Lecture # 1


2) Target: At the completion of this lecture you should be able to answer questions like
   (a) What is Computer Graphics?
   (b) Which are the different types of Computer Graphics?
   (c) How did Computer Graphics originate?
   (d) What is the importance of speed in displaying pictures?
   (e) How does a picture grow, shrink and rotate?
   (f) What are the main applications of Computer Graphics?

You should also be able to explain the working of an interactive graphics display.

3) Introduction

The main objective of this course is to provide an overview of computer graphics. This course also provides you a thorough understanding of the basic concepts, mathematical models, techniques, and algorithms used in computer graphics in two and three dimensions.

Today there are very few aspects of our lives not affected by computers. Practically every cash or monetary transaction that takes place daily involves a computer. In many cases, the same is true of computer graphics. Whether you see them on television, in newspapers, in weather reports or while at the doctor’s surgery, computer images are all around you. “A picture is worth a thousand words” is a well-known saying and highlights the advantages and benefits of the visual presentation of our data. We are able to obtain a comprehensive overall view of our data and also study features and areas of particular interest.

A well-chosen graph is able to transform a complex table of numbers into meaningful results. You know that such graphs are used to illustrate papers, reports and thesis, as well as providing the basis for presentation material in the form of slides and overhead transparencies. A range of tools and facilities are available to enable users to visualize their data, and this document provides a brief summary and overview.
Computer graphics can be used in many disciplines. Charting, Presentations, Drawing, Painting and Design, Image Processing and Scientific Visualization are some among them.

Before we begin on this course, a revision of the concepts developed earlier like

a) Data structures and algorithms (R 405)
b) A good working knowledge of C/ C++ programming (RT 304/R 403)
c) Fundamental Mathematics like Linear algebra, Vector Calculus and Trigonometry

may be helpful. Once we finish this aspect, we will proceed towards exposition of items listed in the synopsis.

In particular, we will emphasize the following

a) Basic concepts of Computer Graphics
b) Different types of Computer Graphics
c) Origin of Computer Graphics
d) Working of an interactive graphics display
e) Importance of speed in displaying pictures
f) The change in size an orientation of pictures
g) Applications of Computer Graphics

4) Revision / Prerequisites

You will get the most out of this course if you:

- have C or C++ programming experience (essential for completing assignments!)
- have taken an introductory data structure course (e.g. R 405)
- have some linear algebra, vector calculus and trigonometry background

We will review the basic concepts in mathematics as needed throughout the course, so don't worry if you're rusty in these areas. If you haven't learnt C/C++ at all, you may have trouble keeping up in the course unless you pick it up quickly in the first few weeks.

The required textbooks for refreshing the concepts developed earlier are:

c) Programming with C - Byron S. Gottfried, Tata McGraw Hill
d) Higher Engg. Mathematics- Grewal B.S.

The required textbooks for the course are:

a) Computer Graphics (C version) - Donald Hearn & Pauline Baker (Pearson Education Asia)

b) Computer Graphics - Donald Hearn & Pauline Baker (Prentice Hall of India)


e) Fundamentals of Computer graphics & multimedia- D. P. Mukherjee, Prentice Hall


5.1) Concept #1: Basic concepts of Computer Graphics

I hope all of you are fond of video games and you may be good at playing them. Have you seen the game of ping-pong? It’s a game played by two people with a pair video game controller and a home television set. You can see that when a game is switched on, a small bright spot, representing a ball, is seen bouncing to and fro across the screen. Now each player uses his video game controller to position a ‘paddle’ to bounce the ball back to his opponent. The player who hits the ball past his opponent wins a point and the one who gains 15 points wins the game. Now how did you invent this video game? This has been done with the aid of Computer Graphics. Video games represent a major use in the home of computer graphics. Computer graphics helps to create and manipulate pictures with the aid of computers.

Computer graphics is concerned with all aspects of producing images using a computer. It concerns with the pictorial synthesis of real or imaginary objects from their computer-based models.

5.2) Concept #2: Different types of Computer Graphics

Computer Graphics can be broadly divided into two

a) Non Interactive Computer Graphics

b) Interactive Computer Graphics
Non Interactive Computer Graphics: In non interactive computer graphics otherwise known as passive computer graphics, the observer has no control over the image. Familiar examples of this type of computer graphics include the titles shown on TV and other forms of computer art.

Interactive Computer Graphics: Interactive Computer Graphics involves a two way communication between computer and user. Here the observer is given some control over the image by providing him with an input device for example the video game controller of the ping pong game. This helps him to signal his request to the computer. The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

Interactive computer graphics affects our lives in a number of indirect ways. For example, it helps to train the pilots of our airplanes. We can create a flight simulator which may help the pilots to get trained not in a real aircraft but on the grounds at the control of the flight simulator. The flight simulator is a mock up of an aircraft flight deck, containing all the usual controls and surrounded by screens on which we have the projected computer generated views of the terrain visible on take off and landing. Flight simulators have many advantages over the real aircrafts for training purposes, including fuel savings, safety, and the ability to familiarize the trainee with a large number of the world’s airports.

5.3) Concept #3: Origin of Computer Graphics

Years of research and development were made to achieve the goals in the field of computer graphics. In 1950 the first computer driven display was used to generate only simple pictures. This display made use of a cathode ray tube similar to the one used in television sets. During 1950’s interactive computer graphics made little progress because the computers of that period were so unsuited to interactive use. These computers were used to perform only lengthy calculations.

The single vent that did the most to promote interactive computer graphics as an important new field was the publication in 1962 of a brilliant thesis by Ivan E. Sutherland. His thesis, entitled ‘Sketchpad: A Man-Machine Graphical Communication System’, proved to many readers that interactive computer graphics was a viable, useful, and exciting field of research. By the mid -1960’s large computer graphics research projects were under taken at MIT, Bell Telephone Labs and General Motors. Thus the golden age of computer graphics
began. In 1970’s the researches began to bear fruit. The instant appeal of computer graphics to users of all ages has helped it to spread into many applications throughout the world.

5.4) Concept #4: Working of an interactive graphics display

Interactive graphics display consists of three components

a) A display controller
b) A digital memory or frame buffer
c) A television monitor
d) A video controller

The display controller gets the inputs and commands from the user and determines the image to be displayed on the monitor. The display controller will divide the image into a number of pixels. This image which is to be displayed is stored in the frame buffer. The image will be stored as a matrix of intensity values. The image will be displayed onto the television monitor and the video controller will act as a simple interface that passes the contents of the frame buffer to the monitor. The image must be repeatedly passed to the monitor, 30 or more times a second. This helps you to maintain a steady picture on the screen.

In the frame buffer the image is stored as a pattern of binary digital numbers. These binary digital numbers represents a rectangular array of picture elements or pixels (a picture can be divided into a number of picture elements or pixels. You will learn more about pixels in the coming lectures.). So corresponding to each pixel you have a binary digital number in the frame buffer. If your image is a black and white image you can represent the black pixels by 0 s and white pixels by 1s. Therefore a 16 X 16 array of black and white pixels could be represented by the binary values stored in the 32 8-bit bytes. Now what happens to this data?

The video controller simply reads each successive byte of data from the frame buffer and converts its 0s and 1s into the corresponding video signal. This signal is then fed into the TV monitor, producing a black and white pattern on the screen. The video controller repeats this operation 30 times a second in order to maintain a steady picture on the TV screen.

Now what should be done to change or modify this image?
All we need is to modify the frame buffer’s contents. Set the frame buffer with a new set of values so that it represents the new image. In this way we can achieve effects like a rotating wheel and a wheel that grows and shrinks.

The figure given below gives an idea about the graphics display system

![Graphics Display System Diagram](image)

5.5) Concept #5: Importance of speed in displaying pictures

Why is speed so important in displaying pictures?

One reason behind this is that any display based on the CRT must be refreshed by repeatedly passing it to the monitor. The image must be transmitted to the display monitor point by point. If the image is not transmitted at least 25 times per second, the image will start to flicker in an unpleasant manner. If the speed of transmitting each picture element is less, then only fewer elements will be transmitted and only less information will be displayed.

The second reason is that the response of a computer program to the actions by its user should be very fast i.e. the response time should be very small. The speed of response depends on two factors.
1. The rate at which the computer can generate a fresh image in response to each action by its users.
2. The rate at which the image is transmitted to the display monitor.

Generally speaking, slow response always makes interactive graphics program more difficult to operate. Perhaps that’s the reason why research efforts are made to improve the speed of interactive response.

