics, the midpoint circle algorithm is an algorithm used to determine the points needed for drawing a circle. This algorithm is a variant of Bresenham's line algorithm, and is thus sometimes known as Bresenham's circle algorithm, although not actually invented by Jack E. Bresenham.

Did u know? In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points.

4.2.1 Algorithm

The algorithm starts with the circle equation $x^2 + y^2 = r^2$. For simplicity, assume the center of the circle is at (0, 0). First we will consider only the first octant and draw a curve which starts at point (r, 0) and proceeds counterclockwise, reaching the angle of 45.

The fast direction here is the y direction. The algorithm always takes a step in the positive direction (upwards), and occasionally takes a step in the "slow" direction (the negative direction).

From the circle equation we obtain the transformed equation $x^2 + y^2 – r^2 = 0$, where r^2 is computed only a single time during initialization.

Let the points on the circle be a sequence of coordinates of the vector to the point. Points are numbered according to the order in which they are drawn, with $n = 1$ assigned to the first point.

For each point, the following holds:

$$x_n^2 + y_n^2 = r^2$$

We can also write this as:

$$x_n^2 = r^2 - y_n^2$$

Also in similar way for next point we can write:

$$x_{n+1}^2 = r^2 - y_{n+1}^2$$

In general, it is true that:

$$\begin{aligned} y_{n+1}^2 &= (y_n + 1)^2 \\ &= y_n^2 + 2y_n + 1 \\ x_{n+1}^2 &= r^2 - y_n^2 - 2y_n - 1 \end{aligned}$$

So we rewrite our next-point-equation into a recursive one by substituting

$$\begin{aligned} x_n^2 &= r^2 - y_n^2 : \\ x_{n+1}^2 &= x_n^2 - 2y_n - 1 \end{aligned}$$

Caution As the continuity of a circle and the maxima along both axes is the same, we will not be skipping *x* points as we advance in the sequence.

The frequent computations of squares in the circle equation, trigonometric expressions and square roots can be avoided by dissolving everything into single steps. Then we can use recursive computation of the quadratic terms from the preceding iterations.

4.2.2 Variant with Integer-based Arithmetic

In the same way like Bresenham's line algorithm, this algorithm can be optimized for integer-based math.

We will start by defining the radius error as the difference between the exact representation of the circle and the center point of each pixel. For any pixel with a center at (x_i, y_i), we define the radius error to be:

$$RE(x_i, y_i) = \left| x_i^2 + y_i^2 - r^2 \right| .$$

For simplicity, we derive this formula for a circle at the origin, but the algorithm can be modified for any other location as well. We will start with the point (*r*, 0) on the positive X-axis. Because the radius will be a whole number of pixels, we can see that the radius error will be zero:

$$RE(x_i, y_i) = \left| r^2 + 0^2 - r^2 \right| = 0 .$$

We will step in the direction with the greatest travel, the Y direction, so we can say that $y_{i+1} = y_i + 1$. We can also create a decision variable that determines if the following is true:

$$RE\ (x_i - 1, y_i + 1) < RE(x_i, y_i + 1).$$

If this inequality holds, we will plot (x_i – 1, y_i + 1); if not, then we plot (x_i, y_i + 1). So how to determine if this inequality holds?

We can start this with our definition of radius error:

$$\begin{aligned} RE(x_i - 1, y_i + 1) &< RE(x_i, y_i + 1) \\ \left| (x_i - 1)^2 + (y_i + 1)^2 - r^2 \right| &< \left| x_i^2 + (y_i + 1)^2 - r^2 \right| \\ \left| (x_i^2 - 2x_i + 1) + (y_i^2 + 2y_i + 1) - r^2 \right| &< \left| x_i^2 + (y_i^2 + 2y_i + 1) - r^2 \right| \end{aligned}$$

The absolute value function is of not much use, so let's square both sides, since the square is always positive:

$$\begin{aligned} \left[(x_i^2 - 2x_i + 1) + (y_i^2 + 2y + 1) - r^2 \right]^2 &< \left[x_i^2 + (y_i^2 + 2y, + 1) - r^2 \right]^2 \\ \left[(x_i^2 + y_i^2 - r^2 + 2y_i + 1) + (1 - 2x_i) \right]^2 &< \left[x_i^2 + y_i^2 - r^2 + 2y_i + 1 \right]^2 \\ (x_i^2 + y_i^2 - r^2 + 2y + 1)^2 + 2(1 - 2x_i)(x_i^2 + y_i^2 - r^2 + 2y_i + 1) + (1 - 2x_i)^2 &< \left[x_i^2 + y_i^2 - r^2 + 2y_i + 1 \right]^2 \\ 2(1 - 2r_i)(x_i^2 + y_i^2 - r^2 + 2y_i + 1) + (1 - 2x_i)^1 &< 0 \end{aligned}$$

Notes

Since $x > 0$, the term $(1 - 2x_i) < 0$, so dividing we get:

$$2\left[\left(x_i^2 + y_i^2 - r^2\right) + \left(2y_i + 1\right)\right] + \left(1 - 2x_i\right) > 0$$

$$2\left[RE\left(x_i, y_i\right) + Y\,Change\right] + X\,Change\ > 0$$

So, now we change our decision criterion from using floating-point operations to simple integer addition, subtraction, and bit shifting.

If *2(RE+YChange)+XChange* > *0*, then we decrement our X value. If *2(RE+YChange)+XChange* <= *0*, then we keep the same X value.

Thus, by reflecting these points in all the octants, we will get the full circle.

Self Assessment

Fill in the blanks:

4. The .... is an algorithm used to determine the points needed for drawing a circle.
5. Because the radius will be a whole number of pixels, we can see that the radius error will be ....

4.3 Bresenham's Circle Drawing Algorithm Using C

Here is a C++/C# implementation of the integer-only variant that follows the logic very closely:

```
public static void DrawCircle(int x0, int y0, int radius)
{
 int x = radius, y = 0;
 int radiusError = 1–x;
  while(x >= y)
 {
  DrawPixel(x + x0, y + y0);
  DrawPixel(y + x0, x + y0);
  DrawPixel(– x + x0, y + y0);
  DrawPixel(– y + x0, x + y0);
  DrawPixel(– x + x0, – y + y0);
  DrawPixel(– y + x0, – x + y0);
  DrawPixel(x + x0, – y + y0);
  DrawPixel(y + x0, – x + y0);
   y++;
     if(radiusError<0)
          radiusError+=2*y+1;
     else
     {
            x—;
            radiusError+=2*(y – x + 1);
        }
  }
}
```

Let us see an implementation of the Bresenham Algorithm for a full circle in C. Here another variable for recursive computation of the quadratic terms $2n + 1$ is used, which corresponds with the term. It just has to be increased by 2 from one step to the next:

```
void rasterCircle(int x0, int y0, int radius)
{
  int f = 1 – radius;
  int ddF_x = 1;
  int ddF_y = -2 * radius;
  int x = 0;
  int y = radius;
 setPixel(x0, y0 + radius);
 setPixel(x0, y0 – radius);
  setPixel(x0 + radius, y0);
  setPixel(x0 – radius, y0);
   while(x < y)
  {
    // ddF_x == 2 * x + 1;
    // ddF_y == – 2 * y;
    // f == x*x + y*y – radius*radius + 2*x – y + 1;
    if(f >= 0)
    {
      y—;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x;
    setPixel(x0 + x, y0 + y);
    setPixel(x0 – x, y0 + y);
    setPixel(x0 + x, y0 – y);
    setPixel(x0 – x, y0 – y);
    setPixel(x0 + y, y0 + x);
```

Notes

```
    setPixel(x0 – y, y0 + x);
    setPixel(x0 + y, y0 – x);
    setPixel(x0 – y, y0 – x);
  }
}
```

Notes It should be noted that there is correlation between this algorithm and the sum of first odd numbers, which this one basically does. That is, $1+3+5+7+9+\cdots=\sum_{n=0}^{N-1}(2n+1)=N^2$.

4.3.1 Optimization

The following optimized circle algorithm is useful for machines where registers are scarce. Only three variables are needed in the main loop to perform the calculation.

```
// 'cx' and 'cy' denote the offset of the circle center from the origin.
void circle(int cx, int cy, int radius)
{
  int error = -radius;
  int x = radius;
  int y = 0;
   // The following while loop may be altered to 'while (x > y)' for a
  // performance benefit, as long as a call to 'plot4points' follows
  // the body of the loop. This allows for the elimination of the
  // '(x != y)' test in 'plot8points', providing a further benefit.
  //
  // For the sake of clarity, this is not shown here.
  while (x >= y)
  {
    plot8points(cx, cy, x, y);
     error += y;
    ++y;
    error += y;
     // The following test may be implemented in assembly language in
    // most machines by testing the carry flag after adding 'y' to
    // the value of 'error' in the previous step, since 'error'
    // nominally has a negative value.
    if (error >= 0)
    {
      error -= x;
      –x;
      error -= x;
    }
  }
}
void plot8points(int cx, int cy, int x, int y)
{
  plot4points(cx, cy, x, y);
  if (x != y) plot4points(cx, cy, y, x);
}
// The '(x != 0 && y != 0)' test in the last line of this function
// may be omitted for a performance benefit if the radius of the
// circle is known to be non-zero.
void plot4points(int cx, int cy, int x, int y)
{
  setPixel(cx + x, cy + y);
  if (x != 0) setPixel(cx - x, cy + y);
  if (y != 0) setPixel(cx + x, cy - y);
  if (x != 0 && y != 0) setPixel(cx - x, cy - y);
}
```

Self Assessment

State whether the following statements are true or false:

6. There is correlation between optimized circle algorithm and the sum of first odd numbers, which this one basically does.
7. Only two variables are needed in the main loop to perform the optimized circle algorithm calculation.

4.4 Summary

- The general equation of a circle is: $x_2 + y_2 + Dx + Ey + F = 0$
- In computer graphics, the midpoint circle algorithm is an algorithm used to determine the points needed for drawing a circle.
- The algorithm starts with the circle equation $x^2 + y^2 = r^2$.
- Eight-way symmetry can hugely reduce the work in drawing a circle.
- Moving in unit steps along the *x* axis at each point along the circle's edge we need to choose between two possible *y* coordinates.

- The midpoint circle drawing algorithm works on the same midpoint concept as the Bresenham's line algorithm.

4.5 Keywords

Circle: A circle is a simple shape of geometry that is the set of all points in a plane that are at a given distance from a given point, the centre.

Midpoint circle algorithm: The midpoint circle algorithm is an algorithm used to determine the points needed for drawing a circle.

Pixel: A pixel is a physical point in a raster image, or the smallest, addressable element in a display device; so it is the smallest, controllable element of a picture represented on the screen.

Radius: The distance between any of the points and the centre is called the radius.

4.6 Review Questions

1. Define circle. What is the circle algorithm?
2. Discuss about symmetry of a circle.
3. What is midpoint circle algorithm?
4. Write an algorithm implement midpoint circle drawing.
5. Discuss the variant with integer-based arithmetic.
6. Briefly explain the Bresenham's circle drawing algorithm.

Answers: Self Assessment

1. Radius
2. Center
3. Eight
4. Midpoint circle algorithm
5. Zero
6. True
7. False

4.7 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. "*Computer Graphics*" Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Angel, Edward. "*Interactive Computer Graphics*" Pearson Education.

Krishnamurthy, N. "*Introduction to Computer Graphics*" Tata McGraw Hill.

Online links

www.cecs.csulb.edu/~pnguyen/cecs449/lectures/midpointcircle.pdf

www.eazynotes.com/notes/.../algorithms/bresenham-circle-algorithm.pdf

www.slideshare.net/tahersb/bresenham-circle

ezekiel.vancouver.wsu.edu/~cs442/lectures/raster/circrect/circrect.ppt

Unit 5: Implementing Ellipse Algorithm

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain Bresenham's ellipse drawing algorithm
- Discuss the midpoint ellipse algorithm
- Identify ellipse area, sector area and segment area

Introduction

Ellipse is a regular oval shape, traced by a point moving in a plane so that the sum of its distances from two other points (the foci) is constant. This unit focuses on implementation of Ellipse algorithms. You will learn about Bresenham ellipse drawing algorithm. Later on in the unit, midpoint ellipse algorithm will be covered. Lastly, the key topics like identifying ellipse area, sector area and segment area will be covered.

5.1 Bresenham's Ellipse Drawing Algorithm

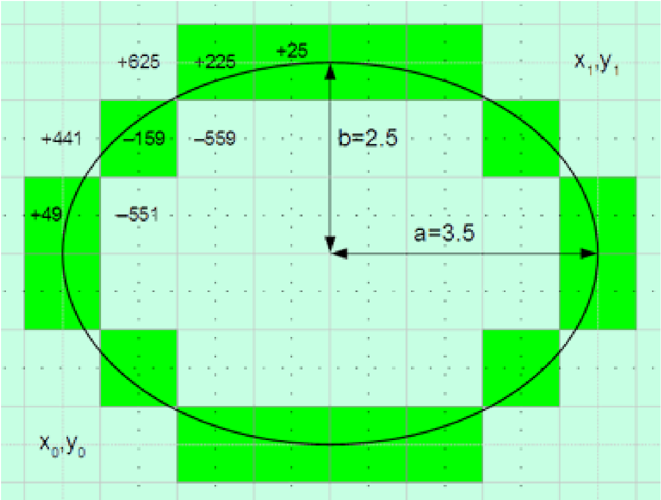
When looking at the circle representation on a graphics device, which distance between the rows of pixels is different from the distance between the columns, it is obvious that the circle on the screen looks like an ellipse. The construction of a centered ellipse can be done by the Bresenham's algorithm, similarly as at the circle, while we have to compute the values as a quadrant of a circle.

Notes Ellipse selects corner point (right mouse button) for new ellipse.

Notes

Example: We will take an example to plots an ellipse inside a specified rectangle.

Figure 5.1: Example of an Ellipse Inside a Specified Rectangle



Source: http://members.chello.at/easyfilter/ellipse.png

Code

```
void plotEllipseRect(int x0, int y0, int x1, int y1)
{
  int a = abs(x1 – x0), b = abs(y1 – y0), b1 = b & 1; /* values of diameter */
  long dx = 4*(1 – a)*b*b, dy = 4*(b1+1)*a*a; /* error increment */
  long err = dx + dy + b1*a*a, e2; /* error of 1.step */
  if (x0 > x1) { x0 = x1; x1 += a; } /* if called with swapped points */
  if (y0 > y1) y0 = y1; /* .. exchange them */
  y0 + = (b + 1)/2; y1 = y0 – b1;   /* starting pixel */
  a *= 8*a; b1 = 8*b*b;
  do {
    setPixel(x1, y0); /*   I. Quadrant */
    setPixel(x0, y0); /*  II. Quadrant */
    setPixel(x0, y1); /* III. Quadrant */
    setPixel(x1, y1); /*  IV. Quadrant */
    e2 = 2*err;
    if (e2 <= dy) {y0++; y1--; err += dy += a; }  /* y step */
    if (e2 >= dx || 2*err > dy) { x0++; x1--; err += dx += b1; } /* x step */
  } while (x0 <= x1);
    while (y0 – y1 < b) {  /* too early stop of flat ellipses a = 1 */
```

```
      setPixel (x0 – 1, y0); /* – > finish tip of ellipse */
      setPixel(x1+1, y0++);
      setPixel(x0 – 1, y1);
      setPixel(x1 + 1, y1--);
  }
}
```

Self Assessment

State whether the following statements are true or false:

1. At the circle representation on a graphics device, the distance between the rows of pixels is similar with the distance between the columns.
2. The construction of a centered ellipse can be done by the Bresenham's algorithm.

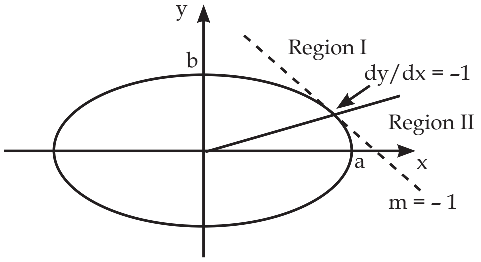
5.2 Midpoint Ellipse Algorithm

Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics. This method is derived from Bresenham's algorithm. The advantage of this derived method is that only addition operations are required in the program loops.

Did u know? This leads to simple and fast implementation in all processors.

Let us consider one quarter of an ellipse. The curve is divided into two regions. In region I, the slope on the curve is greater than –1 while in region II less than –1.

Figure 5.2: Midpoint Ellipse Algorithm Example



Source: http://winnyefanho.net/research/MEA.pdf

General equation of an ellipse:

$$b^2x^2 + a^2y^2 - a^2b^2 = 0$$

Here, a is the horizontal radius and b is the vertical radius, we can define a function $f(x,y)$ which can be used to obtain prediction coordinate (x,y). To form an ellipse the appropriate pixels can be selected according to the error.

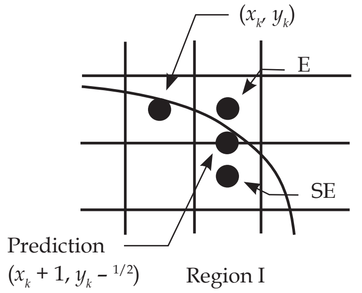
Caution The error can be confined within half a pixel.