5.6) Concept #6: The change in size an orientation of pictures

How are pictures made to rotate? How are pictures made to shrink? How are pictures made to grow? Pictures can be made to change the orientation and size. How are these changes possible?

These transformations or changes are based on standard mathematical techniques: coordinate geometry, trigonometry and matrix methods. These techniques tell us how to compute the new coordinates after applying the transformations. We will study more about transformations of objects in the coming lectures.

5.7) Concept #7: Applications of Computer Graphics

The following are also considered graphics applications:

**Paint programs:** Allow you to create rough freehand drawings. The images are stored as bit maps and can easily be edited. It is a graphics program that enables you to draw pictures on the display screen which is represented as bit maps (bit-mapped graphics). In contrast, draw programs use vector graphics (object-oriented images), which scale better. Most paint programs provide the tools shown below in the form of icons. By selecting an icon, you can perform functions associated with the tool. In addition to these tools, paint programs also provide easy ways to draw common shapes such as straight lines, rectangles, circles, and ovals.

Sophisticated paint applications are often called image editing programs. These applications support many of the features of draw programs, such as the ability to work with objects. Each object, however, is represented as a bit map rather than as a vector image.

**Illustration/design programs:** Supports more advanced features than paint programs, particularly for drawing curved lines. The images are usually stored in vector-based formats. Illustration/design programs are often called draw programs.
**Presentation graphics software:** Lets you create bar charts, pie charts, graphics, and other types of images for slide shows and reports. The charts can be based on data imported from spreadsheet applications.

A type of business software that enables users to create highly stylized images for slide shows and reports. The software includes functions for creating various types of charts and graphs and for inserting text in a variety of fonts. Most systems enable you to import data from a spreadsheet application to create the charts and graphs. Presentation graphics is often called business graphics.

**Animation software:** Enables you to chain and sequence a series of images to simulate movement. Each image is like a frame in a movie. It can be defined as a simulation of movement created by displaying a series of pictures, or frames. A cartoon on television is one example of animation. Animation on computers is one of the chief ingredients of multimedia presentations. There are many software applications that enable you to create animations that you can display on a computer monitor.

There is a difference between animation and video. Whereas video takes continuous motion and breaks it up into discrete frames, animation starts with independent pictures and puts them together to form the illusion of continuous motion.

**CAD software:** Enables architects and engineers to draft designs. It is the acronym for computer-aided design. A CAD system is a combination of hardware and software that enables engineers and architects to design everything from furniture to airplanes. In addition to the software, CAD systems require a high-quality graphics monitor; a mouse, light pen, or digitizing tablet for drawing; and a special printer or plotter for printing design specifications.

CAD systems allow an engineer to view a design from any angle with the push of a button and to zoom in or out for close-ups and long-distance views. In addition, the computer keeps track of design dependencies so that when the engineer changes one value, all other values that depend on it are automatically changed accordingly.

Until the mid 1980s, all CAD systems were specially constructed computers. Now, you can buy CAD software that runs on general-purpose workstations and personal computers.
Desktop publishing: Provides a full set of word-processing features as well as fine control over placement of text and graphics, so that you can create newsletters, advertisements, books, and other types of documents. It means by using a personal computer or workstation high-quality printed documents can be produced. A desktop publishing system allows you to use different typefaces, specify various margins and justifications, and embed illustrations and graphs directly into the text. The most powerful desktop publishing systems enable you to create illustrations; while less powerful systems let you insert illustrations created by other programs.

As word-processing programs become more and more powerful, the line separating such programs from desktop publishing systems is becoming blurred. In general, though, desktop publishing applications give you more control over typographical characteristics, such as kerning, and provide more support for full-color output.

A particularly important feature of desktop publishing systems is that they enable you to see on the display screen exactly how the document will appear when printed. Systems that support this feature are called WYSIWYGs (what you see is what you get).

Until recently, hardware costs made desktop publishing systems impractical for most uses. But as the prices of personal computers and printers have fallen, desktop publishing systems have become increasingly popular for producing newsletters, brochures, books, and other documents that formerly required a typesetter.

Once you have produced a document with a desktop publishing system, you can output it directly to a printer or you can produce a PostScript file which you can then take to a service bureau. The service bureau has special machines that convert the PostScript file to film, which can then be used to make plates for offset printing. Offset printing produces higher-quality documents, especially if color is used, but is generally more expensive than laser printing.

In general, applications that support graphics require a powerful CPU and a large amount of memory. Many graphics applications—for example, computer animation systems—require more computing power than is available on personal computers and will run only on powerful workstations or specially designed graphics computers. This is true of all three-dimensional computer graphics applications.
In addition to the CPU and memory, graphics software requires a graphics monitor and support for one of the many graphics standards. Most PC programs, for instance, require VGA graphics. If your computer does not have built-in support for a specific graphics system, you can insert a video adapter card.

The quality of most graphics devices is determined by their resolution—how many pixels per square inch they can represent—and their color capabilities.

5.8) **University Questions**

(a) What is interactive graphics system?(4 marks) [G 1693 Computer Graphics (R, T) June/ July 2006]

(b) Enumerate the applications of raster scan graphics.(4 marks) [G 1693 Computer Graphics (R, T) June/ July 2006]

(c) Explain in detail the basic concepts in Computer graphics.(12 marks) [G 1693 Computer Graphics (R, T) June/ July 2006]

(d) What are the requirements of interactive graphics? ?(4 marks) [G 1867 Computer Graphics May 2005]

(e) Explain the potential applications of computer graphics.(4 marks) [3794 Computer Graphics (R, T) November 2005]

6.0) **Summary**

In this lecture we had a first look at the basic concepts of computer graphics. We developed an idea on the history of evolution of computer graphics. We also discussed how an interactive display system works. We have also discovered the different application areas of computer graphics.

**Lecture # 2**

1) **Synopsis:** Basic Concepts in Computer Graphics, Video Display Devices.

2) **Target:** At the completion of this lecture you should be able to answer questions like
   
   a) What is a pixel?
   
   b) What is resolution?
   
   c) What is a frame buffer?
d) What is a colour frame buffer?

e) How is an image produced on the monitor?

f) What is a shadow mask CRT?

g) What is the importance of refresh rates?

h) How are refresh rates calculated?

You should also be able to explain about the design and the working of a CRT.

3) Introduction

In lecture#1 we discussed about what computer graphics is and what its applications are. We also discussed briefly on how an interactive graphics display works. As stated above, in this lecture, we will discuss in detail the different methods of display and try to understand how an image is generated and displayed on the monitor. Before we begin on this topic, a revision of the concepts discussed in the introductory lecture may be helpful. Once we finish this aspect, we will proceed towards exposition of items listed in the synopsis. In particular, we will emphasize the following

(a) Pixel

(b) Resolution

(c) Graphics Primitives and Attributes

(d) CRT design

(e) Shadow Mask CRT

(f) Importance of Refresh Rates

(g) How to calculate refresh rates?

4) Revision / Prerequisites

Refer to pages 3 to 34 of your text book.

5.1) Concept #1: pixel

A pixel (short for picture element, using the common abbreviation "pix" for "picture") is one of the many tiny dots that make up the representation of a picture in a computer's memory. Each such information element is not really a dot, nor a square, but an abstract sample. With care, pixels in an image can be reproduced at any size without the appearance of visible dots or squares; but in many contexts, they are reproduced as dots or squares and can be visibly distinct when not fine enough. The intensity of each pixel is
variable; in color systems, each pixel has typically three or four dimensions of variability such as red, green and blue, or cyan, magenta, yellow and black.

![Diagram of Picture element, or pixel](image)

5.2) Concept #2: pixel resolution

**Pixel resolution**

The term resolution is often used as a pixel count in digital imaging, even though American, Japanese, and international standards specify that it should not be so used, at least in the digital camera field. An image of N pixels high by M pixels wide can have any resolution less than N lines per picture height, or N TV lines. But when the pixel counts are referred to as resolution, the convention is to describe the pixel resolution with the set of two positive integer numbers, where the first number is the number of pixel columns (width) and the second is the number of pixel rows (height), for example as 640 by 480. Another popular convention is to cite resolution as the total number of pixels in the image, typically given as number of megapixels, which can be calculated by multiplying pixel columns by pixel rows and dividing by one million. Other conventions include describing pixels per length unit or pixels per area unit, such as pixels per inch or per square inch. None of these pixel resolutions are true resolutions, but they are widely referred to as such; they serve as upper bounds on image resolution.

Below is an illustration of how the same image might appear at different pixel resolutions, if the pixels were poorly rendered as sharp squares (normally, a smooth image reconstruction from pixels would be preferred, but for illustration of pixels, the sharp squares make the point better).
Resolution in various media

- DVDs have roughly 500 lines (or TV lines, or lines per picture height).
- High definition television has 1,080 lines.
- 35mm movie film is scanned for release on DVD at 1080 or 2000 lines as of 2005.
- 35mm optical camera negative motion picture film can resolve up to 6,000 lines.
- 35mm projection positive motion picture film has about 2,000 lines which results from the analogue printing from the camera negative of an interpositive, and possibly an internegative, then a projection positive.
- Newer films are scanned at 4,000 lines, called 4K scanning, anticipating any advances in digital projection or higher resolution in flat panel display.

5.3) Concept #3: Image resolution

Image resolution describes the detail an image holds. The term applies equally to digital images, film images, and other types of images. Higher resolution means more image detail. Image resolution can be measured in various ways. Basically, resolution quantifies how close lines can be to each other and still be visibly resolved. Resolution units can be tied to physical sizes (e.g. lines per mm, lines per inch) or to the overall size of a picture (lines per picture height, also known simply as lines, or TV lines). Furthermore, line pairs are often used instead of lines. A line pair is a pair of adjacent dark and light lines, while a line counts both dark lines and light lines. A resolution of 10 lines per mm means 5 dark lines alternating with 5 light lines, or 5 line pairs per mm. Photographic lens and film resolution are most often quoted in line pairs per mm.

Image resolution on raster displays

A television or raster image display with 525 scan lines makes a picture with somewhat less than 525 TV lines of resolution. The ratio of lines of resolution to the number of format lines in known as the Kell
factor, after Raymond D. Kell, who worked out details of visual resolution in scanned systems at RCA in the 1930s.

5.4) Concept #4: Graphics Primitives and Attributes

Even the most complex computer-generated graphic images are produced by a relatively small set of Graphics Primitives. The usual sets of basic primitives provided in Graphics Packages are:

1. A single point.
2. A line with given end-points.
3. A polyline i.e. a line joining a sequence of points.
4. A filled polygonal area with given points as vertices.
5. Text.

There may be additional, but not essential, primitives such as rectangles, circles, curves of various types, images etc.

Associated with each graphics primitive is a set of Attributes. The attributes of a primitive determine its appearance on the output device. For example a line commonly has attributes such as colour, width, style (full, dotted, dashed etc.) etc.

5.5) Concept #5: Frame Buffer

A frame buffer is a large, contiguous piece of computer memory. At a minimum there is one memory bit for each pixel in the rater; this amount of memory is called a bit plane. The picture is built up in the frame buffer one bit at a time. You know that a memory bit has only two states, therefore a single bit plane yields a black-and-white display. You know that a frame buffer is a digital device and the CRT is an analog device. Therefore, a conversion from a digital representation to an analog signal must take place when information is read from the frame buffer and displayed on the raster CRT graphics device. For this you can use a digital to analog converter (DAC). Each pixel in the frame buffer must be accessed and converted before it is visible on the raster CRT.
A single bit-plane black white frame buffer raster CRT graphics device

5.6) Concept #6: N-bit colour Frame buffer

Color or gray scales are incorporated into a frame buffer raster graphics device by using additional bit planes. The intensity of each pixel on the CRT is controlled by a corresponding pixel location in each of the N bit planes. The binary value from each of the N bit planes is loaded into corresponding positions in a register. The resulting binary number is interpreted as an intensity level between 0 (dark) and $2^n$-1 (full intensity). This is converted into an analog voltage between 0 and the maximum voltage of the electron gun by the DAC. A total of $2^N$ intensity levels are possible. Figure given below illustrates a system with 3 bit planes for a total of 8 ($2^3$) intensity levels. Each bit plane requires the full complement of memory for a given raster resolution; e.g., a 3-bit plane frame buffer for a 1024 X1024 raster requires 3,145,728 (3 X 1024 X1024) memory bits.
An increase in the number of available intensity levels is achieved for a modest increase in required memory by using a lookup table. Upon reading the bit planes in the frame buffer, the resulting number is used as an index into the lookup table. The look up table must contain $2^N$ entries. Each entry in the lookup table is $W$ bit wise. $W$ may be greater than $N$. When this occurs, $2^W$ intensities are available; but only $2^N$ different intensities are available at one time. To get additional intensities, the lookup table must be changed.

Because there are three primary colours, a simple color frame buffer is implemented with three bit planes, one for each primary color. Each bit plane drives an individual color gun for each of the three primary colors used in color video. These three primaries (red, green, and blue) are combined at the CRT to yield eight colors.
5.7) Concept #7: The working of a Cathode Ray Tube

The primary output device in a graphics system is a video monitor. The operation of most video monitors is based on the standard cathode ray tube (CRT) design.

The Cathode Ray Tube

Operation of an electron gun with an accelerating anode
Basic design of a magnetic deflection CRT

Electrostatic deflection of the electron beam in a CRT
The electron gun emits a beam of electrons which are focused to a point on the screen phosphor. The beam is positioned on the screen by a deflection system which operates in the horizontal and vertical directions. The intensity of the beam is controlled by the intensity signal on the control grid. When the phosphor is hit by the electron beam it absorbs energy and jumps to a higher quantum-energy level. As it returns to its normal level it emits visible light i.e. it phosphoresces. In the phosphors used in graphics devices the persistence of the phosphorescence is typically 10-60 microseconds.

Before the human visual system can see a transient image it must be continually redrawn (refreshed) at a rate higher than the critical fusion frequency of the human visual system. To allow the human visual system to see a continuously refreshed image without flicker the refresh rate has to be at least 60 c/s.

To allow continuous refreshing of an image there must be some stored representation of the image from which the refresh system can obtain the graphical information required to re-draw the image. This representation nowadays is invariably a set of values of intensity/colour at each of a discrete set of points laid out in a rectangular array covering the screen.

While it may seem a disadvantage to continually refresh the image there are some very important advantages of such refresh type systems. For example it is possible to edit an image by changing the stored representation between refresh cycles for what appears to be instantaneous updating of the image. Compare this with some earlier systems in which the only way to carry out an edit was to clear the whole screen and then redraw the whole image. Also by changing the stored representation between refresh cycles animation is possible.

5.8) Concept #8: Shadow Mask CRT

In Shadow Mask CRT tiny holes in a metal plate separate the colored phosphors in the layer behind the front glass of the screen. The holes are placed in a manner ensuring that electrons from each of the tube's three cathode guns reach only the appropriately-colored phosphors on the display. All three beams pass through the same holes in the mask, but the angle of approach is different for each gun. The spacing of the holes, the spacing of the phosphors, and the placement of the guns is arranged so that for example the blue gun only has an unobstructed path to blue phosphors. The red, green, and blue phosphors for each pixel are generally arranged in a triangular shape (sometimes called a "triad"). All early color televisions and the majority of computer monitors, past and present, use shadow mask technology.
Traditionally, shadow masks have been made of materials which temperature variations cause to expand and contract to the point of affecting performance. The energy the shadow mask absorbs from the electron gun in normal operation causes it to heat up and expand, which leads to blurred or discolored (see doming) images. The invar shadow mask is composed of the nickel-iron alloy invar. Therefore it expands and contracts much less than other materials in response to temperature changes. This property allows displays made with this technology to provide a clearer, more accurate picture. It also reduces the amount of long-term stress and damage to the shadow mask that can result from repeated expand/contract cycles, thus increasing the display's life expectancy.

In other words, In Shadow Mask CRT, before the stream of electrons produced by the CRT's cathode reach the phosphor coated faceplate, it encounters the shadow mask, a sheet of metal etched with a pattern of holes. The mask is positioned in the glass funnel of the CRT during manufacture and the phosphor is coated onto the screen so that electrons coming from the red, green and blue gun positions only land on the appropriate phosphor.

Stray electrons strike the shadow mask and are absorbed by it, generating a great deal of heat, which in turn causes the metal to expand. To allow flatter CRTs to be made, the metal most commonly used now for shadow masks is Invar, an alloy of iron and nickel. The metal has a low coefficient of expansion and its name derives from the supposed invariability of its dimensions when heat is applied. In reality, its dimensions are not completely invariable and the build up of heat in a shadow mask can lead to a form of distortion known as doming, where the centre of the mask bulges towards the faceplate slightly.
An alternative to the shadow mask which is less prone to distortion, the aperture grille, was included as part of the design of Trinitron CRTs by Sony in 1968 and Mitsubishi in its Diamondtron products in the early 1990s.