Set $f(x,y) = b^2x^2 + a^2y^2 - a^2b^2$

Notes

In region I ($dy/dx > -1$), x is always incremented in each step, i.e. $x_{k+1} = x_k + 1$.

Figure 5.3: Region I Prediction



Source: http://winnyefanho.net/research/MEA.pdf

$y_{k+1} = y_k$ if E is selected, or $y_{k+1} = y_{k-1}$ if SE is selected.

To decide between S and SE, a prediction $(x_{k+1}, y_{k-½})$ is set at the middle between the two candidate pixels. A prediction function P_k can be defined as follows:

$$P_k = f(x_k + 1, y_k - {}^{½})$$

$$= b^2(x_k + 1)^2 + a^2(y_k - {}^{½})2 - a^2b^2$$

$$= b^2(x_k{}^2 + 2x_k + 1) + a^2(y_k{}^2 - y_k + {}^{¼}).$$

If $P_K < 0$, select E:

$$P_{k+1}{}^E = f(x_k + 2, y_k - ½)$$

$$= b^2(x_k + 2)^2 + a^2(y_k - ½)^2 - a^2b^2$$

$$= b(x_k{}^2 + 4x_k + 4) + a^2(y_k{}^2 + ¼) - a^2b^2.$$

Change of $P_k{}^E$ is: $\Delta P_k{}^E = P_{k+1}{}^E - P_k = b^2(2x_k + 3)$

If $P_K > 0$, select SE:

$$P_{k+1}^{SE} = f(x_k + 2, y_k - 3/2)$$

$$= b^2(x_k + 2)^2 + a^2(y_k - 3/2)^2 - a^2b^2$$

$$= b^2(x_k^2 + 4x_k + 4)\, a^2(y_k^2 - 3y_k + 9/4) - a^2b^2 .$$

Change of P_k^{SE} is: $\Delta P_k^{SE} = P_{k+1}^{SE} - P_k = b^2(2x_k + 3) - 2a^2(y_k - 1).$

Calculate the changes of ΔP_k:

If E is selected,

$$\Delta P_{k+1}^{E} = b^2(2x_k + 5)$$

$$\Delta^2 P_k^{E} = \Delta P_{k+1}^{E} - \Delta P_k^{E} = 2b^2$$

$$\Delta P_{k+1}^{SE} = b^2(2x_k + 5) - 2a^2(y_k - 1)$$

$$\Delta^2 P_k^{SE} = \Delta P_{k+1}^{SE} - \Delta_k^{SE} = 2b^2$$

If SE is selected,

$$\Delta P_{k+1}^{E} = b^2(2x_k + 5)$$

$$\Delta^2 P_k^{E} = \Delta P_{k+1}^{E} - \Delta P_k^{E} = 2b^2$$

$$\Delta P_{k+1}^{SE} = b^2(2x_k + 5) - 2a^2(y_k - 2)$$

$$\Delta^2 P_k^{SE} = \Delta P_{k+1}^{SE} - \Delta P_k^{SE} = 2(a^2 + b^2)$$

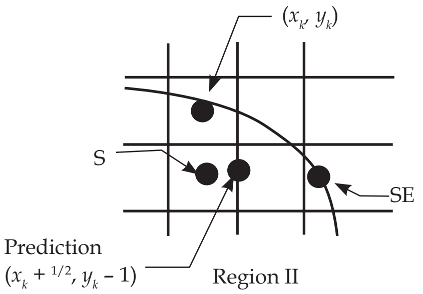
$$\Delta P^{SE}_{k+1} = b^2(2x_k + 5) - 2a^2(y_k - 2)$$
$$\Delta^2 P^{SE}_{k} = \Delta P^{SE}_{k+1} - \Delta P^{SE}_{k} = 2(a^2 + b^2)$$

Initial values:

$$x_0 = 0,\ y_0 = b,\ P_0 = b^2 + \tfrac{1}{4}a^2\ (1 - 4b)$$
$$\Delta P^{E}_{0} = 3b^{\ 2},\ \Delta P^{SE}_{0} = 3b^2 - 2a^2\ (b - 1).$$

In region II (dy/dx < –1), all calculations are similar to that in region we except that y is decremented in each step.

Figure 5.4: Region II Prediction



Source: http://winnyefanho.net/research/MEA.pdf

y is always decremented in each step, i.e. $y_{k+1} = y_k - 1$.

$x_{k+1} = x_k$ if S is selected, or $x_{k+1} = x_k + 1$ if SE is selected.

$$P_k = f(x_k + \tfrac{1}{2},\ y_k - 1)$$
$$= b^2(x_k + \tfrac{1}{2})^2 + a^2(y_k - 1)^2 - a^2b^2$$
$$= b^2(x_k^2 + x_k + \tfrac{1}{4}) + a^2(y_k^2 - 2y_k + 1) - a^2b^2.$$

If $P_k > 0$, select S:

$$P^{S}_{k+1} = f(x_k + \tfrac{1}{2},\ y_k - 2)$$
$$= b^2(x_k + \tfrac{1}{2})^2 + a^2(y_k - 2)^2 - a^2b^2$$
$$= b^2(x_x^2 + x_k + \tfrac{1}{4}) + a^2(y_k^2 - 4y_k + 4) - a^2b^2.$$

Change of P^{S}_{k} is: $\Delta P^{S}_{k} = P^{S}_{k+1} - P_k = a^2(3 - 2y_k).$

If $P_k < 0$, select SE:

$$P^{SE}_{k+1} = f(x_k + 3/2,\ y_k - 2)$$
$$= b^2(x_k + 3/2)^2 + a^2(y_k - 2)^2 - a^2b^2$$
$$= b^2(x_k^2 + 3x_k + 9/4) + a^2(y_k^2 - 4y_k + 4) - a^2b^2.$$

Change of P^{SE}_{k} is: $\Delta P^{SE}_{k} = P^{SE}_{k+1} - P_k = 2b^2(x_k + 1) + a^2(3 - 2y_k)$.

Calculate the changes of ΔP_k:

If S is selected,

$$\Delta P^{S}_{k+1} = a^2(5 - 2y_k)$$
$$\Delta^2 P^{S}_{k} = \Delta P^{S}_{k+1} - \Delta P^{S}_{k} = 2a^2$$

Notes

$$\Delta P^{SE}_{k+1} = 2b^2(x_k + 1) + a^2(5 - 2y_k)$$
$$\Delta^2 P^{SE}_k = \Delta P^{SE}_{k+1} - \Delta P^{SE}_k = 2a^2.$$

If SE is selected,

$$\Delta P^{S}_{k+1} = a^2(5 - 2y_k)$$
$$\Delta^2 P^{S}_k = \Delta P^{S}_{k+1} - \Delta P^{S}_k = 2a^2$$
$$\Delta P^{SE}_{k+1} = 2b^2(2x_k + 2) - a^2(5 - 2y_k)$$
$$\Delta^2 P^{SE}_k = \Delta P^{SE}_{k+1} - \Delta P^{SE}_k = 2(a^2 + b^2).$$

Determine the boundary between region I and II:

$$\text{Set } f(x, y) = 0,\ \frac{dy}{dx} = \frac{-bx}{a^2\sqrt{1 - x^2/a^2}}$$

when $dy/dx = -1$, $x = x = \dfrac{a^2}{\sqrt{a^2 + b^2}}$ and $y = \dfrac{b^2}{\sqrt{a^2 + b^2}}$.

At region I, $dy/dx > -1$, $x < \dfrac{a^2}{\sqrt{a^2 + b^2}}$ and $y > \dfrac{b^2}{\sqrt{a^2 + b^2}}$, therefore,

$$\Delta P^{SE}_k < b^2\left(\frac{2a^2}{\sqrt{a^2 + b^2}} + 3\right) - 2a^2\left(\frac{b^2}{\sqrt{a^2 + b^2}} - 1\right) = 2a^2 + 3b^2.$$

Initial values at region II:

$$x_0 = \frac{a^2}{\sqrt{a^2 + b^2}} \quad \text{and} \quad y_0 = \frac{b^2}{\sqrt{a^2 + b^2}}$$

$$\Delta P_0^{E} = b^2(2x_0 + 3)$$
$$\Delta P_0^{SE} = 2a^2 + 3b^2$$

5.2.1 Implementation of the Algorithm

The algorithm described above shows how to obtain the pixel coordinates in the first quarter only. The ellipse centre is assumed to be at the origin. So, in reality, the pixel coordinates in other quarters can be simply obtained by use of the symmetric characteristics of an ellipse.

For a pixel (x, y) in the first quarter, the corresponding pixels in other three quarters are (x, –y), (–x, y) and (–x, –y) respectively. If the centre is at (x_C, y_C), all calculated coordinate (x, y) should be adjusted by adding the offset (x_C, y_C).

A function PlotEllipse() is defined as follows:

```
PlotEllipse ($x_C$, $y_C$, x, y)
putpixel($x_C$ +x, $y_C$ +y)
putpixel($x^C$ +x, $y_C$ –y)
putpixel($x_C$ –x, $y_C$ +y)
putpixel($x_C$ –x, $y_C$ –y)
end PlotEllipse
```

5.2.2 C Code to Draw an Ellipse

```
//A basic algorithm to draw an N segments ellipse on a window
// hWnd +> Handle to the window where the ellipse will be drawn
// hDC +> Device context to the window where the ellipse will be drawn
// nSeg +> A global variable contains the number of segments or points linked each other to form the ellipse
void MyDrawEllipse(HWND hWnd, HDC hDC)
{
  double x, y, _x, _y, __x, __y, dx, dy, z, wx, hy;
  RECT rc;
  //Prepare objects and data
  Graphics g(hDC);
  Pen p(Color(0xff, 0, 0), 1.0); //Red
  GetClientRect(hWnd, &rc);
  z = 0.99; //Point coordinate affinity
  dx = double(rc.right) / 2.0 - 1.0; //Half window height
  dy = double(rc.bottom) / 2.0 - 1.0; //Half window width
  wx = double(rc.right) / 2.0; //Ellipse center
  hy = double(rc.bottom) / 2.0; //Ellipse center
  for(int i = 0; i < int(nSeg); i++) {
    x = wx * sin((double(i) / double(nSeg)) * (pi*2.0));
    y = hy * cos((double(i) / double(nSeg)) * (pi*2.0));
    if(i > 0) {
      //Draw a line connecting the last point to the one being calculated now
      g.DrawLine(&p, int(dx+_x+z), int(dy+-_y+z), int(dx+x+z), int(dy+-y+z));
    } else {
      //Save the first point coordinate to link it with the last one
      __x = x;
      __y = y;
    }
    //Save the actual point coordinate to link it with the next one
    _x = x;
    _y = y;
  }
  //Draw a line between the last point and the first one
  g.DrawLine(&p, int(dx+x+z), int(dy+-y+z), int(dx+__x+z), int(dy+-__y+z));
```

Notes

```
    /*
    //I use the GDI+ DrawEllipse function to compare my algorithm with
    //the one used by GDI+. Only the visual result is important for me now!
    p.SetColor(Color(0, 0xff, 0));
    g.DrawEllipse(&p, 0, 0, rc.right-1, rc.bottom-1);
    */
}
```

Self Assessment

Fill in the blanks:

3. .... is a method for drawing ellipses in computer graphics.
4. The advantage of midpoint ellipse algorithm method is that only .... operations are required in the program loops.
5. The ellipse centre is assumed to be at the .... in implementation of the algorithm.
6. The .... coordinates in other quarters can be simply obtained by use of the symmetric characteristics of an ellipse.

5.3 Ellipse Area, Sector Area and Segment Area

5.3.1 Ellipse Area

Consider an ellipse that is centered at the origin, with its axes aligned to the coordinate axes. If the semi-axis length along the x-axis is A, and the semi-axis length along the y-axis is B, then the ellipse is defined by a locus of points that satisfy the implicit polynomial equation.

$$\frac{x^2}{A^2}+\frac{y^2}{B^2}=1$$

The same ellipse can be defined parametrically by:

$$\left.\begin{array}{l} x = A\cdot\cos(t) \\ y = B\cdot\sin(t) \end{array}\right\} \quad 0 \le t \le 2\pi \quad .$$

The area of such an ellipse can be found using the parameterized form with the Gauss-Green formula:

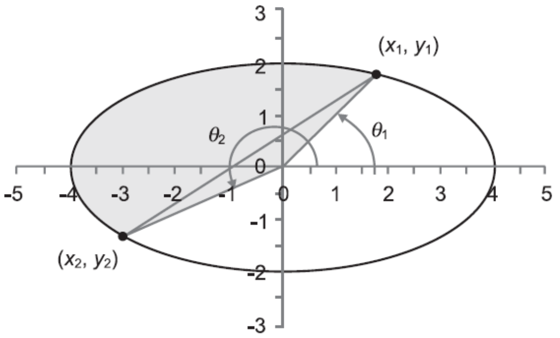
$$\begin{aligned} \text{Area} &= \frac{1}{2}\int_A^B \left[x(t)\cdot y'(t) - y(t)\cdot x'(t)\right]dt \\ &= \frac{1}{2}\int_0^{2\pi} A\cdot\cos(t)\cdot B\cdot\cos(t) - B\cdot\sin(t).(-A)\cdot\sin(t)\big]dt \\ &= \frac{A\cdot B}{2}\int_0^{2\pi}\cos^2(t)+\sin^2(t)\big]dt = \frac{A\cdot B}{2}\int_0^{2\pi}dt \\ &= \pi\cdot A\cdot B \end{aligned}$$

5.3.2 Ellipse Sector Areas

Ellipse sector between two points (x_1,y_1) and (x_2, y_2) on the ellipse is defined as the area that is swept out by a vector from the origin to the ellipse, beginning at (x_1, y_1), as the vector travels along the ellipse in a counter-clockwise direction from (x_1, y_1) to (x_2, y_2). An example is shown in Figure 5.5. The Gauss-Green formula can also be used to determine the area of such an ellipse sector.

$$\text{Sector Area } = \frac{A \cdot B}{2}\int_{\theta_1}^{\theta_2} dt$$
$$= \frac{(\theta_2 - \theta_1) \cdot A \cdot B}{2}.$$

Figure 5.5: Ellipse Sector Areas Example



Source: http://arxiv.org/pdf/1106.3787.pdf

The area of an ellipse sector can be determined with the Gauss-Green formula, using the parametric angles θ_1 and θ_2.

The parametric angle θ_1 that is formed between the x-axis and a point (x, y) on the ellipse is found from the ellipse parameterizations:

$$x = A \cdot \cos(\theta) \Rightarrow \theta = \cos^{-1}(x/A)$$
$$y = A \cdot \sin(\theta) \Rightarrow \theta = \sin^{-1}(y/B).$$

The implicit polynomial form is

$$\frac{x^2}{4^2} + \frac{y^2}{2^2} = 1.$$

Suppose the point (x_1, y_1) is at $(4/\sqrt{5}, 4/\sqrt{5})$. The point is on the ellipse, since

$$\frac{(4/\sqrt{5})^2}{4^2} + \frac{(4/\sqrt{5})^2}{2^2} = \frac{4^2/5}{4^2} + \frac{4^2/5}{2^2} = \frac{1}{5} + \frac{4}{5} = 1$$

A line segment from the origin to $(4/\sqrt{5}, 4/\sqrt{5})$ forms an angle with the x-axis

of $\pi/4(\approx 0.7485398)$. However, the ellipse parametric angle to the same point is:

$$\theta = \cos^{-1}\left(\frac{4/\sqrt{5}}{4}\right) = \cos^{-2}\left(\frac{1}{\sqrt{5}}\right) \approx 1.10715$$

Notes

The same angle can also be found from the parametric equation for y:

$$\theta = \sin^{-1}\left(\frac{4/\sqrt{5}}{2}\right) = \sin^{-2}\left(\frac{2}{\sqrt{5}}\right) \approx 1.10715$$

As we can see the angle found by using the parametric equations does not match the geometric angle to the point that defines the angle.

Relations for finding the parametric angle that corresponds to a given point (x, y) on the ellipse $x^2/A^2 + y^2/B^2 = 1$ is shown in Table 5.1.

Table 5.1: Relations for Finding the Parametric Angle that Corresponds to a Given Point (x, y) on the Ellipse $x^2/A^2 + y^2/B^2 = 1$

Quadrant II ($x < 0$ and $y \geq 0$) $\theta = \arccos(x/A)$ $= \pi - \arcsin(y/B)$	Quadrant I ($x \geq 0$ and $y \geq 0$) $\theta = \arccos(x/A)$ $= \arcsin(y/B)$
Quadrant III ($x < 0$ and $y < 0$) $\theta = 2\pi - \arccos(x/A)$ $= \pi - \arcsin(y/B)$	Quadrant IV ($x \geq 0$ $y < 0$) $\theta = 2\pi \arccos(x/A)$ $= 2\pi + \arcsin(y/B)$

Source: http://arxiv.org/pdf/1106.3787.pdf

5.3.3 Ellipse Segment Area

The ELLIPSE_SEGMENT algorithm is given as:

1. $$\theta_1 = \begin{cases} \arccos(x_1/A) & y_1 \geq 0 \\ 2\pi - \arccos(x_1/A), y_1 < 0 \end{cases}$$

$$\theta_2 = \begin{cases} \arccos(x_2/A) & y_2 \geq 0 \\ 2\pi - \arccos(x_1/A), y_2 < 0 \end{cases}$$

2. $$\theta_1 = \begin{cases} \theta_1, & \theta < \theta_2 \\ \theta_1 - 2\pi, \theta_1 > \theta_2 \end{cases}$$

3. $$\text{Area} = \frac{(\theta_2 - \theta_1)A \cdot B}{2} + \frac{\sin(\theta_2 - \theta_1 - \pi)}{2} \cdot |x_1 \cdot y_2 - x_2 \cdot y_1|$$

Where the ellipse implicit polynomial equation is $\frac{x^2}{A^2} + \frac{y^2}{B^2} = 1$

A > 0 is the semi-axis length along the x-axis

B > 0 is the semi-axis length along the y-axis

(x_1, y_1) is the first given point on the ellipse

(x_2, y_2) is the second given point on the ellipse

Self Assessment

Fill in the blanks:

7. The area of ellipse can be found using the parameterized form with the ....

8. The Gauss-Green formula can also be used to determine the area of a ....

9. The area of an ellipse sector can be determined with the Gauss-Green formula, using the parametric angles .... and ....

5.4 Summary

- It is possible to generalize the algorithm to handle ellipses (of which circles are a special case).
- These algorithms involve calculating a full quadrant of the ellipse, as opposed to an octant as explained above, since non-circular ellipses lack the x-y symmetry of a circle.
- Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics. This method is modified from Bresenham's algorithm.
- An elliptic sector is a region bounded by an arc and line segments connecting the center of the ellipse and the endpoints of the arc.

5.5 Keywords

Algorithm: In mathematics and computer science, an algorithm is a step-by-step procedure for calculations.

Ellipse: In mathematics, an ellipse is a plane curve that results from the intersection of a cone by a plane in a way that produces a closed curve.

Midpoint ellipse algorithm: Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics.

Tangent: In geometry, the tangent line (or simply the tangent) to a plane curve at a given point is the straight line that "just touches" the curve at that point.

5.6 Review Questions

1. What is ellipse?
2. Briefly explain the Bresenham's ellipse drawing algorithm.
3. Discuss ellipse inside a specified rectangle with an example.
4. What is midpoint ellipse algorithm?
5. Elucidate the implementation of the midpoint ellipse algorithm.
6. How C code is use to draw an ellipse?
7. Define ellipse area.
8. Discuss about area intersecting ellipses.
9. Discuss the Gauss-Green formula to determine the area of an ellipse sector.
10. What is the ellipse segment area?

Answers: Self Assessment

1.	False	2.	True
3.	Midpoint ellipse algorithm	4.	Addition

Notes

5. Origin

6. Pixel

7. Gauss-Green formula

8. Ellipse sector

9. θ_1, θ_2

5.7 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. "*Computer Graphics*" Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Angel, Edward. "*Interactive Computer Graphics*" Pearson Education.

Krishnamurthy, N. "*Introduction to Computer Graphics*" Tata McGraw Hill.

Online links

www.ddegjust.ac.in/studymaterial/mca-3/ms-13.pdf

citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.6985...pdf

gyan.frcrce.ac.in/lm/sem6/cg.pdf

wscg.zcu.cz/wscg2001/Papers_2001/R18.pdf

Unit 6: Implementing Polygon Filling Algorithm

CONTENTS

Objectives

After studying this unit, you will be able to:

- Define scan-line fill algorithm
- Discuss about the boundary-fill algorithm
- Explain the flood fill algorithm

Introduction

Polygon is a plane figure which is a closed contour of straight lines. A basic primitive in the graphical representation of objects. Polygon fill is a series of ordered planar vertices connected to form an enclosed area. This area is then completely rendered using a specified color or texture. In this unit, you will learn about implementing polygon filling algorithms. First, will be covered scan-line algorithm for polygon filling. Later on in the chapter boundary fill algorithm will be discussed and finally you will learn about flood fill algorithm.

6.1 Scan-line Fill Algorithm

Polygon filling means given the edges defining a polygon, and a color for the polygon, we need to fill all the pixels inside the polygon. There are three different algorithms for this:

1. Scan-line fill
2. Boundary fill
3. Flood fill

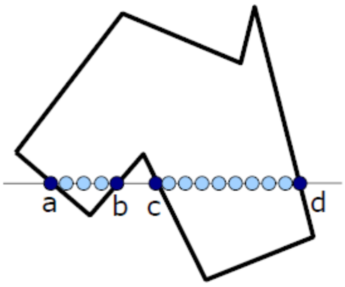
6.1.1 Basic Scan-line Fill Algorithm

The scanline fill algorithm is an ingenious way of filling in irregular polygons. For each scan-line:

Notes

- Locate the intersection of the scan-line with the edges (y=ys)
- Sort the intersection points from left to right.
- Draw the interiors intersection points pairwise. (a-b), (c-d)

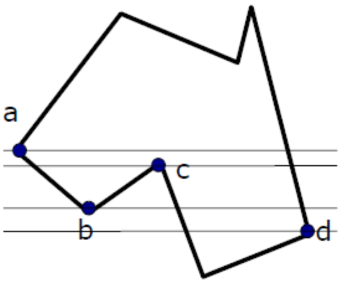
Figure 6.1: Basic Scan-line Polygon Fill



Source: http://www.cecs.csulb.edu/~pnguyen/cecs449/lectures/fillalgorithm.pdf

Notes So you will notice the problem with corners. a, b, c and d are intersected by 2 line segments each. Count b, c twice but a and d once.

Figure 6.2: Problem in Scan-fill

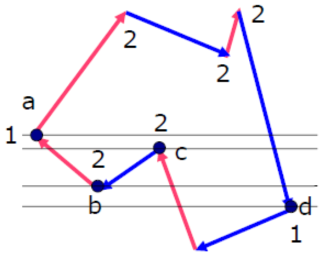


Source: http://www.cecs.csulb.edu/~pnguyen/cecs449/lectures/fillalgorithm.pdf

Solution

Make a clockwise or counter-clockwise traversal on edges. Check if y is monotonically increasing or decreasing. If direction changes, double intersection, otherwise single intersection.

Figure 6.3: Clockwise Traversal on Edges



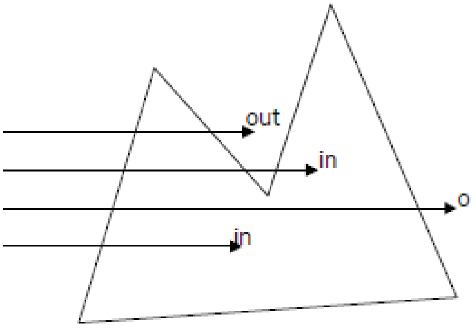
Source: http://www.cecs.csulb.edu/~pnguyen/cecs449/lectures/fillalgorithm.pdf

6.1.2 Basic Algorithm

Notes

- Assume scan line start from the left and is outside the polygon.
- When intersect an edge of polygon, start to color each pixel (because now we're inside the polygon), when intersect another edge, stop coloring.
- Odd number of edges: inside
- Even number of edges: outside

Figure 6.4: Using Basic Scan Fill



Source: http://www.cecs.csulb.edu/~pnguyen/cecs449/lectures/fillalgorithm.pdf

A scan-line is a line of constant y value, i.e., y=c, where c lies within our drawing region, e.g., the window on our computer screen.

⚠ *Caution* When filling a polygon, you will most likely have a set of vertices, indicating the x and y Cartesian coordinates of each vertex of the polygon.

The following steps should be taken to turn your set of vertices into a filled polygon.

Initializing All of the Edges

The first thing that needs to be done is determine how the polygon's vertices are related. The all_edges table will hold this information. Each adjacent set of vertices (the first and second, second and third, ..., last and first) defines an edge. For each edge, the following information needs to be kept in a table:

1. The minimum y value of the two vertices.
2. The maximum y value of the two vertices.
3. The x value associated with the minimum y value.
4. The slope of the edge.

The slope of the edge can be calculated from the formula for a line:

y = mx + b;

where m = slope, b = y-intercept,

y_0 = maximum y value,

y_1 = minimum y value,

Notes

x_0 = maximum x value,

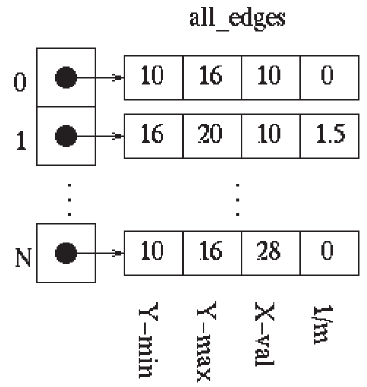
x_1 = minimum x value

The formula for the slope is as follows:

$$m = (y_0 - y_1) / (x_0 - x_1).$$

For example, the edge values can be kept as give below. Here, N is equal to the total number of edges – 1 and each index is kept into the all_edges array which contains a pointer to the array of edge values.

Figure 6.5: Edge Values



Source: http://www.cs.rit.edu/~icss571/filling/how1.gif

Initializing the Global Edge Table

The global edge table will be used to keep track of the edges that are still needed to complete the polygon. Edges with the same minimum y values are sorted on minimum x values as follows:

1. Place the first edge with a slope that is not equal to zero in the global edge table.
2. If the slope of the edge is zero, do not add that edge to the global edge table.
3. For every other edge, start at index 0 and increase the index to the global edge table once each time the current edge's y value is greater than the edge value at the current index in the global edge table.
4. Next, increase the index to the global edge table once each time the current edge's x value is greater than and the y value is less than or equal to the edge value at the current index in the global edge table.

If the index, at any time, is equal to the number of edges currently in the global edge table, do not increase the index. Place the edge information for minimum y value, maximum y value, x value, and 1/m in the global edge table at the index.

Initializing Parity

The initial parity is even since no edges have been crossed yet.

Initializing the Scan-Line

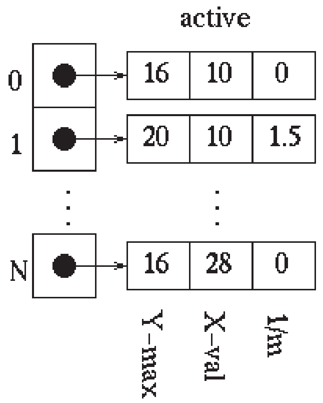
The initial scan-line is equal to the lowest y value for all of the global edges.

Initializing the Active Edge Table

The active edge table will be used to keep track of the edges that are intersected by the current scan-line. This should also contain ordered edges. This is initially set up as follows:

- Since the global edge table is ordered on minimum y and x values, search, in order, through the global edge table and, for each edge found having a minimum y value equal to the current scan-line, append the edge information for the maximum y value, x value, and 1/m to the active edge table. Do this until an edge is found with a minimum y value greater than the scan line value.
- The active edge table will now contain ordered edges of those edges that are being filled as such:

Figure 6.6: Active Edge Values



Source: http://www.cs.rit.edu/~icss571/filling/how2.gif

Filling the Polygon

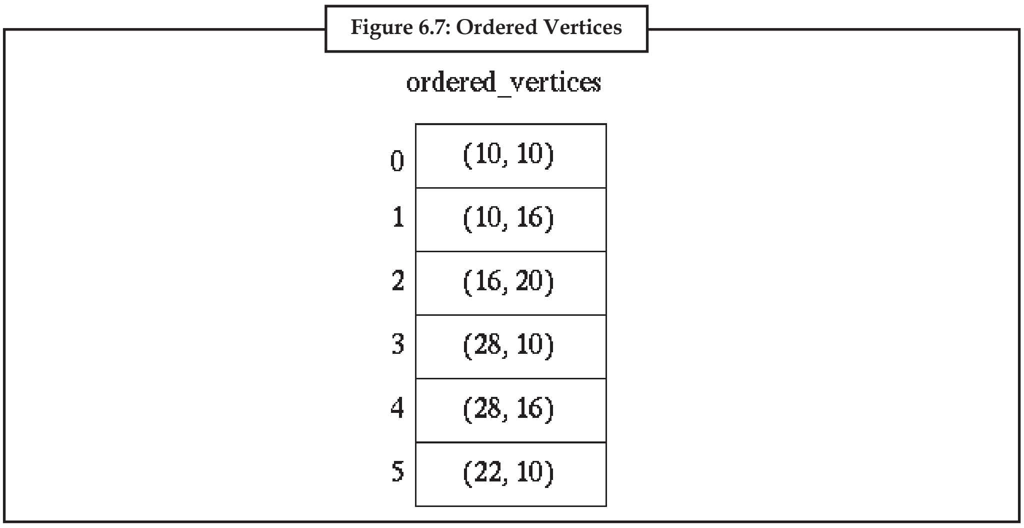
Filling the polygon involves deciding whether or not to draw pixels, adding to and removing edges from the active edge table, and updating x values for the next scan-line. Starting with the initial scan-line, until the active edge table is empty, do the following:

1. Draw all pixels from the x value of odd to the x value of even parity edge pairs.
2. Increase the scan line by 1.
3. Remove any edges from the active edge table for which the maximum y value is equal to the scan line.
4. Update the x value for for each edge in the active edge table using the formula $x_1 = x_0 + 1/m$.
5. Remove any edges from the global edge table for which the minimum y value is equal to the scan line and place them in the active edge table.
6. Reorder the edges in the active edge table according to increasing x value. This is done in case edges have crossed.

Example: The following example of scan-line polygon filling will be helpful to better understand the algorithm. Initially, each vertice of the polygon is given in the form of (x,y) and is in an ordered array as such:

Notes

Figure 6.7: Ordered Vertices



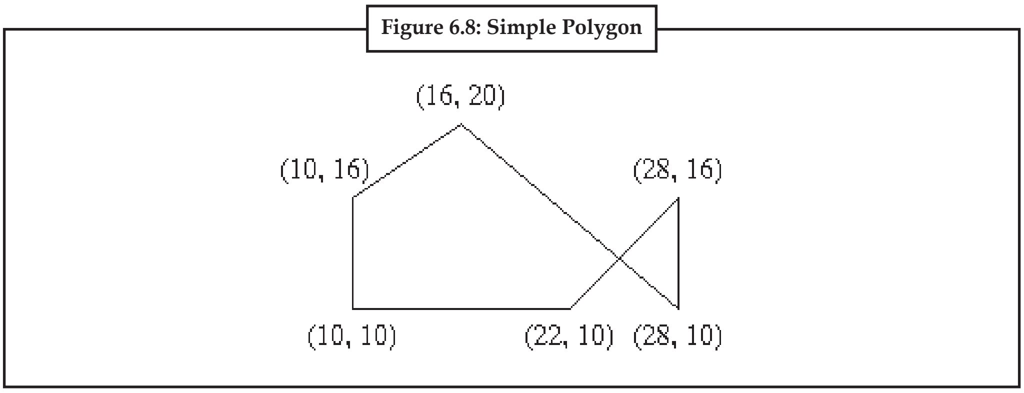
ordered_vertices

0	(10, 10)
1	(10, 16)
2	(16, 20)
3	(28, 10)
4	(28, 16)
5	(22, 10)

Source: http://www.cs.rit.edu/~icss571/filling/ordered_verts.gif

Unfilled, the polygon would look like this to the human eye:

Figure 6.8: Simple Polygon



Source: http://www.cs.rit.edu/~icss571/filling/simple1.gif

Now, we will walk through the steps of the algorithm to fill in the polygon.

Initializing All of the Edges

We want to determine the minimum y value, maximum y value, x value, and 1/m for each edge and keep them in the all_edges table. We determine these values for the first edge as follows:

Y-min:

Since the first edge consists of the first and second vertex in the array, we use the y values of those vertices to choose the lesser y value. In this case it is 10.

Y-max:

In the first edge, the greatest y value is 16.

X-val:

Since the x value associated with the vertex with the highest y value is 10, 10 is the x value for this edge.

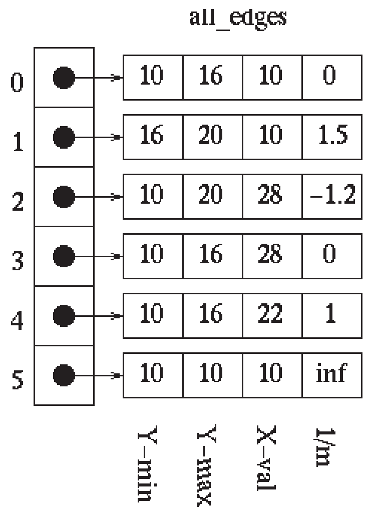
1/m:

Using the given formula, we get (10-10)/(16-10) for 1/m.

Notes

The edge value results are in the form of Y-min, Y-max, X-val, Slope for each edge array pointed to in the all_edges table. As a result of calculating all edge values, we get the following in the all_edges table.