5.9) Concept #9: Importance of Refresh Rates

When choosing a monitor, one of the factors that the customer usually considers is the refresh rate. A high refresh rate is important in providing a clear picture and avoiding eye fatigue.

What is a refresh rate and why is a monitor's refresh rate important?

An image appears on screen when electron beams strike the surface of the screen in a zig-zag pattern. A refresh rate is the number of times a screen is redrawn in one second and is measured in Hertz (Hz). Therefore, a monitor with a refresh rate of 85 Hz is redrawn 85 times per second. A monitor should be "flicker-free meaning that the image is redrawn quickly enough so that the user cannot detect flicker, a source of eye strain. Today, a refresh rate of 75 Hz or above is considered to be flicker-free.

5.10) Concept #10: How are refresh rates calculated?

Factors in determining refresh rates

A refresh rate is dependent upon a monitor's horizontal scanning frequency and the number of horizontal lines displayed. The horizontal scanning frequency is the number of times the electron beam sweeps one line and returns to the beginning of the next in one second. Horizontal scanning frequency is measured in kilohertz (kHz). A monitor with a horizontal scanning frequency of 110 kHz means 110,000 lines are scanned per second.
The number of horizontal lines on the screen depends upon the monitor's resolution. If a monitor is set to a resolution of 1024 x 768 then there are 768 horizontal lines (1024 is the number of pixels on one line). For a monitor set to a 1280 x 1024 resolution, there are 1024 horizontal lines.

Additionally, the time it takes for the electron beam to return to the top of the screen and begin scanning again must be taken into account. This is roughly 5% of the time it takes to scan the entire screen. Therefore, the total is multiplied by 0.95 to calculate the maximum refresh rate.

How to calculate maximum refresh rates?
The following formula is used to calculate maximum refresh rates
\[ f_V = \frac{f_H}{\text{# of horizontal lines}} \times 0.95 \]
\[ f_V = \text{vertical scanning frequency (refresh rate)} \]
\[ f_H = \text{horizontal scanning frequency} \]

Example: A monitor with a horizontal scanning frequency of 96 kHz at a resolution of 1280 x 1024 would have the following refresh rate based on the calculation above.

\[ f_V = \frac{96,000}{1024} \times 0.95 \]
\[ f_V = 89.06 \]
This figure is rounded down to produce a maximum refresh rate of 89Hz.

If the same monitor is set to a resolution of 1600 x 1200, then the equation will be as follows:

\[ f_V = \frac{96,000}{1200} \times 0.95 \]
\[ f_V = 76 \]
The maximum refresh rate at this resolution is 76 Hz.

**5.11) University Questions**

(f) Explain with a neat diagram the working of a shadow mask CRT bringing out its merits and demerits.(12 marks) [G 1867 Computer Graphics May 2005]
6.0) Summary

In this lecture we familiarized the terms pixel and resolution. We learned that the primary output device in a graphics system is a video monitor and its operation is based on CRT design. We also discussed the working of a Shadow mask CRT. We learned how refresh rates of a monitor is calculated.

7) Exercise questions

7.0) Consider three different raster systems with resolutions of 640 by 480, 1280 by 1024, and 2560 by 2048. What size frame buffer (in bytes) is needed for each of these systems to store 12 bits per pixel? How much storage is required for each system if 24 bits per pixel are to be stored?

7.1) How long would it take to load a 640 by 480 frame with 12 bits per pixel, if 10^5 can be transferred per second? How long will it take to load a 24 bit per pixel frame buffer with a resolution of 1280 by 1024 using this same transfer rate?

7.2) Compute the size of a 640X480 image at 240 pixels per inch.

7.3) Compute the resolution of a 2 X 2 inch image that has 512 X 512 pixels.

7.4) What is an image’s aspect ratio?

Lecture # 3

1) Synopsis: Raster scan and Random scan displays – Generating a raster image

2) Target: At the completion of this lecture you should be able to answer questions like
   i) What is rasterization?
   j) What is raster scan display?
   k) What is random scan display?

You will also be able to explain the Raster Graphics Display Architecture.

3) Introduction
In lecture#1 we discussed about what computer graphics is and what its applications are. We also discussed briefly on how an interactive graphics display works. In lecture#2 we learned how an image is stored in the frame buffer and how it is converted into video signals. We have learned the working of the CRT monitor. Keeping all these data in mind let us proceed to the next topic, rasterization. As stated above, in this lecture, we will discuss in detail the different methods of display and try to understand how an image is generated and displayed on the monitor. Before we begin on this topic, a revision of the concepts discussed in the introductory lecture may be helpful. Once we finish this aspect, we will proceed towards exposition of items listed in the synopsis. In particular, we will emphasize the following

(h) Raster Scan display
(i) Random Scan Display
(j) Colour in Raster Graphics

4) Revision / Prerequisites
Refer to pages 3 to 34 of your text book.

5.1) Concept #1: Raster Graphics Display Architecture
The figure which describes the architecture of Raster graphics is given below.
It includes:
1. Display controller
2. Refresh Buffer
3. Video Controller
4. CRT monitor
5.1) Concept #1: Raster Graphics

A raster graphics image, digital image, or bitmap, is a data file or structure representing a generally rectangular grid of pixels, or points of color, on a computer monitor, paper, or other display device. The color of each pixel is individually defined; images in the RGB color space, for instance, often consist of colored pixels defined by three bytes—one byte each for red, green and blue. Less colorful images require less information per pixel; an image with only black and white pixels requires only a single bit for each pixel. Raster graphics are distinguished from vector graphics in that vector graphics represent an image through the use of geometric objects such as curves and polygons.

A colored raster image (or pixmap) will usually have pixels with between one and eight bits for each of the red, green, and blue components, though other color encodings are also used, such as four- or eight-bit indexed representations that use vector quantization on the (R, G, B) vectors. The green component sometimes has more bits than the other two to allow for the human eye's greater discrimination in this component.

The quality of a raster image is determined by the total number of pixels (resolution), and the amount of information in each pixel (often called color depth). For example, an image that stores 24 bits of color information per pixel (the standard for all displays since around 1995) can represent smoother degrees of shading than one that only stores 16 bits per pixel, but not as smooth as one that stores 48 bits (technically; this would not be discernible by the human eye). Likewise, an image sampled at 640 x 480 pixels (therefore
containing 307,200 pixels) will look rough and blocky compared to one sampled at 1280 x 1024 (1,310,720 pixels). Because it takes a large amount of data to store a high-quality image, data compression techniques are often used to reduce this size for images stored on disk. Some techniques sacrifice information, and therefore image quality, in order to achieve a smaller file size. Compression techniques that lose information are referred to as "lossy" compression.

Raster graphics cannot be scaled to a higher resolution without loss of apparent quality. This is in contrast to vector graphics, which easily scale to the quality of the device on which they are rendered. Raster graphics are more practical than vector graphics for photographs and photo-realistic images, while vector graphics are often more practical for typesetting or graphic design. Modern computer monitors typically display about 72 to 130 pixels per inch (PPI), and some modern consumer printers can resolve 2400 dots per inch (DPI) or more; determining the most appropriate image resolution for a given printer resolution can be difficult, since printed output may have a greater level of detail than can be discerned on a monitor.

**Raster example**

To illustrate the matter further, here is the letter "J":

J

A close look at the letter will appear as such, where the "X" and "." characters represent a grid of pixels:

.....X
.....X
.....X
.....X
.....X
.....X
.XX.X
.XX.X
.AXX..

A computer sees something more like this, where "." represents a zero and "X" represents a one:

000001
Where a zero appears, the computer software instructs its video hardware to paint the current background color. A one calls for the current foreground color. The software makes a distinction between the colors of adjacent pixels, which together form an image. This is the basic principle behind graphics editing on a computer.

Raster graphics was first patented by Texas Instruments in the 1970s, and is now ever-present.

5.2) Concept #2: Rasterization

Rasterization is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer.