Figure 6.9: All Edges



all_edges

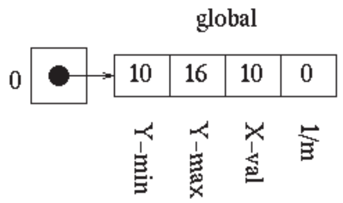
	Y–min	Y–max	X–val	1/m
0	10	16	10	0
1	16	20	10	1.5
2	10	20	28	–1.2
3	10	16	28	0
4	10	16	22	1
5	10	10	10	inf

Source: http://www.cs.rit.edu/~icss571/filling/all_edges.gif

Initializing the Global Edge Table

We want to place all the edges in the global edge table in increasing y and x values, as long as slope is not equal to zero. For the first edge, the slope is not zero so it is placed in the global edge table at index=0.

Figure 6.10: Global Edge 1



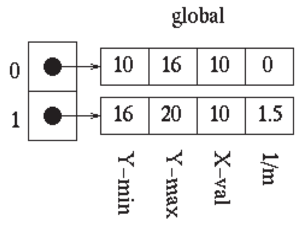
global

	Y–min	Y–max	X–val	1/m
0	10	16	10	0

Source: http://www.cs.rit.edu/~icss571/filling/global2.gif

For the second edge, the slope is not zero and the minimum y value is greater than that at zero, so it is placed in the global edge table at index=1.

Figure 6.11: Global Edge 2



global

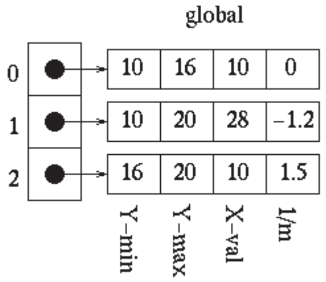
	Y–min	Y–max	X–val	1/m
0	10	16	10	0
1	16	20	10	1.5

Source: http://www.cs.rit.edu/~icss571/filling/global3.gif

Notes

For the third edge, the slope is not zero and the minimum y value is equal the edge's at index zero and the x value is greater than that at index 0, so the index is increased to 1. Since the third edge has a lesser minimum y value than the edge at index 2 of the global edge table, the index for the third edge is not increased again. The third edge is placed in the global edge table at index=1.

Figure 6.12: Global Edge 3

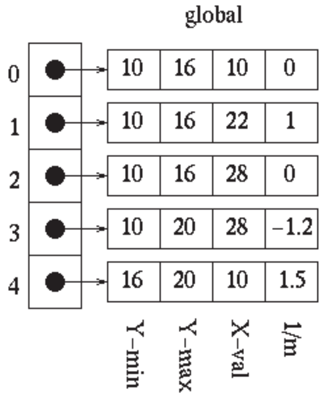


	Y–min	Y–max	X–val	1/m
0	10	16	10	0
1	10	20	28	–1.2
2	16	20	10	1.5

Source: http://www.cs.rit.edu/~icss571/filling/global4.gif

We continue this process until we have the following:

Figure 6.13: Global Edge Final Table



	Y–min	Y–max	X–val	1/m
0	10	16	10	0
1	10	16	22	1
2	10	16	28	0
3	10	20	28	–1.2
4	16	20	10	1.5

Source: http://www.cs.rit.edu/~icss571/filling/global1.gif

Notice that the global edge table has only five edges and the all_edges table has six. This is due to the fact that the last edge has a slope of zero and, therefore, is not placed in the global edge table.

Initializing Parity

Parity is initially set to even.

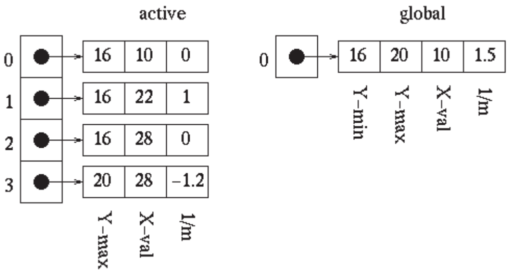
Initializing the Scan-Line

Since the lowest y value in the global edge table is 10, we can choose 10 as our initial scan-line.

Initializing the Active Edge Table

Since our scan-line value is 10, we choose all edges which have a minimum y value of 10 to move to our active edge table. This results in the following.

Figure 6.14: Active Edge Table



Source: http://www.cs.rit.edu/~icss571/filling/active1.gif

Filling the Polygon

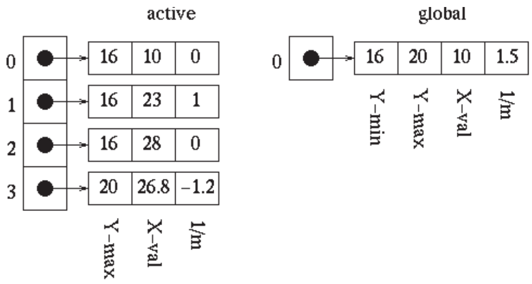
Starting at the point (0,10), which is on our scan-line and outside of the polygon, will want to decide which points to draw for each scan-line.

Scan-line = 10:

Once the first edge is encountered at x = 10, parity = odd. All points are drawn from this point until the next edge is encountered at x = 22. Parity is then changed to even. The next edge is reached at x = 28, and the point is drawn once on this scan-line due to the special parity case. We are now done with this scan-line.

First, we update the x values in the active edge table using the formula x1 = x0 + 1/m to get the following:

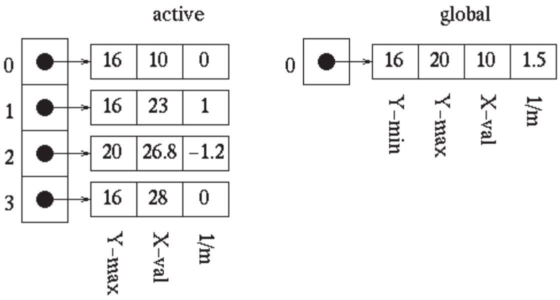
Figure 6.15: Active Edge Table Scan line 10



Source: http://www.cs.rit.edu/~icss571/filling/active2.gif

The edges then need to be reordered since the edge at index 3 of the active edge table has a lesser x value than that of the edge at index 2. Upon reordering, we get:

Figure 6.16: Reordered

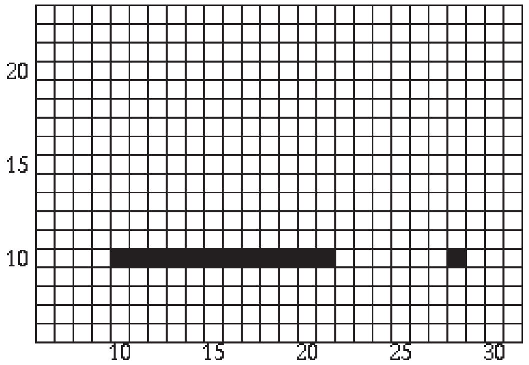


Source: http://www.cs.rit.edu/~icss571/filling/active3.gif

Notes

The polygon is now filled as follows:

Figure 6.17: Current Polygon

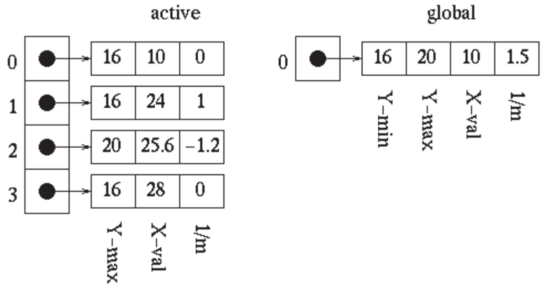


Source: http://www.cs.rit.edu/~icss571/filling/simple3.gif

Scan-line = 11:

Once the first edge is encountered at x = 10, parity = odd. All points are drawn from this point until the next edge is encountered at x = 23. Parity is then changed to even. The next edge is reached at x = 27 and parity is changed to odd. The points are then drawn until the next edge is reached at x = 28. We are now done with this scan-line. Upon updating the x values, the edge tables are as follows:

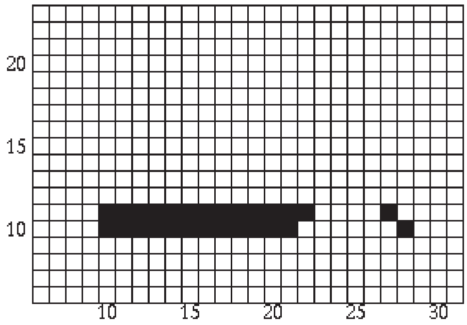
Figure 6.18: Active Table for Scan Line 11



Source: http://www.cs.rit.edu/~icss571/filling/active4.gif

It can be seen that no reordering of edges is needed at this time. The polygon is now filled as follows:

Figure 6.19: Current Polygon

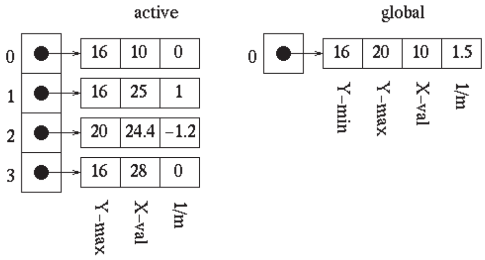


Source: http://www.cs.rit.edu/~icss571/filling/simple4.gif

Scan-line = 12:

Once the first edge is encountered at x = 10, parity = odd. All points are drawn from this point until the next edge is encountered at x = 24. Parity is then changed to even. The next edge is reached at x = 26 and parity is changed to odd. The points are then drawn until the next edge is reached at x = 28. We are now done with this scan-line. Updating the x values in the active edge table gives us:

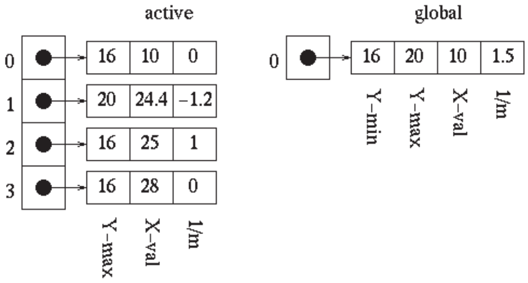
Figure 6.20: Active Edge Table at Scan Line 12



Source: http://www.cs.rit.edu/~icss571/filling/active5.gif

We can see that the active edges need to be reordered since the x value of 24.4 at index 2 is less than the x value of 25 at index 1. Reording produces the following:

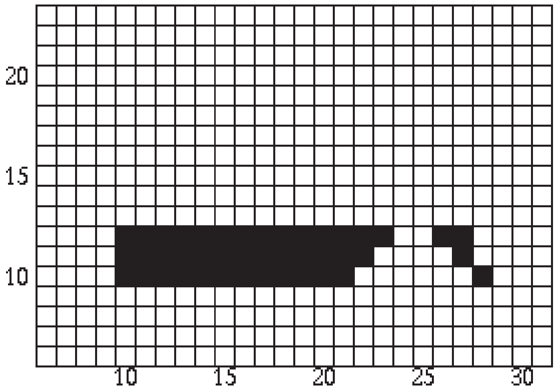
Figure 6.21: Reordered



Source: http://www.cs.rit.edu/~icss571/filling/active6.gif

The polygon is now filled as follows:

Figure 6.22: Current Polygon



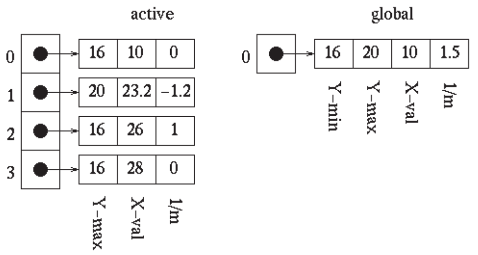
Source: http://www.cs.rit.edu/~icss571/filling/simple5.gif

Notes

Scan-line = 13:

Once the first edge is encountered at x = 10, parity = odd. All points are drawn from this point until the next edge is encountered at x = 25 Parity is then changed to even. The next edge is reached at x = 25 and parity is changed to odd. The points are then drawn until the next edge is reached at x = 28. We are now done with this scan-line. Upon updating the x values for the active edge table, we can see that the edges do not need to be reordered.

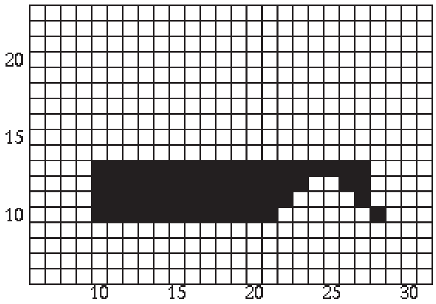
Figure 6.23: Active Edge Table at Scan Line 13



Source: http://www.cs.rit.edu/~icss571/filling/active7.gif

The polygon is now filled as follows:

Figure 6.24: Current Polygon

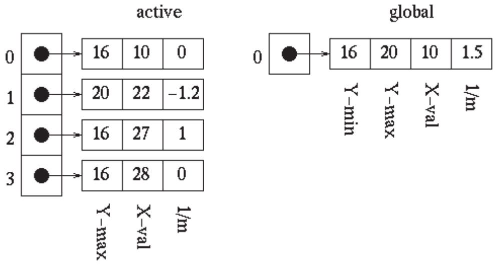


Source: http://www.cs.rit.edu/~icss571/filling/simple6.gif

Scan-line = 14:

Once the first edge is encountered at x = 10, parity = odd. All points are drawn from this point until the next edge is encountered at x = 24. Parity is then changed to even. The next edge is reached at x = 26 and parity is changed to odd. The points are then drawn until the next edge is reached at x = 28. We are now done with this scan-line. Upon updating the x values for the active edge table, we can see that the edges still do not need to be reordered.

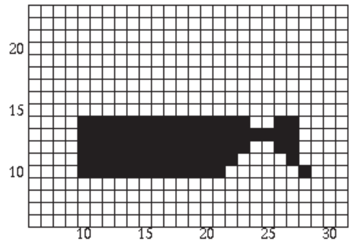
Figure 6.25: Active Edge Table at Scan Line 14



Source: http://www.cs.rit.edu/~icss571/filling/active8.gif

The polygon is now filled as follows:

Figure 6.26: Active Edge Table at Scan Line 14

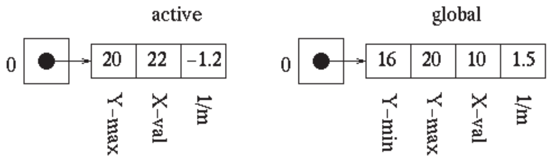


Source: http://www.cs.rit.edu/~icss571/filling/simple7.gif

Scan-line = 15:

Once the first edge is encountered at x = 10, parity = odd. All points are drawn from this point until the next edge is encountered at x = 22. Parity is then changed to even. The next edge is reached at x = 27 and parity is changed to odd. The points are then drawn until the next edge is reached at x = 28. We are now done with this scan-line. Since the maximum y value is equal to the next scan-line for the edges at indices 0, 2, and 3, we remove them from the active edge table. This leaves us with the following:

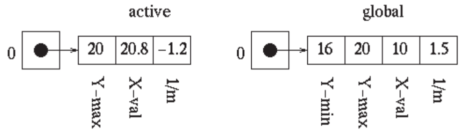
Figure 6.27: Active Edge Table at Scan-line 15



Source: http://www.cs.rit.edu/~icss571/filling/active9.gif

We then need to update the x values for all remaining edges.