Raster Display
System Components

- The screen is subdivided into a matrix of pixels (smallest addressable units).
Raster scanline -- A line of pixels along the screen

- Frame (refresh) buffer -- Block of memory used to store the screen pattern

How it works

1. The DISPLAY PROCESSOR produces the raster image in the frame buffer from the commands
2. The VIDEO CONTROLLER moves the beam row wise across the pixels setting it on and off according to the content of the frame buffer
3. The display must be refreshed to avoid flickering (raster image redisplayed 30 to 60 times per second)
5.3) Concept #3: Raster Scan Display

Raster Scan methods have increasingly become the dominant technology since about 1975. These methods use the TV type raster scan. The growth in the use of such methods has been dependent on rapidly decreasing memory prices and on the availability of cheap scan generating hardware from the TV industry.
The screen is coated with discrete dots of phosphor, usually called pixels, laid out in a rectangular array. The image is then determined by how each pixel is intensified. The representation of the image used in servicing the refresh system is thus an area of memory holding a value for each pixel. This memory area holding the image representation is called the frame buffer.

The values in the frame buffer are held as a sequence of horizontal lines of pixel values from the top of the screen down. The scan generator then moves the beam in a series of horizontal lines with fly-back (non-intensified) between each line and between the end of the frame and the beginning of the next frame. This is illustrated below.

Unlike random-scan which is a line drawing device, refresh CRT is a point-plotting device. Raster displays store the display primitives (lines, characters, shaded and patterned areas) in a refresh buffer. Refresh buffer (also called frame buffer) stores the drawing primitives in terms of points and pixels components.
This scan is synchronized with the access of the intensity values held in the frame buffer.

The maximum number of points that can be displayed without overlap by a system is called the resolution and is quoted as the number of points per horizontal line versus the number of horizontal lines. Typical resolutions are 640*480, 1024*768, and 1280*1024. The maximum resolution is determined by the characteristics of the monitor and/or by the memory capacity available for storing the frame buffer.

5.4) Concept #4: Random Scan Display

Random scan displays, often termed vector, Stroke, and Line drawing displays, came first and are still used in some applications. Here the characters are also made of sequences of strokes (or short lines). The electron gun of a CRT illuminates straight lines in any order. The display processor repeatedly reads a variable 'display file' defining a sequence of X,Y coordinate pairs and brightness or color values, and converts these to voltages controlling the electron gun.
In random scan display an electron beam is deflected from endpoint to end-point. The order of deflection is dictated by the arbitrary order of the display commands. The display must be refreshed at regular intervals – minimum of 30 Hz (fps) for flicker-free display.

Ideal line drawing

Random Scan Display
5.5) Concept #5: Colour in Raster Graphics

To handle colour the phosphor is applied to the screen in small triads of red, green and blue phosphor dots, each triad representing one pixel. The CRT then has an electron gun for each of the additive primary colours red, green and blue. When the three primary colour dots are intensified the human visual system combines the triad of primary colours into a single perceived colour.

To ensure the beams hit the correct elements of each triad there is a thin perforated metal sheet (the shadow mask) behind the phosphor that ensures each of the three beams hits the correct element of the triad.

5.6) University Questions

(g) Write short note on Raster Scan displays (6 marks) [F 3875 Computer Graphics November 2005]

(h) Explain the following in detail a) Generation of a raster image.(6 marks) [3794 Computer Graphics (R, T) November 2005]

6.0) Summary

We discussed about the two display methods and understood that a raster system uses a frame buffer to store intensity information for each screen position (pixel). Pictures are then painted on the screen by retrieving this information from the frame buffer as the electron beam in the CRT sweeps across each scan line, from top to bottom. In random scan display we know that pictures are constructed by drawing lines between specified line endpoints. Picture information is then stored as a set of line drawing instructions.

7) Exercise questions

7.0) What do you call the path the electron beam takes when returning to the left side of the CRT screen?

7.1) What do you call the path the electron beam takes at the end of each refresh cycle?

7.2) List the applications of Raster scan graphics.
1) Synopsis: Raster Scan and random Scan Systems, Display Processors, Display Files.

2) Target: At the completion of this lecture you should be able to answer questions like
   (g) What are raster scan systems?
   (h) What is the function of a video controller?
   (i) What is the function of a Display processor?
   (j) What are random scan systems?

You should also be able to explain the working of a video controller refresh buffer.

3) Introduction

   In lecture 2 and 3 we discussed about different display devices and the display methods. As stated above, in this lecture, we plan to explore what is the difference between raster scan systems and random scan systems. We will discuss their architecture and will define the functions performed by a video controller and a display processor. Before we begin on this topic, a revision of the concepts developed earlier in lecture 2 and 3 may be helpful. Hence I advise you to have a look at it before starting off. Once you finish this aspect, we will proceed towards exposition of items listed in the synopsis.

   In particular, we will emphasize the following
   (a) Raster Scan Systems
   (b) Video Controller
   (c) Video Controller refresh operations
   (d) Display Processor
   (e) Random Scan Systems

4) Revision / Prerequisites

   Please refer to pages 36 to 52 of your text book

5.1) Concept #1: Raster Scan Systems

   You know that interactive raster graphics system typically employs several processing units. In addition to central processing unit, or CPU, there is a special –purpose processor, called the video controller or display controller. It is used to control the operation of the display device.
Architecture of a simple raster graphics system

Here the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen.

5.2) Concept #2: Video Controller

In commonly used raster systems a fixed area is reserved for the frame buffer, and the video controller is given direct access to the frame buffer memory. Frame buffer locations and the corresponding screen positions are referenced in Cartesian coordinates.

Architecture of a raster system with a fixed portion of the system memory reserved for the frame buffer

For the graphics monitor, the origin I defined at the lower left screen corner. The screen is represented as a two dimensional system i.e. the x values increasing to the right and the y values increasing from bottom to top. Each scan line is labeled as $y_{\text{max}}$ at the top of the screen to 0 at the bottom. Along each scan line, screen pixel positions are labeled from 0 to $x_{\text{max}}$.

5.3) Concept #3: Refresh operation of the video buffer

In the basic refresh operation, two registers are used. The purpose of this registers are to store the coordinates of the screen pixels. Initially the x register is set to 0 and the y register is set to $y_{\text{max}}$. The value
stored in the frame buffer for this pixel position is retrieved and used to set the intensity of the CRT beam. Then the x register is incremented by 1, and the process repeated for the next pixel on the top scan line. This is repeated for each pixel along the scan line.

After the last pixel on the top of the scan line has been processed, the x register is reset to 0 and the y register is decremented by 1. The pixels along this scan line are then processed and the procedure is repeated for each successive scan line.

After cycling through all the pixels along the bottom scan line (y=0), the video controller resets the registers to the first pixel position on the top scan line and refresh process starts over.

As you know, a screen has to be refreshed at the rate of 60 frames per second. So this cannot be accommodated in RAM chips. Therefore the cycle time is too low and as a result the pixel processing is too slow. Now what can be done to speed up this process? Multiple pixels can be retrieved in one pass i.e. video controller retrieve multiple pixel values from the refresh buffer on each pass.

Then how do you store all these different intensity values? They are stored using separate registers and are used to control the CRT beam intensity for a group of adjacent pixels. When that group of pixels has been processed; the next block of pixel values is retrieved from the frame buffer.

In high quality systems two buffers are used so that one is used for refreshing and other is being filled with intensity values. Then the two buffers can switch the roles. This provides a fast mechanism for generating real time animations, since different views of moving objects can be successively loaded into the refresh buffer.
5.4) Concept #4: Raster Scan Display processor

Raster Scan system may contain a separate display processor.

Raster Scan Display processor is sometimes referred to as graphics controller or a display coprocessor. The purpose of this processor is to free the CPU from the graphics tasks. But what is its major task? Its major task is to digitize a picture definition given in an application program into a set of pixel intensity values for storage in the frame buffer. This digitization process is called scan conversion. Graphics commands specifying straight lines and other geometric shapes can be converted into a set of intensity points. When you scan convert a straight line segment we have to locate the pixel positions closest to the line path and store the intensity for each position in the frame buffer.

Characters can be defined with rectangular grids or they can be defined with outlines. The array size for character grids can vary from about 5 by 7 or 9 by 12.

-defined as a grid of pixel position
-defined as a curve outline
Display processor are designed to perform different other functions. They are

- To generate various line styles (dashed, dotted, or solid)
- To display colour areas
- To perform certain manipulations and transformations on displayed objects
- To interface with interactive input devices, such as mouse

How is the frame buffer implemented?

It is implemented as a linked list and encodes the intensity information. One way to do this is to store each scan line as a set of integer pairs. One number of each pair indicates an intensity value and the second number specifies the number of adjacent pixels on the scan line that have that intensity. This technique is called run length encoding.