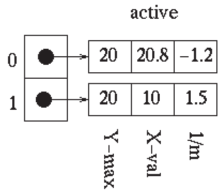
Figure 6.28: Updated Table



Source: http://www.cs.rit.edu/~icss571/filling/active10.gif

Now we can add the last edge from the global edge table to the active edge table since its minimum y value is equal to the next scan-line. The active edge table now look as follows (the global edge table is now empty):

Figure 6.29: Updated Table

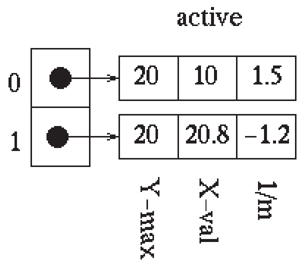


Source: http://www.cs.rit.edu/~icss571/filling/active11.gif

Notes

These edges obviously need to be reordered. After reordering, the active edge table contains the following:

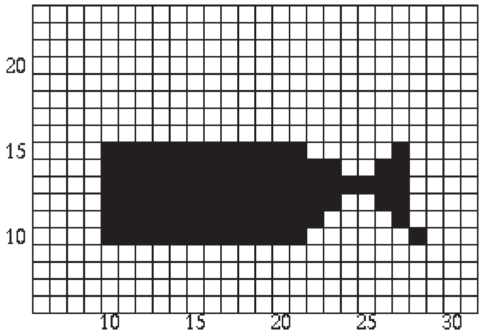
Figure 6.30: Reordered Table



Source: http://www.cs.rit.edu/~icss571/filling/active12.gif

The polygon is now filled as follows:

Figure 6.31: Current Polygon

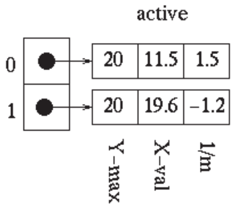


Source: http://www.cs.rit.edu/~icss571/filling/simple8.gif

Scan-line = 16:

Once the first edge is encountered at x = 10, parity = odd. All points are drawn from this point until the next edge is reached at x = 21. We are now done with this scan-line. The x values are updated and the following is obtained:

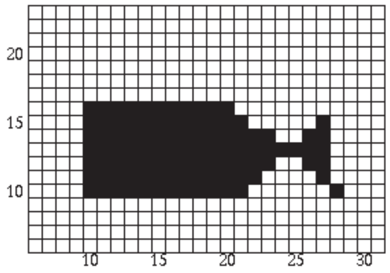
Figure 6.32: Active Edge Table at Scan Line 16



Source: http://www.cs.rit.edu/~icss571/filling/active13.gif

The polygon is now filled as follows:

Figure 6.33: Current Polygon

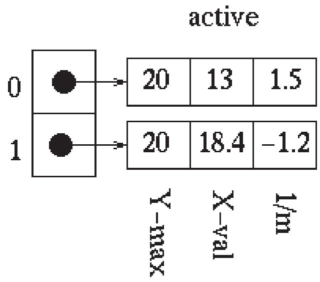


Source: http://www.cs.rit.edu/~icss571/filling/simple9.gif

Scan-line = 17:

Once the first edge is encountered at x = 12, parity = odd. All points are drawn from this point until the next edge is reached at x = 20. We are now done with this scan-line. We update the x values and obtain:

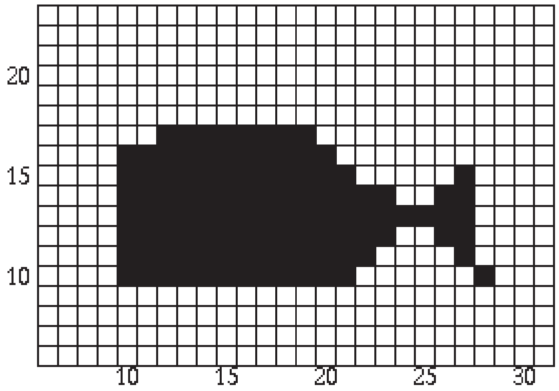
Figure 6.34: Active Edge Table at Scan Line 17



Source: http://www.cs.rit.edu/~icss571/filling/active14.gif

The polygon is now filled as follows:

Figure 6.35: Current Polygon



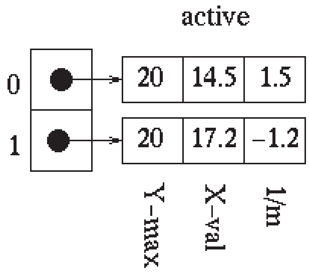
Source: http://www.cs.rit.edu/~icss571/filling/simple10.gif

Notes

Scan-line = 18:

Once the first edge is encountered at x = 13, parity = odd. All points are drawn from this point until the next edge is reached at x = 19. We are now done with this scan-line. Upon updating the x values we get:

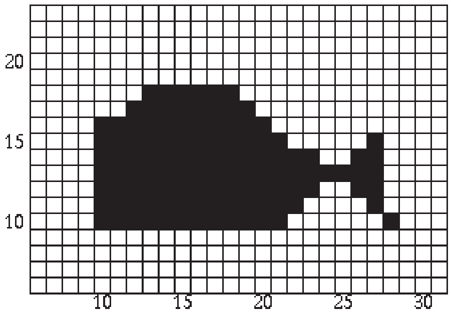
Figure 6.36: Active Edge Table at Scan line 18



Source: http://www.cs.rit.edu/~icss571/filling/active15.gif

The polygon is now filled as follows:

Figure 6.37: Current Polygon

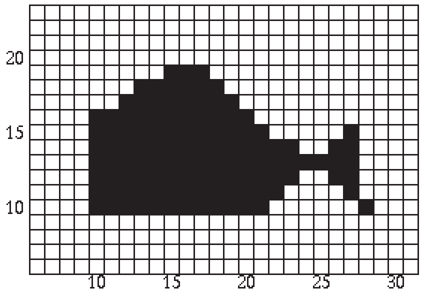


Source: http://www.cs.rit.edu/~icss571/filling/simple11.gif

Scan-line = 19:

Once the first edge is encountered at x = 15, parity = odd. All points are drawn from this point until the next edge is reached at x = 18. We are now done with this scan-line. Since the maximum y value for both edges in the active edge table is equal to the next scan-line, we remove them. The active edge table is now empty and we are now done. The polygon is now filled as follows:

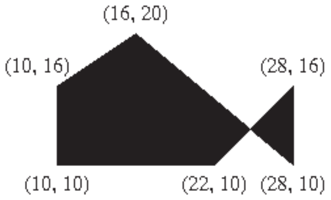
Figure 6.38: Changed Polygon



Source: http://www.cs.rit.edu/~icss571/filling/simple12.gif

Now that we have filled the polygon, let’s see what it looks like to the naked eye:

Figure 6.39: Final Polygon



Source: http://www.cs.rit.edu/~icss571/filling/simple2.gif

C program for Scan Line Polygon Filling Algorithm

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
main()
{
int n,i,j,k,gd,gm,dy,dx;
int x,y,temp;
int a[20][2],xi[20];
float slope[20];
clrscr();
printf(“\n\n\tEnter the no. of edges of polygon : “);
scanf(“%d”,&n);
printf(“\n\n\tEnter the cordinates of polygon :\n\n\n “);
for(i=0;i<n;i++)
{
printf(“\tX%d Y%d : “,i,i);
scanf(“%d %d”,&a[i][0],&a[i][1]);
}
a[n][0]=a[0][0];
a[n][1]=a[0][1];
detectgraph(&gd,&gm);
initgraph(&gd,&gm,”c:\\tc\\bgi”);
/*- draw polygon -*/
for(i=0;i<n;i++)
{
line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);
```

Notes

```
}
getch();
for(i=0;i<n;i++)
{
dy=a[i+1][1]-a[i][1];
dx=a[i+1][0]-a[i][0];
if(dy==0) slope[i]=1.0;
if(dx==0) slope[i]=0.0;
if((dy!=0)&&(dx!=0)) /*- calculate inverse slope -*/
{
slope[i]=(float) dx/dy;
}
}
for(y=0;y< 480;y++)
{
k=0;
for(i=0;i<n;i++)
{
if( ((a[i][1]<=y)&&(a[i+1][1]>y)) | |
((a[i][1]>y)&&(a[i+1][1]<=y)))
{
xi[k]=(int)(a[i][0]+slope[i]*(y-a[i][1]));
k++;
}
}
for(j=0;j<k-1;j++) /*- Arrange x-intersections in order -*/
for(i=0;i<k-1;i++)
{
if(xi[i]>xi[i+1])
{
temp=xi[i];
xi[i]=xi[i+1];
xi[i+1]=temp;
}
}
```

```
setcolor(35);
for(i=0;i<k;i+=2)
{
line(xi[i],y,xi[i+1]+1,y);
getch();
}
}
}
```

Self Assessment

Fill in the blanks:

1. .... is a plane figure which is a closed contour of straight lines.
2. Polygon fill is a series of ordered planar vertices connected to form an ....
3. .... means given the edges defining a polygon, and a color for the polygon, we need to fill all the pixels inside the polygon.
4. The .... algorithm is an ingenious way of filling in irregular polygons.

6.2 Boundary Fill Algorithm

It refers to expansion and filling of region until you reach boundary color. Here suppose that the edges of the polygon have already been colored and suppose that the interior of the polygon is to be colored a different color from the edge. Suppose we start with a pixel inside the polygon, then we color that pixel and all surrounding pixels until we meet a pixel that is already.

C Program to implement Boundary Fill Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void fill_right(x,y)
int x , y ;
{
if((getpixel(x,y) != WHITE)&&(getpixel(x,y) != RED))
{
putpixel(x,y,RED);
fill_right(++x,y);
x = x - 1 ;
fill_right(x,y-1);
fill_right(x,y+1);
```

Notes

```
}
delay(1);
}
void fill_left(x,y)
int x , y ;
{
if((getpixel(x,y) != WHITE)&&(getpixel(x,y) != RED))
{
putpixel(x,y,RED);
fill_left(--x,y);
x = x + 1 ;
fill_left(x,y-1);
fill_left(x,y+1);
}
delay(1);
}
void main()
{
int x,y,n,i;
int gd=DETECT,gm;
clrscr();
initgraph(&gd,&gm,"c:\\tc\\bgi");
/*- draw object -*/
line (50,50,200,50);
line (200,50,200,300);
line (200,300,50,300);
line (50,300,50,50);
/*- set seed point -*/
x = 100; y = 100;
fill_right(x,y);
fill_left(x-1,y);
getch();
}
```

Self Assessment

State whether the following statement are true or false:

5. Boundary Fill Algorithm refers to expansion and filling of region until you reach boundary color.

6.3 Flood Fill Algorithm

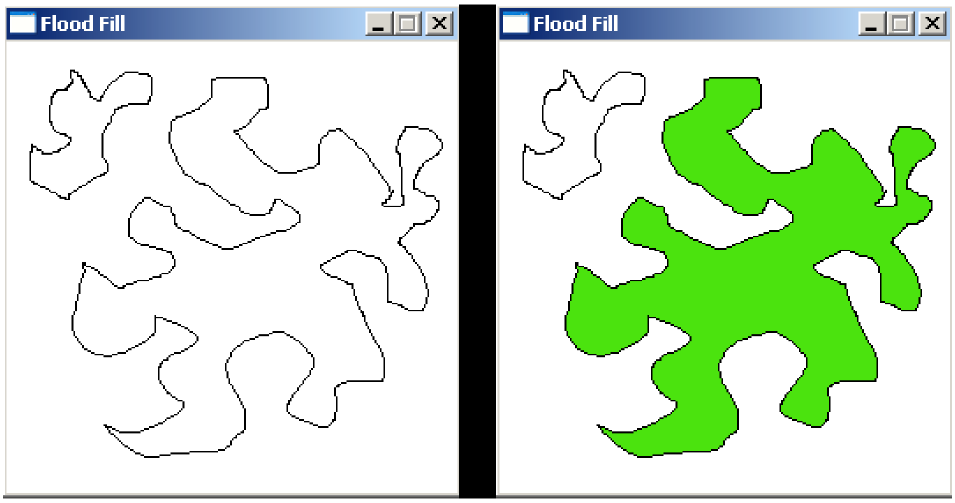
Flood-filling of a region on raster devices is one of the most widely used techniques in the interactive graphical systems. Existing filling algorithms, in general, require significant amount of memory and/or are limited to the specific types of regions or fill styles. It is proved, in particular, that necessary size of work memory for this particular algorithm is proportional to the horizontal dimension of the box containing region to be filled.

Notes The algorithm may be used to fill 4-step and 8-step connected regions both border- and interior-defined.

Did u know? The purpose of Flood Fill is to color an entire area of connected pixels with the same color. It's the Bucket Tool in many painting programs.

Example: The original image is on the left. Then a floodfill was started inside the large shape, and the algorithm gave all pixels inside the shape the new color, leaving its borders and the pixels outside intact.

Figure 6.40: Flood Fill Algorithm



Source: http://lodev.org/cgtutor/images/floodfill.gif

Depending on whether we consider nodes touching at the corners connected or not, we have two variations, Eight-way and Four-way, respectively.

6.3.1 Stack-based Recursive Implementation (Four-way)

One implicitly stack-based (recursive) flood-fill implementation (for a two-dimensional array) goes as follows:

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

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

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

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

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

Though easy to understand, the implementation of the algorithm used above is impractical in languages and environments where stack space is severely constrained.

6.3.2 Alternative Implementations

An explicitly queue-based implementation is shown in pseudo-code below. It is similar to the simple recursive solution, except that instead of making recursive calls, it pushes the nodes into a LIFO queue — acting as a stack — for consumption:

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

1. Set Q to the empty queue.
2. Add node to the end of Q.
3. While Q is not empty:
4. Set n equal to the last element of Q.
5. Remove last element from Q.
6. If the color of n is equal to target-color:
7. Set the color of n to replacement-color.
8. Add west node to end of Q.
9. Add east node to end of Q.
10. Add north node to end of Q.
11. Add south node to end of Q.
12. Return.

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

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

1. Set Q to the empty queue.
2. If the color of node is not equal to target-color, return.
3. Add node to Q.

4. For each element N of Q:
5. If the color of N is equal to target-color:
6. Set w and e equal to N.
7. Move w to the west until the color of the node to the west of w no longer matches target-color.
8. Move e to the east until the color of the node to the east of e no longer matches target-color.
9. For each node n between w and e:
10. Set the color of n to replacement-color.
11. If the color of the node to the north of n is target-color, add that node to Q.
12. If the color of the node to the south of n is target-color, add that node to Q.
13. Continue looping until Q is exhausted.
14. Return.

Self Assessment

Fill in the blanks:

6. ....of a region on raster devices is one of the most widely used techniques in the interactive graphical systems.
7. The algorithm may be used to fill .... and 8-step connected regions both border- and interior-defined.

6.4 Summary

- Polygon filling means given the edges defining a polygon, and a color for the polygon, we need to fill all the pixels inside the polygon.
- Boundary Fill Algorithm refers to expansion and filling of region until you reach boundary color.
- The purpose of Flood Fill is to color an entire area of connected pixels with the same color. It's the Bucket Tool in many painting programs.
- Flood-filling of a region on raster devices is one of the most widely used techniques in the interactive graphical systems.
- The purpose of Flood Fill is to color an entire area of connected pixels with the same color.

6.5 Keywords

Boundary Fill Algorithm: It refers to expansion and filling of region until you reach boundary color.

Flood Fill: Flood fill, also called seed fill, is an algorithm that determines the area connected to a given node in a multi-dimensional array.

Polygon Filling: Polygon filling means given the edges defining a polygon, and a color for the polygon, we need to fill all the pixels inside the polygon.

Polygon: A plane figure with at least three straight sides and angles, and typically five or more.

Scan Line Fill Algorithm: The scanline fill algorithm is an ingenious way of filling in irregular polygons. The algorithm begins with a set of points.

Notes

6.6 Review Questions

1. What is polygon?
2. Write down the meaning of polygon filling.
3. What are the three different algorithms for polygon filling?
4. What is scan line algorithm?
5. What steps should be taken to turn set of vertices into a filled polygon?
6. Write short note on filling the polygon.
7. What are the steps of the algorithm to fill in the polygon?
8. What is boundary-fill algorithm? Explain in detail.
9. Write a C program to implementing the boundary-fill algorithm.
10. What is flood fill algorithm? Discuss the stack-based recursive implementation.

Answers: Self Assessment

1. Polygon
2. Enclosed area
3. Polygon filling
4. Scanline fill
5. True
6. Flood-filling
7. 4-step

6.7 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. "*Computer Graphics*" Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Angel, Edward. "*Interactive Computer Graphics*" Pearson Education.

Krishnamurthy, N. "*Introduction to Computer Graphics*" Tata McGraw Hill.

Online links

www2.tku.edu.tw/~tkjse/2-4/2-4-1.pdf

go4coding.blogspot.com/2012/12/polygon-filling-algorithms.html

cs.brown.edu/stc/summer/2ViewRender/2ViewRender_23.html

eng.harran.edu.tr/~msuzer/files/cg/slides/lec6-AreaFill.pdf

Unit 7: Implementation of Hidden Surface in 2D

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain visibility algorithm categorization
- Discuss the hidden surface removal algorithms

Introduction

In displaying things, there often occurs overlapping, meaning that the objects which are closer are evident, and those behind them are non-visible, because they emerge as hidden behind them. There have been created many, rather complex algorithms for sorting this issue. Their aim is to find out objects whose parts are evident only from the observer location. Occasionally it is necessary to draw edges of overlapped objects, most often by dash lines. The visibility algorithms habitually count on the detail of how the objects are represented in an area. The optimal variant is to have objects recounted using their limits, best by surfaces. Most of these algorithms too are prepared for this sort of representation of objects. In this unit, you will learn about implementation of hidden surface in 2D. Visibility algorithm will be categorized. Later, in the unit Hidden surface removal algorithms will be covered.

7.1 Visibility Algorithm Categorization

Visibility computation is crucial for computer graphics from its very beginning. The first visibility algorithms in computer graphics aimed to determine visible surfaces in a synthesized image of a 3D scene. Nowadays there are many different visibility algorithms for various visibility problems.

Sutherland, Sproull and Schumacker termed the visibility algorithms according to their approaches and principles used. They are either image area or object area types. The visibility algorithms can be split depending on the form of output data:

1. ***Line algorithm (Hidden line eliminator):*** On the output we receive a set of abscissas representing visible edges. The benefit of this solution is the fact that we have a set of vectors, and we are not restricted due to resolution issues. In this case, we can scale the scene without any decrease in quality.

2. ***Raster algorithm (Hidden surface eliminator):*** On the output, we get a set of pixels representing the exact points of the scene in the fixed given resolution, without any choice to scale the scene. Here it is, although, possible to draw surfaces with shadowing and lighting.

   As it is time consuming to assess all algorithms, at the starting it is necessary to analyze the object in the view. This is called data pre-processing. First, all objects outside visible angle, are excluded and all polygons of which the scalar vector product is positive, and the given polygon is then reversed

Did u know? Visibility is a phenomenon that can be defined by means of mutually unoccluded points: two points are mutually visible if the line segment connecting them is unoccluded.

7.1.1 Robertson Algorithm

One of the first algorithms is the Robertson algorithm; originally intended for the solution of visibility of a group of convex polyhedrons. This algorithm subsequently processes individual edges of the body. It tests every edge and checks if it is overlapped by another body or not. If it is hidden only partially, then this edge is split into two parts (one hidden, the other unhidden).

Notes Every unhidden part becomes a new tested edge.

7.1.2 Apple Algorithm

This algorithm works in the area of objects and is based on the fact that every edge is a conjunction of two polygons. The first step is to remove invisible polygons. First, we split polygons into two groups: near (potentially visible) and reverse (invisible).

The visible polygons have an obtuse angle with the projection direction; and the invisible ones have an acute angle with the projection direction.

In the following step, we split the edges into three types, according to which polygons they cross:

1. ***First type:*** A conjunction of two reverse polygons, called the invisible edge

2. ***Second type:*** A conjunction of the near polygon and the reverse polygon, called the visible contour edge.

3. ***Third type:*** A conjunction between two near polygons, called the potential visible edge.

Self Assessment

Fill in the blanks:

1. One of the first algorithms is the .... algorithm; originally intended for the solution of visibility of a group of convex polyhedrons.

2. .... algorithm works in the area of objects and is based on the fact that every edge is a conjunction of two polygons.

7.2 Hidden Surface Removal Algorithms

When you draw a scene composed of 3D objects, some of them might obscure all or parts of others. Changing your point of view you can change the obscuring relationship.

Example: If you view the scene from the opposite direction, any object that was previously in front of another is now behind it.

The elimination of parts of solid objects that are obscured by others is called hidden-surface removal. Methods can be categorized as:

- ***Object Space Methods:*** These methods examine objects, faces, edges, etc., to decide which are visible. The complexity depends upon the number of faces, edges, etc., in all the objects.
- ***Image Space Methods:*** These methods examine each pixel in the image to decide which face of which object should be displayed at that pixel.

Caution The complexity depends upon the number of faces and the number of pixels to be considered.

7.2.1 The z-Buffer Algorithm

The z-Buffer algorithm is one of the most commonly used routines. It is simple, easy to implement, and is often found in hardware implementation. The Z-buffer algorithm is a convenient algorithm for rendering images properly according to depth. To begin with, a buffer containing the closest depth at each pixel location is created parallel to the image buffer. Each location in this depth buffer is initialized to negative infinity. Now, the z Intersect and dz Per Scanline fields are added to each edge record in the polyfill algorithm. For each point that is to be rendered, the depth of the point against the depth of the point at the desired pixel location. If the point depth is greater than the depth at the current pixel, the pixel is colored with the new color and the depth buffer is updated. Otherwise, the point is not rendered because it is behind another object.

The idea behind it is uncomplicated: Assign a z-value to each polygon and then display the one (pixel by pixel) that has the smallest value.

When an object is rendered, the depth of a generated pixel (z coordinate) is stored in a buffer (the z-buffer or depth buffer). A 24-bit or 32-bit z-buffer behaves much better than an 8-bit z-buffer (which is hardly used) as it has more precision. Although the problem cannot be entirely eliminated without additional algorithms.

Algorithm

Given: A list of polygons {P1,P2,.....Pn}

Output: A COLOR array, which displays the intensity of the visible polygon surfaces.

```
Initialize:
        note : z-depth and z-buffer(x,y) is positive........
         z-buffer(x,y)=max depth; and
         COLOR(x,y)=background color.
Begin:
     for(each polygon P in the polygon list)
     do{
        for(each pixel(x,y) that intersects P)
        do{
            Calculate z-depth of P at (x,y)
            If (z-depth < z-buffer[x,y])
            then{
                  z-buffer[x,y]=z-depth;
                  COLOR(x,y)=Intensity of P at(x,y);
               }
          }
       }
  display COLOR array.
```

There are some advantages and disadvantages to Z-buffer algorithm.

Advantages:

- It is simple to use
- It can be implemented easily in object or image space
- It can be executed quickly, even with many polygons

Disadvantages:

- It takes up a lot of memory
- It can't do transparent surfaces without additional code

7.2.2 Painter's Algorithm

The painter's algorithm (also known as a priority fills) is one of the simplest solutions to the visibility problem in 3D computer graphics. When a 3D scene is projected onto a 2D plane, it is necessary at one point to decide which polygons are visible, and which are hidden.

Painter's Algorithm (in plain English)

- Sort polygons by farthest depth.
- Check if polygon is in front of any other.
- If no, render it.
- If yes, has its order already changed backward?
    - If no, render it.
    - If yes, break it apart.

Advantages of Painter's Algorithm

- It is quite simple
- It has an easy transparency

Disadvantages of Painter's Algorithm

- It is required to do sorting first
- Its needed to split polygons to solve cyclic and intersecting objects

7.2.3 Binary Space Partitioning (BSP)

Binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyperplanes. By doing this subdivision we can do a representation of the scene by means of a tree data structure known as a BSP tree.

Did u know? Probably the earliest game to use a BSP data structure was Doom.

Algorithm

The recursive algorithm for construction of a BSP tree from that list of polygons is:

1. Choose a polygon P from the list.
2. Make a node N in the BSP tree, and add P to the list of polygons at that node.
3. For each other polygon in the list:
   (a) If that polygon is wholly in front of the plane containing P, move that polygon to the list of nodes in front of P.
   (b) If that polygon is wholly behind the plane containing P, move that polygon to the list of nodes behind P.
   (c) If that polygon is intersected by the plane containing P, split it into two polygons and move them to the respective lists of polygons behind and in front of P.
   (d) If that polygon lies in the plane containing P, add it to the list of polygons at node N.
4. Apply this algorithm to the list of polygons in front of P.
5. Apply this algorithm to the list of polygons behind P.

7.2.4 Ray Tracing

In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane. The effects of its encounter are then simulated with virtual objects.

The Main Program

The procedure RayTrace is recursive, that is, it calls itself. The following code depicts ray tracing algorithm:

```
Procedure RenderPicture()
 For each pixel on the screen,
  Generate a ray R from the viewing position through the point on the view
```

Notes

```
      plane corresponding to this pixel.
    Call the procedure RayTrace() with the arguments R and 0
    Plot the pixel in the colour value returned by RayTrace()
  Next pixel
End Procedure
Procedure RayTrace(ray R, integer Depth) returns colour
  Set the numerical variable Dis to a maximum value
  Set the object pointer Obj to null
  For each object in the scene
    Calculate the distance (from the starting point of R) of the nearest
      intersection of R with the object in the forward direction
    If this distance is less than Dis
      Update Dis to this distance
      Set Obj to point to this object
    End if
  Next object
  If Obj is not null
    Set the position variable Pt to the nearest intersection point of R and Obj
    Set the total colour C to black
    For each light source in the scene
      For each object in the scene
        If this object blocks the light coming from the light source to Pt
          Attenuate the intensity of the received light by the transmittivity
            of the object
        End if
      Next object
      Calculate the perceived colour of Obj at Pt due to this light source
        using the value of the attenuated light intensity
      Add this colour value to C
    Next light source
    If Depth is less than a maximum value
      Generate two rays Refl and Refr in the reflected and refracted directions,
        starting from Pt
      Call RayTrace with arguments Refl and Depth + 1
      Add (the return value * reflectivity of Obj) to C
      Call RayTrace with arguments Refr and Depth + 1
      Add (the return value * transmittivity of Obj) to C
    End if
  Else
    Set the total colour C to the background colour
  End if
  Return C
End Procedure
```

Advantage

It is relatively simple to implement yet yield impressive visual results.

The computational independence of each ray makes ray tracing follow parallelization.

Disadvantage

A serious disadvantage of ray tracing is performance metrics. Scanline and other algorithms use data coherence to share computations between pixels, while ray tracing normally starts the process fresh, treating each eye ray separately.

7.2.5 The Warnock Algorithm

The Warnock algorithm is a hidden surface algorithm invented by John Warnock that is used in the field of computer graphics. It solves the problem of rendering a complicated image by recursive subdivision of a scene until areas are obtained that is trivial to compute. In simple words, if the scene is simple enough to compute efficiently then it is rendered; otherwise it is divided into smaller parts which are likewise tested for simplicity.

This is a divide and conquer algorithm. Its run-time is O(np), where n is the number of polygons and p is the number of pixels in the viewport.

Algorithm

1. Initialize the area.
2. Create list of polygons by sorting them with their z values of vertices. Eliminate disjoint polygons (polygons which are outside the area).
3. Find relationship of each polygon.
4. Perform visibility decision test:
   (a) If all polygons are disjoint, fill area with background color.
   (b) If there is only one contained polygon, fill entire area with background color and fill part of polygon contained in area with color of polygon.
   (c) If there is a single surrounding polygon but not contained then fill area with color of surrounding polygon.
   (d) If area of the pixel (x,y) and if neither of a to d applies compute z coordinate at pixel (x,y) of polygons. Set pixel to the color of polygon which is closer to view.
5. If none of above is true, then subdivide the area and Go to Step 2.

Notes

Self Assessment

Fill in the blanks:

3. .... methods examine objects, faces, edges etc. to decide which are visible.

4. .... methods examine each pixel in the image to decide which face of which object should be displayed at that pixel.

5. The .... algorithm is a convenient algorithm for rendering images properly according to depth.

6. The .... is one of the simplest solutions to the visibility problem in 3D computer graphics.

7. .... is a method for recursively subdividing a space into convex sets by hyper planes.

8. In computer graphics, ... is a technique for generating an image by tracing the path of light through pixels in an image plane.

9. A serious disadvantage of ray tracing is ....

10. The .... algorithm is a hidden surface algorithm invented by John Warnock that is used in the field of computer graphics.

7.3 Summary

- The first visibility algorithms in computer graphics aimed to determine visible surfaces in a synthesized image of a 3D scene.
- The visibility algorithms can be split depending on the form of output data: Line algorithm and Raster algorithm.
- One of the first algorithms is the Robertson algorithm; originally intended for the solution of visibility of a group of convex polyhedrons.
- Apple algorithm works in the area of objects and is based on the fact that every edge is a conjunction of two polygons.
- When you draw a scene composed of 3D objects, some of them might obscure all or parts of others. Changing your point of view you can change the obscuring relationship.
- The z-Buffer algorithm is one of the most commonly used routines.
- When an object is rendered, the depth of a generated pixel (z coordinate) is stored in a buffer (the z-buffer or depth buffer).
- The Painter’s algorithm (also known as a priority fills) is one of the simplest solutions to the visibility problem in 3D computer graphics.
- Binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyperplanes.
- In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane.
- The Warnock algorithm is a hidden surface algorithm invented by John Warnock that is used in the field of computer graphics.

7.4 Keywords

Apple Algorithm: This algorithm works in the area of objects and is based on the fact that every edge is a conjunction of two polygons.

Binary Space Partitioning (BSP): Binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyperplanes.

Hidden-surface removal: The elimination of parts of solid objects that are obscured by others is called hidden-surface removal.

Image Space Methods: These methods examine each pixel in the image to decide which face of which object should be displayed at that pixel.

Object Space Methods: These methods examine objects, faces, edges etc. to decide which are visible.

Painter's Algorithm: The painter's algorithm (also known as a priority fills) is one of the simplest solutions to the visibility problem in 3D computer graphics.

Ray Tracing: In computer graphics, ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane.

Visibility: Visibility is a measure of the distance at which an object or light can be clearly discerned.

Warnock Algorithm: The Warnock algorithm is a hidden surface algorithm invented by John Warnock that is used in the field of computer graphics.

Z-Buffer Algorithm: The Z-buffer algorithm is a convenient algorithm for rendering images properly according to depth.

7.5 Review Questions

1. What is visibility algorithm?
2. Write a short note on line algorithm and raster algorithm.
3. Define Robertson algorithm.
4. What is apple algorithm?
5. Discuss about the hidden surface removal algorithms.
6. What is z-Buffer Algorithm? Also solve the problem of hidden surface removal using Z-buffering algorithm.
7. What are the advantages and disadvantages to Z-buffer algorithm?
8. What is the Painter's algorithm?
9. Write down the advantages and disadvantages of Painter's algorithm.
10. Explain the Binary Space Partitioning (BSP) with algorithm.
11. Define ray tracing. Discuss the advantages and disadvantages.
12. What is Warnock's Algorithm? Explain with algorithm.

Answers: Self Assessment

1. Robertson
2. Apple
3. Object Space
4. Image Space

Notes

5. Z-buffer
6. Painter’s algorithm
7. Binary space partitioning (BSP)
8. Ray tracing
9. Performance metrics
10. Warnock

7.6 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. “*Computer Graphics*” Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. “*Fundamentals of Computer Graphics*” A K Peters Limited.

Angel, Edward. “*Interactive Computer Graphics*” Pearson Education.

Krishnamurthy, N. “*Introduction to Computer Graphics*” Tata McGraw Hill.

Online links

cs.fit.edu/~wds/classes/graphics/Hidden/hidden/hidden.html

www.slideshare.net/HiteshJain007/hidden-surfaces

www.cs.bath.ac.uk/~pjw/NOTES/75-ACG/ch3-surf-rend.pdf

Unit 8: Implementing of Scaling in 2D Transformation

Notes

CONTENTS

Objectives

After studying this unit, you will be able to:

- Define 2D Transformation
- Discuss the concept of Implementing of Scaling in 2D Transformation

Introduction

Transformations are a fundamental part of computer graphics. Transformations are used to position objects, to shape objects, to change viewing positions, and even to change how something is viewed. They are central in computer graphics. They are used to map from one space to another along the graphics pipeline. In this unit, you will learn about implementation of scaling in 2D transformation. 4 types of transformation will be covered. Later on in the unit, concept of Implementing of Scaling in 2D transformation will be discussed followed by a C program to implement scaling in 2D.

8.1 2D Transformation

There are 4 main types of transformations that one can perform in 2 dimensions:

- translations
- scaling
- rotation
- shearing

These basic transformations can be combined to get more complex transformations. In order to make the representation of these complex transformations simpler to realise and more efficient, we introduce the concept of homogeneous coordinates.

Notes

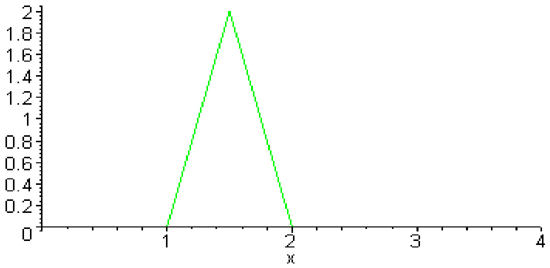
Representation of Points/Objects

A point p in 2D is represented as a pair of numbers: p= (x, y) where x is the x-coordinate of the point p and y is the y-coordinate of p.

Did u know? 2D objects are often represented as a set of points (vertices), {p_1,p_2,...,p_n}, and an associated set of edges {e_1,e_2,...,e_m}. An edge is defined as a pair of points e = {p_i,p_j}.

Points and edges of triangle shown below:

Figure 8.1: Sample Triangle



Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

We can also write points in vector/matrix notation as

$$p = \begin{bmatrix} x \\ y \end{bmatrix}$$

8.1.1 Translations

Assume you are given a point at (x,y)=(2,1).

Where will the point be if you move it 3 units to the right and 1 unit up?

Ans: (x′,y′) = (5,2).

So you must be thinking how was this obtained?

– (x′,y′) = (x+3,y+1).

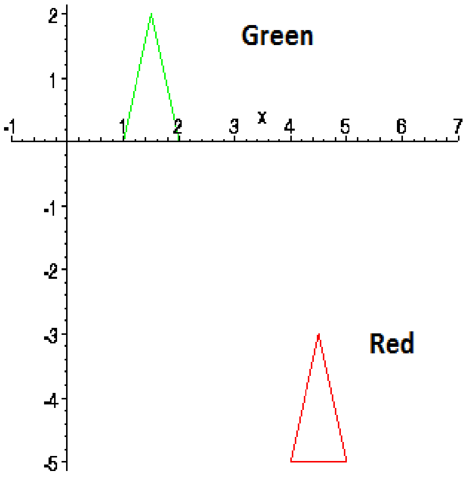
That is, to move a point by some amount dx to the right and dy up, you must add dx to the x-coordinate and add dy to the y-coordinate.

What was the required transformation to move the green triangle to the red triangle?

Here the green triangle is represented by 3 points

triangle = { p1=(1,0), p2=(2,0), p3=(1.5,2) }

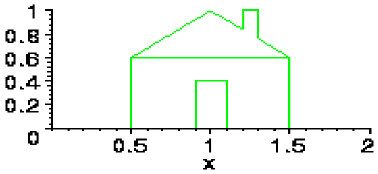
Figure 8.2: Sample Green and Red Triangle



Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

What are the points and edges in this picture of a house? What is the transformation is required to move this house so that the peak of the roof is at the origin? What is required to move the house as shown in animation?