5.5) Concept #5: Random Scan Systems

In Random Scan Systems an application program is input and stored in the system memory along with a graphics package. Graphics commands in the application program are translated by the graphics package into a display file stored in the system memory. This display file is then accessed by the display processor to refresh the screen. The display processor cycles through each command in the display file program once during every refresh cycle. Sometimes the display processor in the random scan system is referred to as a display processing unit or a graphics controller. Graphics patterns are drawn on a random scan system by directing the electron beam along the component lines of picture. Lines are defined by the values for their coordinate endpoints, and these input coordinate values are converted to x and y deflection voltage. A scene is then drawn one line at a time by positioning the beam to fill in the line between specified endpoints.

5.6) University Questions

(i) Explain in detail the basic concepts in Computer Graphics (12 marks) [G 1693 Computer Graphics (R) June/July 2006]

(j) Explain and compare the working of raster scan and random scan display systems. (12 marks) [G 1867 Computer Graphics (R) May 2005]

(k) Write short note on display file interpreter. (6 marks) [F 3879 Computer Graphics (R) November 2005]

(l) Explain with the help of a block diagram, display processor for a refresh line drawing display. (12 marks) [Computer Graphics (R) old scheme November 2001]
Using an illustrate example, explain how a picture can be represented using linked lists. (12 marks) [Computer Graphics (R) old scheme November 2001]

Explain in detail the display processor (6 marks) [3794 Computer Graphics (R) November 2005]

Explain in detail the generation of a raster image (6 marks) [3794 Computer Graphics (R) November 2005]

6.0) Summary

In this lecture we have discussed about raster scan systems and random scan systems. We have looked into their design issues and we saw the functions performed by each one. We are now familiarized with the functions of the video controller and the display processor. We can also list out the differences between random scan and raster scan systems.

7) Exercise questions

7.0) List out the differences between random scan systems and raster scan systems.

Lecture # 5

1) Synopsis: Line drawing Algorithms – DDA

2) Target: At the completion of this lecture you should be able to answer questions like

   (k) What is rasterization?
   (l) What is scan conversion?
   (m) How do you scan convert a line?
   (n) How do you display straight lines?
   (o) How is DDA algorithm implemented?

You should also be able to explain the mathematical and algorithmic aspects of the DDA scan conversion method.

3) Introduction

In the introductory lecture of this course we have discussed about the key features and applications of computer graphics. We have discussed that the end product of computer graphics is a picture. In its broadest sense you know that a picture means any collection of lines, points, text, etc. that can be displayed on a graphics device. In your Higher Secondary classes you have learned that the fundamental building block of picture representation is a point and the algorithms that organize these points. As stated above, in this lecture, we plan to explore how the graphics system convert the graphical primitives such as points and lines from its geometrical definition into a set of pixels that make up the primitive in the image space. We will also discuss
on DDA algorithm that helps in drawing a straight line from one point to another. Before we begin on this topic, a revision of the concepts developed earlier like CRT raster display, pixels -the smallest picture unit accepted by the display and the Cartesian slope intercept equation for straight lines may be helpful. Once we finish this aspect, we will proceed towards exposition of items listed in the synopsis. In particular, we will emphasize the following

(f) Rasterization and Scan Conversion  
(g) Scan converting a point  
(h) Design criteria of straight lines  
(i) Scan converting a line  
(j) Implementing the Digital Differential Analyzer

4) Revision / Prerequisites

You know that computer graphics is a complex and diversified technology. To begin to understand the technology, it is necessary to subdivide it into manageable parts. This can be accomplished by considering that the fundamental cohesive concept in computer graphics is a picture. We must therefore consider:

(a) How pictures are represented in computer graphics  
(b) How pictures are prepared for presentation  
(c) How previously prepared pictures are presented

Representing pictures

Points are the fundamental building blocks of picture representation. Of equal fundamental importance is the algorithm, which explains how to organize these points. To illustrate this consider a unit square in the first quadrant. Refer the figure given below

The unit square can be represented by its four corners points  
P1 (0, 0)  P2 (1, 0)  P3 (1, 1)  P4 (0, 1)

An associated algorithm description might be
Connect P1P2P3P4P1 in sequence

The unit square can also be described by its four edges

\[ E_1 = P_1P_2 \quad E_2 = P_2P_3 \quad E_3 = P_3P_4 \quad E_4 = P_4P_1 \]

Here, the algorithm description is

Connect E1E2E3E4 in sequence

The fundamental building blocks, i.e., points can be represented as either pairs or triplets of numbers, depending whether the data are two or three dimensional. Thus \((x_1, y_1)\) or \((x_1, y_1, z_1)\) represent a point in either two or three dimensional space. Two points represents a line or edge, and a collection of three or more points represents a polygon. The representation of curved lines is usually accomplished by approximating them by connected short straight line segments.

*Preparing pictures for presentation*

Pictures ultimately consist of points and a drawing algorithm to display them. This information is generally stored in a file before it is used to present the picture; this file is called a data base. Very complex pictures require very complex data bases, which require a complex algorithm to access them.

*Presenting previously prepared pictures*

The data used to prepare the picture for presentation is rarely the same as that used to present the picture. The data used to present the picture is frequently called a display file. The display file represents some portion, view or scene of the picture represented by the total database.

Also, refer to pages 86 to 94 of your text book.

### 5.1) Concept #1: Raster CRT graphics device

Before discussing what rasterization is we first look at some fundamentals of cathode ray tubes and how they are used in computer graphics. The most common method to implement a raster CRT graphics device is to use a frame buffer. A frame buffer is a large, contiguous piece of computer memory. At a minimum there is one memory bit for each pixel in the raster; this amount of memory is called a bit plane. The picture is built up in the frame buffer one bit at a time. You know that a memory bit has only two states, therefore a single bit plane yields a black-and white display. You know that a frame buffer is a digital device and the CRT is an analog device. Therefore, a conversion from a digital representation to an analog signal must take place when information is read from the frame buffer and displayed on the raster CRT graphics device. For this you can use a digital to analog converter (DAC). Each pixel in the frame buffer must be accessed and converted before it is visible on the raster CRT.
Color or gray scales are incorporated into a frame buffer rater graphics device by using additional bit planes. The intensity of each pixel on the CRT is controlled by a corresponding pixel location in each of the N bit planes. The binary value from each of the N bit planes is loaded into corresponding positions in a register. The resulting binary number is interpreted as an intensity level between 0 (dark) and $2^n - 1$ (full intensity). This is converted into an analog voltage between 0 and the maximum voltage of the electron gun by the DAC.

5.2) Concept #2: Rasterization of straight lines

You know that a raster graphics device can be considered as a matrix of discrete cells, each of which can be made bright. You can find that it is not possible to directly draw a straight line from one addressable point, or pixel, in the matrix to another. The line can be approximated by a series of dots (pixels) close to the path of the line. The process of determining which pixels provides the best approximation to the desired line is properly known as rasterization. Only in a special case of completely horizontal, vertical or for square pixels 45° lines, does a straight line or pixels result. All the other lines appear as a series of stair steps; this is called aliasing, or the ‘jaggies’.

5.3) Concept #3: Scan Conversion

When the process of rasterization is combined with the process of generating a picture in scan line order, it is known as scan conversion.

5.4) Concept #4: Scan converting a point

A mathematical point $(x, y)$ where $x$ and $y$ are real numbers within an image area, needs to be scan converted to a pixel at location $(x', y')$. This may be done by making $x'$ to be the integer part of $x$, and $y'$ to be the integer part of $y$. In other words, $x' = \text{floor}(x)$ and $y' = \text{floor}(y)$, where function floor returns the largest integer that is less than or equal to the arguments. Doing so in essence places the origin of a continuous coordinate system for $(x, y)$ at the lower left corner of the pixel grid in the image space. All the points that satisfy $x' \leq x \leq x' + 1$ and $y' \leq y \leq y' + 1$ are mapped to pixel $(x', y')$.

Let us take for example a point P1(1.7, 0.8). It will be represented by pixel (1, 0). Points P2 (2.2, 1.3) and P3(2.8, 1.9) are both represented by pixel (2, 1).

Let us take another approach to align the integer values in the coordinate system for $(x, y)$ with the pixel coordinates. Here we can convert $(x, y)$ by making $x' = \text{floor}(x + 0.5)$ and $y' = \text{floor}(y + 0.5)$. This approach places the origin of the coordinate system for $(x, y)$ at the center of pixel(0,0). All points that satisfy
\[ x' - 0.5 \leq x \leq x' + 0.5 \text{ and } y' - 0.5 \leq y \leq y' + 0.5 \] are mapped to pixel \((x', y')\). This means that points \(P1\) and \(P2\) are now both represented by pixel \((2, 1)\), whereas point \(P3\) is represented by pixel \((3, 2)\).

5.5) Concept #5: Scan converting a line

You know that a line in computer graphics typically refers to a line segment, which is a portion of a straight line that extends indefinitely in opposite directions. You can define a line by its two end points and by the line equation \(y = mx + c\), where \(m\) is called the slope and \(c\) the y intercept of the line. Let the two end points of a line be \(P1(x1, y1)\) and \(P2(x2, y2)\). The line equation describes the coordinates of all the points that lie between the two endpoints.

A simple approach to scan convert a line is to first scan convert \(P1\) and \(P2\) to pixel coordinates \((x1', y1')\) and \((x2', y2')\) respectively. Then let us set \(m = (y2' - y1')/(x2' - x1')\) and \(b = y1' - mx1'\). Find \(|m|\) and if \(|m| \leq 1\), then for every integer value of \(x\) between and excluding \(x1'\) and \(x2'\), calculate the corresponding value of \(y\) using the equation and scan convert \((x, y)\). If \(|m| > 1\), then for every integer value of \(y\) between and excluding \(y1'\) and \(y2'\), calculate the corresponding value of \(x\) using the equation and scan convert \((x, y)\).

5.6) Concept #6: Design criteria of straight lines

From geometry we know that a line, or line segment, can be uniquely specified by two points. From algebra we also know that a line can be specified by a slope, usually given the name \(m\) and a y-axis intercept called \(b\). Generally in computer graphics, a line will be specified by two endpoints. But the slope and y-intercept are often calculated as intermediate results for use by most line-drawing algorithms.

The goal of any line drawing algorithm is to construct the best possible approximation of an ideal line given the inherent limitations of a raster display. Before discussing specific line drawing algorithms, it is useful to consider general requirements for such algorithms. Let us see what are the desirable characteristics needed for these lines.

The primary design criteria are as follows

- Straight lines appear as straight lines
- Straight lines start and end accurately
- Displayed lines should have constant brightness along their length, independent of the line length and orientation.
- Lines should be drawn rapidly
5.7) Concept #7: Digital Differential Analyzer

DDA algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. The unit steps are always along the coordinate of greatest change, e.g. if $dx = 10$ and $dy = 5$, then we would take unit steps along $x$ and compute the steps along $y$.

Suppose at step $i$ we have calculated $(x_i, y_i)$ to be a point on the line. Since the next point $(x_{i+1}, y_{i+1})$ should satisfy $\Delta y/\Delta x = m$ where $\Delta y = y_{i+1} - y_i$ and $\Delta x = x_{i+1} - x_i$, we have $y_{i+1} = y_i + m\Delta x$ or $x_{i+1} = x_i + \Delta y/m$.

These formulas are used in DDA algorithm as follows. When $|m| \leq 1$, we start with $x = x_1' \leq (\text{assuming that } x_1' < x_2')$ and $y = y_1'$, and set $\Delta x = 1$ (i.e., unit increment in the $x$ direction). The $y$ coordinate of each successive point on the line is calculated using $y_{i+1} = y_i + m$. When $|m| > 1$, we start with $x = x_1'$ and $y = y_1'$ (assuming that $y_1' < y_2'$), set $\Delta y = 1$ (i.e., unit increment in the $y$ direction). The $x$ coordinate of each successive point on the line is calculated using $x_{i+1} = x_i + 1/m$. This process continues until $x$ reaches $x_2'$ (for $m| \leq 1$ case) or $y$ reaches $y_2'$ (for $m| > 1$ case) and all points are scan converted to pixel points.

The explanation is as follows: In DDA algorithm we have to find the new point $x_{i+1}$ and $y_{i+1}$ from the existing points $x_i$ and $y_i$. As a first step here we identify the major axis and the minor axis of the line to be drawn. Once the major axis is found we sample the major axis at unit intervals and find the value in the other axis by using the slope equation of the line. For example if the end points of the line is given as $(x_1, y_1) = (2,2)$ and $(x_2, y_2) = (9,5)$. Here we will calculate $y_2-y_1$ and $x_2-x_1$ to find which one is greater. Here $y_2-y_1 = 3$ and $x_2-x_1 = 7$; therefore here the major axis is the $x$ axis. So here we need to sample the $x$ axis at unit intervals i.e. $\Delta x = 1$ and we will find out the $y$ values for each $\Delta x$ in the $x$ axis using the slope equation.

In DDA we need to consider two cases; one is slope of the line less than or equal to one ($|m| \leq 1$) and slope of the line greater than one ($|m| > 1$). When $|m| \leq 1$ means $y_2-y_1 = x_2-x_1$ or $y_2-y_1 < x_2-x_1$. In both these cases we assume $x$ to be the major axis. Therefore we sample $x$ axis at unit intervals and find the $y$ values corresponding to each $x$ value. We have the slope equation as

$$\Delta y = m \Delta x$$

$$y_{2-y1} = m (x_{2-x1})$$

In general terms we can say that $y_{i+1} = m(x_{i+1} - x_i)$. But here $\Delta x = 1$; therefore the equation reduces to $y_{i+1} = y_i + m = y_i + dy/dx$.

When $m > 1$ means $y_2-y1 > x_2-x1$ and therefore we assume $y$ to be the major axis. Here we sample $y$ axis at unit intervals and find the $x$ values corresponding to each $y$ value. We have the slope equation as
\[ \Delta y = m \Delta x \]
\[ y_2 - y_1 = m(x_2 - x_1) \]

In general terms we can say that \[ y_{i+1} - y_i = m(x_{i+1} - x_i) \]. But here \( \Delta y = 1 \); therefore the equation reduces to \( 1 = m(x_{i+1} - x_i) \). Therefore
\[ x_{i+1} = x_i + \frac{1}{m} \]
\[ x_{i+1} = x_i + \frac{dx}{dy} \]

DDA Algorithm is given below:

```pseudocode
procedure DDA( x1, y1, x2, y2: integer);
var
    dx, dy, steps: integer;
    x_inc, y_inc, x, y: real;
begin
    dx := x2 - x1; dy := y2 - y1;
    if abs(dx) > abs(dy) then
        steps := abs(dx);  {steps is larger of dx, dy}
    else
        steps := abs(dy);
    x_inc := dx/steps; y_inc := dy/steps;
    {either x_inc or y_inc = 1.0, the other is the slope}
    x:=x1; y:=y1;
    set_pixel(round(x), round(y));
    for i := 1 to steps do
        begin
            x := x + x_inc;
            y := y + y_inc;
            set_pixel(round(x), round(y));
        end;
    end;  {DDA}
end;
```

The DDA algorithm is faster than the direct use of the line equation since it calculates points on the line without any floating point multiplication.

**5.8) University Questions**
(p) What do you mean by scan conversion? Explain its significance in display devices. (4 marks) [F3879 Computer Graphics (R) November 2005]

(q) Discuss why incremental methods are used in line drawing in graphic displays. (6 marks) [Computer Graphics (R) November 2001]

(r) What is real time scan conversion? Explain. (8 marks) [Computer Graphics (R) November 2001]

6.0) Summary

In this lecture we have taken our first look at how the graphics system converts the graphical primitives from its geometrical definition into their component pixels. The elements discussed further will give you an idea on how to draw a straight line from a point to another. We have discussed about the DDA incremental scan conversion method for constructing a line by plotting pixel positions along a straight line.

7) Exercise questions

7.0) The endpoints of a line are given as (0, 0) and (6, 18). Compute each value of y as x steps from 0 to 6 and plot the results

7.1) Use pseudo-code to describe the steps that are required to plot a line whose slope is between 45° and -45° (i.e., |m|>1) using the slope intercept equation.

7.2) Use pseudo-code to describe the DDA algorithm for scan converting a line whose slope is between -45° and 45° (i.e., |m|≤1).

7.3) Indicate which raster locations would be chosen by DDA algorithm when scan converting a line from pixel coordinate (1, 1) to a pixel coordinate (8, 5)

7.4) What are the steps required to plot a dashed line?

8.0) Programming / Computational assignments

Implement the DDA line drawing algorithm.