Figure 8.3: Sample House Picture



Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

Matrix/Vector Representation of Translations

A translation can also be represented by a pair of numbers, $t=(t_x,t_y)$ where t_x is the change in the x-coordinate and t_y is the change in y coordinate. To translate the point p by t, we simply add to obtain the new (translated) point q = p + t.

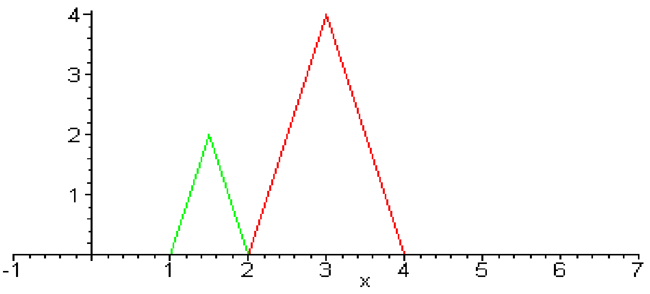
$$q = p + t = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

8.1.2 Scaling

Suppose we want to double the size of a 2-D object. When we talk about scaling we usually mean some amount of scaling along each dimension. That is, we must specify how much to change the size along each dimension. Below we see a triangle and a house that have been doubled in *both* width and height.

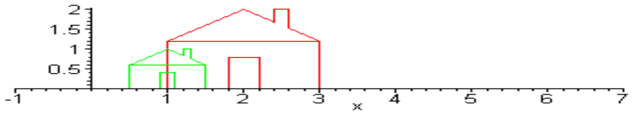
Notes

Figure 8.4: Scaling Triangle Example



Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

Figure 8.5: Scaling House Example

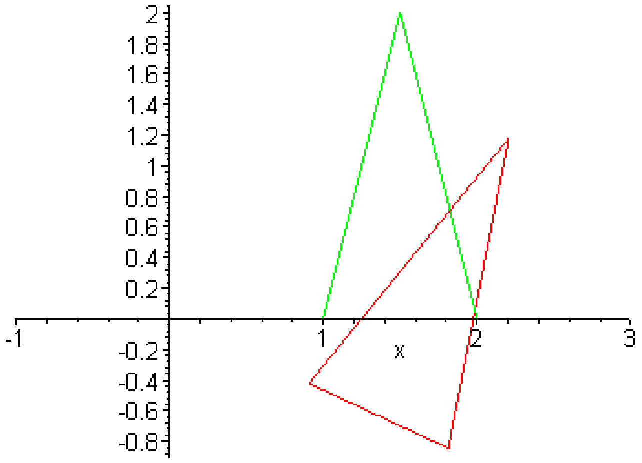


Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

8.1.3 Rotation

Below, we see objects that have been rotate by 25 degrees.

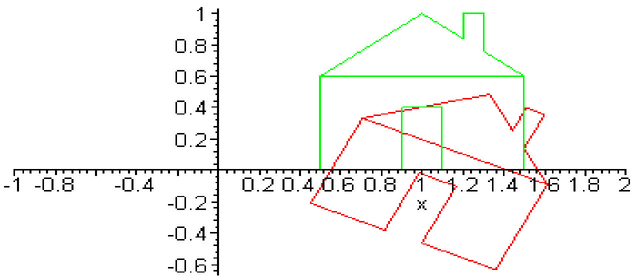
Figure 8.6: Rotation Triangle Example



Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

Notes

Figure 8.7: Rotation House Example



Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

Matrix/Vector Representation of Translations

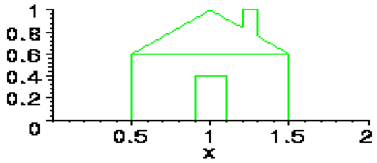
Counterclockwise rotation of **a** degrees = $\begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$

8.1.4 Shear

A shear is a transformation that distorts the shape of an object along either or both of the axis.

Figure 8.8 shows shearing view.

Figure 8.8: Shearing View



Source: http://www.willamette.edu/~gorr/classes/GeneralGraphics/Transforms/transforms2d.htm

Matrix/Vector Representation of Translations

shear along x axis = $\begin{bmatrix} 1 & \text{shear } x \\ 0 & 1 \end{bmatrix}$

shear along y axis = $\begin{bmatrix} 1 & 0 \\ \text{shear } y & 1 \end{bmatrix}$

Self Assessment

Fill in the blanks:

1. There are 4 main types of transformations that one can perform in 2 dimensions: ...., scaling, rotation and ....

2. 2D objects are often represented as a set of points (vertices) and an associated set of ....

3. A translation can also be represented by a ....

4. A .... is a transformation that distorts the shape of an object along either or both of the axis.

8.2 Concept of Implementing of Scaling in 2D Transformation

Let $S_x, S_y = s$ be the scale in the positie **x** and **y** directions respectively. Then the scaled vertex is given by

$$x' = x.S_x \qquad (1)$$

$$y' = y.S_y \qquad (2)$$

If $S_x, S_y = s$, then it is said to be *homogenous* or *uniform* scaling transformation that maintains relative proportions of the scaled objects.

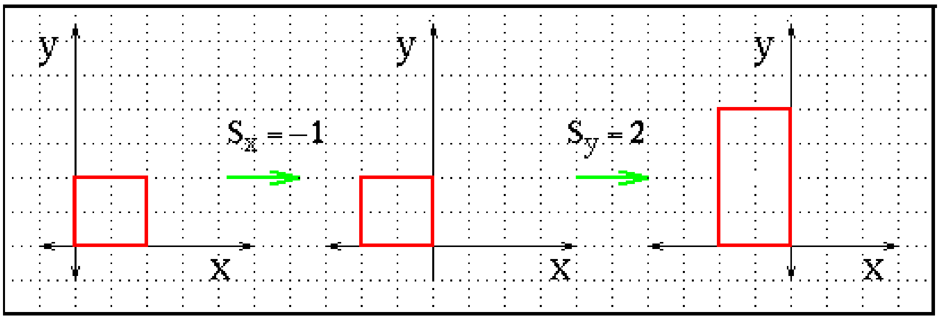
The magnification factor is **|s|**. All points move **s** times away from the origin. If **|s| < 1**, all the points move towards the origin, or demagnified.

Notes If s is negative, reflections occur.

Caution Pure reflections occur only when both the scaling factors have the value +1 or -1.

Example: If $S_x = -1, S_y = 2$, the image is flipped about the y-axis, replacing each point on **x** to **-x** and scaled by **2** in the y direction.

Figure 8.9: Example of Scaling



Source: http://cs.fit.edu/~wds/classes/cse5255/thesis/images/scale/scale.gif

If $S_x \neq S_y$, then the scaling is called *differential scaling*. When either S_x or S_y equals one, simplest differential caling called strain occurs.

2-Dimensional scaling in the matrix form is given by

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3)$$

Lengths and distances from the origin are scaled by multiplying the coordinates of each endpoint by the scaling factors.

Inverse of 2D Scale

$$\begin{bmatrix} x^{-1} & y^{-1} & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{S_x} & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4)$$

The scaling discussed so far has the origin (0,0) as the fixed point and scaling is about the origin. When an object is scaled, it is moved S_x, S_y times away from the origin. It is also possible to have any arbitrary point as a fixed point, and scale about that point.

2D Scaling about an Arbitrary Fixed Point

Usually the origin (0,0) is fixed under scaling. However, any arbitrary fixed point *(xF, yF)* can be selected for scaling. To scale around the arbitrary point:

- Translate *(xF, yF)* to (0,0)
- Scale by (S_x, S_y)
- Translate (0,0) to (*xF*, *yF*)

The result is:

$$x' = x_F + (x - x_F)S_x$$
$$y' = y_F + (y - y_F)S_y$$

C Program to Implement 2D Transformations

```
/*TITLE: WRITE A PROGRAM TO IMPLEMENT 2D TRANSFORMATIONS.
*/

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

void li(float [][3],int x);
void mul(float [][3],float [][3],float [][3],int);
void print(float [][3],float [][3],int);
void main()
{
 float p[10][3],tx,ty;
 int gd=DETECT,gm=DETECT;
 int op,x,i;
```

Notes

```
void scal(float [][3],int);
void trans(float [][3],float,float,int);
void rot(float [][3],int);
void ref(float [][3],int);
void shea(float [][3],int);
clrscr();
initgraph(&gd,&gm,”c:\\tc\\bgi”);
printf(“\nEnter the no of sides of polygon:”);
scanf(“%d”,&x);
printf(“\nEnter the co-ordinates of polygon:”);
 for(i=0;i<x;i++)
 {
  scanf(“%f %f”,&p[i][0],&p[i][1]);
  p[i][2]=1;
 }
do
 {
 printf(“\n*****MENU******”);
 printf(“\n1.Translation\n2. Rotation\n3. Reflection\n4. Shearing\n5. Scaling\n6. Exit\nEnter the choice:”);
 scanf(“%d”,&op);
 initgraph(&gd,&gm,”c:\\tc\\bgi”);
 switch(op)
 {
  case 1: printf(“\nTRANSLATION”);
      printf(“\nEnter the tx and ty: “);
      scanf(“%f %f”,&tx,&ty);
      trans(p,tx,ty,x);
      break;
  case 2: rot(p,x);
      break;
  case 3: ref(p,x);
      break;
  case 4: shea(p,x);
      break;
  case 5: scal(p,x);
```

```
      break;
   case 6: printf(“\nCOMING OUT!!!”);
      break;
   default: printf(“\nWrong choice”);
       break;
  }
 getch();
 capture(“c:\\tc\\2d.bmp”);
 closegraph();
 }while(op!=6);
}

void rot(float p[][3],int x)
{
 float  c[3][3]={0,0,0,0,0,0,0,0,1},d[10][3],ange;
 int i;
 printf(“\nROTATION”);
 printf(“\nEnter the angle for rotation:”);
 scanf(“%d”,&i);
 ange=3.14/180*i;
 c[0][0]=c[1][1]=cos(ange);
 c[0][1]=sin(ange);
 c[1][0]=-sin(ange);
 mul(p,c,d,x);
 print(p,d,x);
capture(“gh.jpg”);
}
 void print(float p[][3],float d[][3],int x)
 {
  line(300,0,300,1000);
  line(0,250,1000,250);
  li(p,x);
  li(d,x);
 }
 void li(float p[][3],int x)
 {
```

Notes

```
 int i;
 for(i=0;i<x;i++)
  {
   if(i==x-1)
    line(p[i][0]+300,p[i][1]+250,p[0][0]+300,p[0][1]+250);
   else
    line(p[i][0]+300,p[i][1]+250,p[i+1][0]+300,p[i+1][1]+250);
  }
}
void trans(float p[][3],float tx,float ty,int x)
{
 float c[3][3]={1,0,0,0,1,0,0,0,1},d[10][3];
 c[2][0]=tx;
 c[2][1]=ty;
 mul(p,c,d,x);
 if(tx!=-300&&tx!=300)
 print(p,d,x);
capture("fg.jpg");
}
void scal(float p[][3],int x)
{
 float c[3][3]={1,0,0,0,1,0,0,0,1},d[10][3];
 printf("\nSCALING");
 printf("\nEnter the sx and sy: ");
 scanf("%f %f",&c[0][0],&c[1][1]);
 mul(p,c,d,x);
 print(p,d,x) ;
capture("cd.jpg");
}
void mul(float p[][3],float c[][3],float d[][3],int x)
{
 int i,j,k;
 for(i=0;i<x;i++)
  {
   for(j=0;j<3;j++)
```

```
    {
     d[i][j]=0;
     for(k=0;k<3;k++)
      {
   d[i][j]=d[i][j]+(p[i][k]*c[k][j]);
      }
    }
  }
}
void shea(float p[][3],int x)
{
 int op,i,j,k;
 float c[3][3]={1,0,0,0,1,0,0,0,1},d[10][3];
 printf(“\n1. X-Shear\n2. Y-Shear\nEnter the choice:”);
 scanf(“%d”,&op);
 switch(op)
  {
   case 1: printf(“\nEnter the X-Shear:”);
       scanf(“%f”,&c[1][0]);
       c[0][1]=0;
       break;
   case 2: printf(“\nEnter the Y-Shear:”);
       scanf(“%f”,&c[0][1]);
       c[1][0]=0;
       break;
  }
 trans(p,-300,-250,x);
 mul(p,c,d,x);
 trans(p,300,250,x);
 print(p,d,x);
capture(“pl.jpg”);
}

void ref(float p[][3],int x)
{
 int ch;
 float c[3][3]={1,0,0,0,1,0,0,0,1},d[10][3];
 printf("\n\t\tREFLECTION");
  printf("\n1. Reflection about Y-axis\n2. Reflection about X-axis\n3. Reflection about Origin\
n4. Reflection about line Y=X\n5. Reflection about line Y=-X\nEnter the choice:");
 scanf("%d",&ch);
 switch(ch)
  {
   case 1: c[0][0]=-1;
       break;
   case 2: c[1][1]=-1;
       break;
   case 3: c[0][0]=-1;
       c[1][1]=-1;
       break;
   case 4: c[0][0]=c[1][1]=0;
       c[0][1]=c[1][0]=1;
       break;
   case 5: c[0][0]=c[1][1]=0;
       c[0][1]=c[1][0]=-1;
       break;
  }
 mul(p,c,d,x);
 print(p,d,x);
 capture("po.jpg");
}
```

Self Assessment

State whether the following statements are true or false:

5. If $S_x \neq S_y$, then it is said to be homogenous or uniform scaling transformation that maintains relative proportions of the scaled objects.

6. If $S_x , S_y = s$, then the scaling is called differential scaling.

8.3 Summary

- There are 4 main types of transformations that one can perform in 2 dimensions: translations, scaling, rotation and shearing.
- When we talk about scaling we usually mean some amount of scaling along each dimension. That is, we must specify how much to change the size along each dimension.

Notes

- A shear is a transformation that distorts the shape of an object along either or both of the axis.
- The magnification factor is $|s|$. All points move **s** times away from the origin. If $|s| < 1$, all the points move towards the origin, or demagnified.
- 2D transformations are used by the application for modelling transformations.

8.4 Keywords

Rotation: Rotation means turning around a center. The distance from the center to any point on the shape stays the same.

Scaling: It does not preserve angles between parts of objects (except when scaling is uniform x=sy).

Shear: A shear is a transformation that distorts the shape of an object along either or both of the axis.

Transformations: Transformations are used to position objects, to shape objects, to change viewing positions, and even to change how something is viewed (e.g. the type of perspective that is used).

8.5 Review Questions

1. What is 2D transformation?
2. What are the different functions of transformation?
3. What are the 4 main types of transformations?
4. Discuss about the matrix/vector representation of translations.
5. Write a C program to implement 2D transformations.
6. Describe the transformation between translation and scale.
7. Discuss the 2D scaling from the origin.
8. Write down the properties of scaling.
9. Elucidate the concept of implementing of scaling in 2d transformation.
10. What is 2D scaling about an arbitrary fixed point? Discuss the C program to implement 2D transformations.

Answers: Self Assessment

1. Translations, shearing
2. Edges
3. Pair of numbers
4. Shear
5. False
6. False

8.6 Further Readings

Books

Angel, Edward. “*Interactive Computer Graphics*” Pearson Education.

Hearn, Donald & Baker, M. Pauline. “*Computer Graphics*” Pearson Education.

Notes

Krishnamurthy, N. "*Introduction to Computer Graphics*" Tata McGraw Hill.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Online links

web.iitd.ac.in/~hegde/cad/lecture/L5_2dtrans.pdf

gmm.fsksm.utm.my/~suriati/graphics/2D_TRANSFORMATIONS.ppt

www.cs.uic.edu/~jbell/CourseNotes/.../2DTransforms.html

Unit 9: Translation

Notes

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain visibility algorithm categorization
- Discuss the hidden surface removal algorithms

Introduction

In geometry, an affine transformation or affine map or an affinity is a transformation of an affine space which preserves straight lines and ratios of distances between points lying on a straight line. Examples of affine transformations include translation, reflection, rotation, shear mapping etc. Since a translation is an affine transformation but not a linear transformation, homogeneous coordinates are normally used to represent the translation operator by a matrix and thus to make it linear. Thus we write the 3-dimensional vector w = (wx, wy, wz) using 4 homogeneous coordinates as w = (wx, wy, wz, 1). In this unit, you will learn about translation. Translation of 2-D and 3-D will be covered in this unit. Lastly, inverse of translation is explained with an example.

9.1 2–Dimensional Translation in C program

To translate an object by a vector v, each homogeneous vector p (written in homogeneous coordinates) would need to be multiplied by this translation matrix:

$$T_V = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As shown below, the multiplication will give the expected result:

Notes

$$T_V p = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix} = p + v$$

Notes The inverse of a translation matrix can be obtained by reversing the direction of the vector:

$$T_v^{-1} = T_{-v}$$

Similarly, the product of translation matrices is given by adding the vectors:

$$T_u T_v = T_{u+v}.$$

C Program

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>
#include<math.h>
int x1,y1,x2,y2,x3,y3,mx,my;
void draw();
void tri();
void main()
{
  int gd=DETECT,gm;
  int c;
  initgraph(&gd,&gm,”d:\\tc\\bgi “);
  printf(“Enter the 1st point for the triangle:”);
  scanf(“%d%d”,&x1,&y1);
  printf(“Enter the 2nd point for the triangle:”);
  scanf(“%d%d”,&x2,&y2);
  printf(“Enter the 3rd point for the triangle:”);
  scanf(“%d%d”,&x3,&y3);
   cleardevice();
  draw();
  getch();
  tri();
```

Notes

```
   getch();
}
 void draw()
{
   line(x1,y1,x2,y2);
   line(x2,y2,x3,y3);
   line(x3,y3,x1,y1);
}
void tri()
{
   int x,y,a1,a2,a3,b1,b2,b3;
   printf("Enter the Transaction coordinates");
   scanf("%d%d",&x,&y);
   cleardevice();
   a1=x1+x;
   b1=y1+y;
   a2=x2+x;
   b2=y2+y;
   a3=x3+x;
   b3=y3+y;
   line(a1,b1,a2,b2);
   line(a2,b2,a3,b3);
   line(a3,b3,a1,b1);
}
```

Self Assessment

Fill in the blanks:

1. The .... of a translation matrix can be obtained by reversing the direction of the vector.

2. In geometry, an affine transformation is a transformation of an affine space which preserves .... of distances between points lying on a straight line.

9.2 3–Dimensional Translation in C Program

3-dimensional transformation has three axis x,y,z. Depending upon there coordinate it will perform there translation, rotation and scaling.

Did u know? The translation can be performed by changing the sign of the translation components T_x, T_y, and T_z.

Notes

Source code

```
#include <stdio.h>
#include <stdlib.h>
#include<graphics.h>
#include<conio.h>
void draw3d(int fs,int x[20],int y[20],int tx,int ty,int d);
void draw3d(int fs,int x[20],int y[20],int tx,int ty,int d)
{
int i,j,k=0;
for(j=0;j<2;j++)
{
for(i=0;i<fs;i++)
{
if(i!=fs-1)
line(x[i]+tx+k,y[i]+ty-k,x[i+1]+tx+k,y[i+1]+ty-k);
else
line(x[i]+tx+k,y[i]+ty-k,x[0]+tx+k,y[0]+ty-k);
}
k=d;
}
for(i=0;i<fs;i++)
{
line(x[i]+tx,y[i]+ty,x[i]+tx+d,y[i]+ty-d);
}
}
void main()
{
int gd=DETECT,gm;
int x[20],y[20],tx=0,ty=0,i,fs,d;
initgraph(&gd,&gm,"");
printf("no of sides (front view only) : ");
scanf("%d",&fs);
printf("co-ordinates : ");
for(i=0;i<fs;i++)
{
```

```
printf("(x%d,y%d)",i,i);
scanf("%d%d",&x[i],&y[i]);
}
printf("Depth :");
scanf("%d",&d);
draw3d(fs,x,y,tx,ty,d);
printf("translation (x,y)");
scanf("%d%d",&tx,&ty);
draw3d(fs,x,y,tx,ty,d);
getch();
}
```

Self Assessment

Fill in the blanks:

3. .... dimensional transformation has three axis x,y,z.

4. The translation can be performed by changing the .... of the translation components T_x, T_y, and T_z.

9.3 Translating with Matrices

A translation is an affine transformation with no fixed points.

Caution Matrix multiplications always have the origin as a fixed point.

We can write the 3-dimensional vector w = (w_x, w_y, w_z) using 4 homogeneous coordinates as w = (w_x, w_y, w_z, 1).

We can express the translation as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \qquad T\begin{bmatrix} t_x \\ t_y \end{bmatrix}.$$

This allows us to write the two-dimensional translation equations in the matrix form:

$$P' = P + T$$

9.4 The Inverse of a Translation

The inverse of a translation matrix can be obtained by reversing the direction of the vector:

$$T_v^{-1} = T_{-v}$$

As long as we do not scale by zero, a scale can always be inverted (undone) by the matrix

Notes

$$s^{-1} = \begin{bmatrix} \frac{1}{S_x} & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 \\ 0 & 0 & \frac{1}{S_z} \end{bmatrix}.$$

The product SS^{-1} = S^{-1}S = I, the 3×3 identity matrix.

Now, let us take an example, to undo a translation by t_x, t_y, t_z use the matrix

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -t_x & -t_y & -t_z & 1 \end{bmatrix}$$

We can now complete scaling about an arbitrary fixed point and rotation about an arbitrary pivot. To scale about $F = [x_f\, y_f\, z_f]$ use the composition of matrices

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_f & -y_f & -z_f & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_f & y_f & z_f & 1 \end{bmatrix}$$

which when multiplied out yields

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ x_f(1-S_x) & y_f(1-S_y) & z_f(1-S_z) & 1 \end{bmatrix}$$

So a scaled point [x y z 1] becomes

$$[x'\; y'\; z'\; 1] = [x\; y\; z\; 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ x_f(1-S_x) & y_f(1-S_y) & z_f(1-S_z) & 1 \end{bmatrix}$$

$$= \left[x8_x + x_f\,(1-8_x)y8_y + y_f\left(1-s_y\right)z8_z + z_f\,(1-8_z)1\right]$$

In a similar manner you can determine that rotation about a pivot R = $[x_r\, y_r]$ results in

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

Notes As addition of vectors is commutative, multiplication of translation matrices is therefore also commutative.

Self Assessment

Fill in the blanks:

5. A translation is an affine transformation with no .... points.
6. As addition of vectors is commutative, multiplication of translation matrices is therefore also ....

9.5 Summary

- A translation is applied to an object to reposition it along a straight-line path from one coordinate location to another.
- A translation is an affine transformation with no fixed points. Matrix multiplications always have the origin as a fixed point.
- Translation transformation is used to position the origin point of the initial space relative to the destination space.
- Since all of the coordinates in a space are relative to the origin point of that space, all a translation needs to do is add a vector to all of the coordinates in that space.
- The vector added to these values is the location of where the user wants the origin point relative to the destination coordinate system.
- 3 dimensional transformation has three axis x,y,z. Depending upon there coordinate it will perform there translation, rotation and scaling.
- A translation is an affine transformation with no fixed points.

9.6 Keywords

Translation: A translation is applied to an object to reposition it along a straight-line path from one coordinate location to another.

Translation Matrix: This function takes as parameters a reference to the matrix that holds the current state of the transformation and the X and Y translation values.

Translation Vector: The translation distance pair (tx; ty) is called a translation vector.

9.7 Review Questions

1. Discuss the C program in 2-Dimensional of the translation.
2. Explain the C program in 3-Dimensional of the translation with the help of source code.
3. "A translation is an affine transformation with no fixed points". Elucidate.
4. What is the process of inversing the translation?

Answers: Self Assessment

1.	Inverse	2.	Straight lines and ratios
3.	3	4.	Sign
5.	Fixed	6.	Commutative

9.8 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. “*Computer Graphics*” Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. “*Fundamentals of Computer Graphics*” A K Peters Limited.

Angel, Edward. “*Interactive Computer Graphics*” Pearson Education.

Krishnamurthy, N. “*Introduction to Computer Graphics*” Tata McGraw Hill.

Online links

http://www.studentcpu.com/2009/03/3d-transformation-in-c-program-computer.html

www.c.happycodings.com/Games_and_Graphics/code36.html

cprogramtutorials.blogspot.com/.../3d-transformation-program-in-c.html

Unit 10: Shearing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain shearing with C programming
- Discuss rotation with C programming
- Identify reflection with C programming

Introduction

In plane geometry, a shear mapping is a linear map that displaces each point in a fixed direction, by an amount proportional to its signed distance from a line that is parallel to that direction. This type of mapping is also known as shear transformation or just shearing. In this unit, we will discuss different types of shearing. Later on in this unit, rotation and reflection will be covered.

10.1 Shearing

Shear is the translation along an axis (say, X axis) by an amount that increases linearly with another axis (Y). It leads to the production of shape distortions as if objects were composed of layers that are caused to slide over each other.

Did u know? Shear transformations are very useful in creating italic letters and slanted letters from regular letters.

10.1.1 2D Shear along X-direction

Shear in X direction is represented by the following set of equations.

$$Sh_x = P_x + hP_y$$
$$Sh_y = P_y$$

where h is the negative or positive fraction of Y coordinate of P to be added to the X coordinate. Sh_x can be any real number.

The matrix of form of shear in x-direction is given by:

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

10.1.2 2D Shear along Y Direction

Similarly, shear along y-direction is given by

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

Combining the shear in X and Y directions,

$$Sh_x = P_x + hP_y$$
$$Sh_y = gP_y + P_y$$

where g is a non-zero fraction of P_x to be added to the Y coordinate.

General Matrix Form of Shear

The general matrix form of shear is

$$\begin{bmatrix} 1 & g \\ h & 1 \end{bmatrix}$$

Notes Shear will reduce to a pure shear in the y-direction, when h = 0.

The inverse of Shear is given by

$$\left[Sh^{-1} = frac11 - gh \begin{bmatrix} 1 & -g \\ -h & 1 \end{bmatrix} \right.$$

Horizontal and vertical shear of the plane.

In the plane $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$, a horizontal shear (or shear parallel to the x axis) is a function that takes a generic point with coordinates (x, y) to the point (x+ my, y); where m is a fixed parameter, called the shear factor.

If the coordinates of a point are written as a column vector (a 2 × 1 matrix), the shear mapping can be written as multiplication by a 2 × 2 matrix:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + my \\ y \end{pmatrix} = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

A vertical shear (or shear parallel to the *y*-axis) of lines is similar, except that the roles of *x* and *y* are swapped. It corresponds to multiplying the coordinate vector by the transposed matrix:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} y \\ mx+y \end{pmatrix} = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

Self Assessment

Fill in the blanks:

1. .... leads to the production of shape distortions as if objects were composed of layers that are caused to slide over each other.
2. Shear transformations are very useful in creating italic letters and slanted letters from .... letters.
3. Shear will reduce to a pure shear in the y-direction, when h = ...

10.2 Rotation

In linear algebra, a rotation matrix is a matrix that is used to perform a rotation in Euclidean space.

Example: For example the matrix

$$R = \begin{bmatrix} \cos\theta - \sin\theta \\ \sin\theta \quad \cos\theta \end{bmatrix}$$

rotates points in the *xy*-Cartesian plane counterclockwise through an angle θ about the origin of the Cartesian coordinate system.

Caution To perform the rotation using a rotation matrix R, the position of each point must be represented by a column vector v, containing the coordinates of the point.

A rotated vector is obtained by using the matrix multiplication Rv.

10.2.1 2D Rotation Program Using C Programming

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>
#include<math.h>
void TriAngle(int x1,int y1,int x2,int y2,int x3,int y3);
void Rotate(int x1,int y1,int x2,int y2,int x3,int y3);
void main()
{
  int gd=DETECT,gm;
```

Notes

```
    int x1,y1,x2,y2,x3,y3;
    initgraph(&gd,&gm," ");
    printf("Enter the 1st point for the triangle:");
    scanf("%d%d",&x1,&y1);
    printf("Enter the 2nd point for the triangle:");
    scanf("%d%d",&x2,&y2);
    printf("Enter the 3rd point for the triangle:");
    scanf("%d%d",&x3,&y3);
    TriAngle(x1,y1,x2,y2,x3,y3);
    getch();
    cleardevice();
    Rotate(x1,y1,x2,y2,x3,y3);
     setcolor(1);
    TriAngle(x1,y1,x2,y2,x3,y3);
    getch();
}
void TriAngle(int x1,int y1,int x2,int y2,int x3,int y3)
{
   line(x1,y1,x2,y2);
   line(x2,y2,x3,y3);
   line(x3,y3,x1,y1);
}
void Rotate(int x1,int y1,int x2,int y2,int x3,int y3)
{
    int x,y,a1,b1,a2,b2,a3,b3,p=x2,q=y2;
    float Angle;
    printf("Enter the angle for rotation:");
    scanf("%f",&Angle);
    cleardevice();
    Angle=(Angle*3.14)/180;
    a1=p+(x1-p)*cos(Angle)-(y1-q)*sin(Angle);
    b1=q+(x1-p)*sin(Angle)+(y1-q)*cos(Angle);
    a2=p+(x2-p)*cos(Angle)-(y2-q)*sin(Angle);
    b2=q+(x2-p)*sin(Angle)+(y2-q)*cos(Angle);
    a3=p+(x3-p)*cos(Angle)-(y3-q)*sin(Angle);
    b3=q+(x3-p)*sin(Angle)+(y3-q)*cos(Angle);
    printf(“Rotate”);
   TriAngle(a1,b1,a2,b2,a3,b3);
}
```

10.2.2 3D Rotation Program Using C Programming

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<process.h>
#include<graphics.h>
int x1,x2,y1,y2,mx,my,depth;
void draw();
void rotate();
void main()
{
   int gd=DETECT,gm,c;
   initgraph(&gd,&gm,”d:\\tc\\bgi”);
   printf(“\n3D Transformation Rotating\n\n”);
   printf(“\nEnter 1st top value(x1,y1):”);
   scanf(“%d%d”,&x1,&y1);
   printf(“Enter right bottom value(x2,y2):”);
   scanf(“%d%d”,&x2,&y2);
   depth=(x2-x1)/4;
   mx=(x1+x2)/2;
   my=(y1+y2)/2;
   draw();
   getch();
   cleardevice();
   rotate();
   getch();
}
void draw()
{
   bar3d(x1,y1,x2,y2,depth,1);
}
 void rotate()
```

Notes

```
{
  float t;
  int a1,b1,a2,b2,dep;
  printf("Enter the angle to rotate=");
  scanf("%f",&t);
  t=t*(3.14/180);
  a1=mx+(x1-mx)*cos(t)-(y1-my)*sin(t);
  a2=mx+(x2-mx)*cos(t)-(y2-my)*sin(t);
  b1=my+(x1-mx)*sin(t)-(y1-my)*cos(t);
  b2=my+(x2-mx)*sin(t)-(y2-my)*cos(t);
  if(a2>a1)
    dep=(a2-a1)/4;
  else
   dep=(a1-a2)/4;
  bar3d(a1,b1,a2,b2,dep,1);
  setcolor(5);
  //draw();
}
```

Self Assessment

State whether the following statements are true or false:

4. In linear algebra, a rotation matrix is a matrix that is used to perform a rotation in Euclidean space.
5. To perform the rotation using a rotation matrix R, the position of each point must be represented by a column vector v.

10.3 Reflection

Reflection in computer graphics is used to emulate reflective objects like mirrors and shiny surfaces. There are four types of reflection:

Polished or Mirror reflection: Mirrors are usually almost 100% reflective.

Metallic Reflection: Metallic objects reflect lights and colors altered by the color of the metallic object itself.

Blurry: A Blurry Reflection means that tiny random bumps on the surface of the material cause the reflection to be blurry.

Glossy Reflection: Fully glossy reflection, shows highlights from light sources, but does not show a clear reflection from objects.

Self Assessment

Fill in the blanks:

6. .... in computer graphics is used to emulate reflective objects like mirrors and shiny surfaces.
7. Mirrors are usually almost .... reflective.
8. A .... Reflection means that tiny random bumps on the surface of the material cause the reflection to be blurry.

10.4 Summary

- Shear is the translation along an axis (say, X axis) by an amount that increases linearly with another axis (Y).
- In linear algebra, a rotation matrix is a matrix that is used to perform a rotation in Euclidean space.
- Reflection in computer graphics is used to emulate reflective objects like mirrors and shiny surfaces. There are four types of reflection:
    - Mirrors are usually almost 100% reflective.
    - Metallic objects reflect lights and colors altered by the color of the metallic object itself.
    - A Blurry Reflection means that tiny random bumps on the surface of the material cause the reflection to be blurry.
    - Fully glossy reflection, shows highlights from light sources, but does not show a clear reflection from objects.

10.5 Keywords

Blurry: A Blurry Reflection refers to the small random bumps on the surface of the material which causes the blurred reflection.

Metallic Reflection: It is a kind of reflection in which metallic objects reflect lights and colors which are changed by the color of the metallic object itself.

10.6 Review Questions

1. What do you mean by shearing?
2. Discuss different types of reflection.
3. Define Y-shear shearing.
4. Write the C code for 2D Rotation Program.
5. Explain shearing along x-axis.

Answers: Self Assessment

1.	Shear	2.	Regular
3.	0	4.	True

Notes

5. True
6. Reflection
7. 100%
8. Blurry

10.7 Further Readings

Books

Hearn, Donald & Baker, M. Pauline. "*Computer Graphics*" Pearson Education.

Shirley, Peter & Marschner, Stephen Robert. "*Fundamentals of Computer Graphics*" A K Peters Limited.

Angel, Edward. "*Interactive Computer Graphics*" Pearson Education.

Krishnamurthy, N. "*Introduction to Computer Graphics*" Tata McGraw Hill.

Online links

www.gomezconsultants.com/CSE5280/Transformations/2DTrans.html

ltcconline.net/greenl/courses/203/Vectors/computerGraphics.htm

dpancaroglu.etu.edu.tr/bil421/Lecture%2006.ppt

LOVELY PROFESSIONAL UNIVERSITY
Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-300360
Fax.: +91-1824-506111
Email: odl@lpu.co.in