Lecture #6

1) Synopsis: Bresenham’s line drawing Algorithm

2) Target: At the completion of this lecture you should be able to answer questions like

(a) How is a line drawn using Bresenham’s algorithm?
You should also be able to explain the mathematical and algorithmic aspects of the Bresenham’s scan conversion method.

3) Introduction
In lecture <1> we discussed about the line drawing algorithm. You know that DDA algorithm is an incremental scan conversion method which performs calculations at each step using the results from the preceding step. Here we are going to discover an accurate and efficient raster line generating algorithm, the **Bresenham's line-drawing algorithm**. This algorithm was developed by Jack E. Bresenham in 1962 at IBM. As stated above, in this lecture, I will explain how to draw lines using the **Bresenham's line-drawing algorithm**. And then show you the complete line drawing function. Before we begin on this topic, a revision of the concepts developed earlier like scan conversion methods and rasterization may be helpful. Once we finish this aspect, we will proceed towards exposition of items listed in the synopsis. In particular, we will emphasize the following
(a) The working of Bresenham’s algorithm.
(b) Implementation of the Bresenham’s algorithm.

4) Revision / Prerequisites
Please refer to pages 86 to 94 of your textbook.

5.1) Concept #1: The working of Bresenham’s algorithm
The following is an explanation of how the Bresenham's line-drawing algorithm works, rather than exact implementation.
Let's take a look at this image. One thing to note here is that it is impossible to draw the true line that we want because of the pixel spacing. Putting it in other words, there's no enough precision for drawing true lines on a PC monitor especially when dealing with low resolutions. The Bresenham's line-drawing algorithm is based on drawing an approximation of the true line. The true line is indicated in bright color, and its approximation is indicated in black pixels.

In this example the starting point of the line is located exactly at 0, 0 and the ending point of the line is located exactly at 9, 6. Now let discuss the way in which this algorithm works. First it decides which axis is the major axis and which is the minor axis. The major axis is longer than the minor axis. On this picture illustrated above the major axis is the X axis. Each iteration progresses the current value of the major axis (starting from the original position), by exactly one pixel. Then it decides which pixel on the minor axis is appropriate for the current pixel of the major axis. Now how can you approximate the right pixel on the minor axis that matches the pixel on the major axis? - That’s what Bresenham's line-drawing algorithm is all about. And it does so by checking which pixel's center is closer to the true line.

Now you take a closer look at the picture. The center of each pixel is marked with a dot. The algorithm takes the coordinates of that dot and compares it to the true line. If the span from the center of the pixel to the true line is less or equal to 0.5, the pixel is drawn at that location. That span is more generally known as the error term.

You might think of using floating variables but you can see that the whole algorithm is done in straight integer math with no multiplication or division in the main loops(no fixed point math either). Now how is it possible? Basically, during each iteration through the main drawing loop the error term is tossed around to identify the right pixel as close as possible to the true line. Let's consider these two deltas between the length and height of the line: dx = x1 - x0; dy = y1 - y0; This is a matter of precision and since we're working with integers you will need to scale the deltas by 2 generating two new values: dx2 = dx*2; dy2 = dy*2; These are the values that will be used to change the error term. Why do you scale the deltas? That’s because the error term must be first initialized to 0.5 and that cannot be done using an integer. Finally the scaled values must be subtracted by either dx or dy (the original, non-scaled delta values) depending on what the major axis is (either x or y).

5.2) Concept #2: The implementation of Bresenham’s algorithm

The function given below handles all lines and implements the complete Bresenham's algorithm.
function line(x0, x1, y0, y1)
    boolean steep := abs(y1 - y0) > abs(x1 - x0)
    if steep then
        swap(x0, y0)
        swap(x1, y1)
    if x0 > x1 then
        swap(x0, x1)
        swap(y0, y1)
    int deltax := x1 - x0
    int deltay := abs(y1 - y0)
    real error := 0
    real deltaerr := deltay / deltax
    int y := y0
    if y0 < y1 then ystep := 1 else ystep := -1
    for x from x0 to x1
        if steep then plot(y, x) else plot(x, y)
        error := error + deltaerr
        if error ≥ 0.5
            y := y + ystep
        error := error - 1.0

Note:-To draw lines with a steeper slope, we take advantage of the fact that a steep line can be reflected across the line y=x to obtain a line with a small slope. The effect is to switch the x and y variables throughout, including switching the parameters to plot.

5.8) University Questions

(s) Explain Bresenham’s line drawing algorithm (10 marks) [Computer Graphics (R) November 2001]

6.0) Summary

In this lecture we discussed about an interesting algorithm developed by Bresenhan. This algorithm is designed so that each iteration changes one of the coordinate values by ±1. The other coordinate may or may not change depending on the value of the error term maintained by the algorithm. We have discussed that this
error term records the distance, measured perpendicular to the axis of greatest movement, between the exact path of the line and the actual dots generated.

7) Exercise questions

7.0) Write a program to implement Bresenham's algorithm (as given in class) to draw a straight line segment in any direction with any slope. Your program should prompt the user to enter the beginning and ending points of the line segment and display the line on the console. Your program should also output a list of the coordinates of the pixels used in drawing the line, in the order in which they where turned black.

7.1) Draw a line from (0, 0) to (11, 3) using the Bresenham’s line algorithm and show the values of \( dx \), \( dy \), \( 2dy \), \( 2dy - 2dx \) and \( p_0 \). Implement it with the aid of MATLAB.

7.2) How would you generalize the Bresenham line drawing algorithm to draw circles rather than lines? Assume that as input you get the integer coordinates of the circle's center and an integer radius.

8.0) Programming / Computational assignments

Implement an interactive demonstration of Bresenham's midpoint algorithm for scan-converting line segments. The demo should have the following behavior:

- The pixels are displayed very large, drawn as a rectangular grid of white or black squares on a gray background. The size of the pixel grid is fixed at 10 x 10.
- The user will click the mouse on the starting pixel, drag the pointer to the end pixel, and release the mouse button. This causes two components to be displayed:
  a. The line joining the two endpoints.
  b. The pixels set by Bresenham's algorithm.

For example: in the following image, the user has clicked near pixel (1,3) and dragged to near pixel (8,6).
Notice that the endpoints of the line chosen by the user do NOT lie on pixel centers. If you want to, you can force the drawn line to start and end on pixel centers.

**Lecture #7**

1) **Synopsis:** 2D Transformations

2) **Target:** At the completion of this lecture you should be able to answer questions like
   a) What is transformation?
   b) What are the different types of transformations?
   c) How do you represent them in the matrix form?

You should also be able to understand the theory and practice of 2D Geometry and 2D transformations

3) **Introduction**

   In lecture #1 we have discussed that the end product of computer graphics is a picture. In its broadest sense you know that a picture means any collection of lines, points, text, etc. that can be displayed on a graphics device. Also, a graphics system should allow the programmer to define pictures that include a variety of transformations. So let’s discuss about two dimensional transformations. As stated above, in this lecture, we plan to explore the transformation principles, the different forms of two dimensional transformations and their matrix representations. Before we begin on this topic, a revision of the concepts developed earlier in your Engineering Mathematics classes like vector calculus, trigonometry and matrices may be helpful. Once we
finish this aspect, we will proceed towards exposition of items listed in the synopsis. In particular, we will emphasize the following

(a) Transformation Principles
(b) The coordinate systems
(c) Different forms of transformation
   i. Translation
   ii. Rotation
   iii. Scaling
(d) The Matrix representations
(e) The concatenation process

4) Revision / Prerequisites
Refer to pages 184 to 190 of your text book.

5.1) Concept #1: Transformation Principles

You can say that 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 (e.g. the type of perspective that is used). Putting it in other words a transformation is a function that maps every position \((x, y)\) into a new position \((x', y')\). Instead of applying the transformation to every point in every line that makes up the object, we simply apply the function to the objects vertices and then draw new lines between the resulting new endpoints.

Two aspects of the formulation of transformation should be emphasized:

- A transformation is a single mathematical entity and such can be denoted by a single name or symbol.
- Two transformations can be combined, or concatenated, to yield a single transformation with the same effect as the sequential application of the original two. Thus transformation A might be a translation and transformation B a scaling.
- The concatenation property allows us to determine a transformation \(C = AB\) whose effect is to translate and then scale.
Each of the transformations is used to generate a new point \((x', y')\) from the coordinates of a point \((x, y)\) in the original picture description. If the original definition includes a line, it suffices to apply the transformation to the endpoints of the line and display the line between the two transformed endpoints.

5.2) Concept #2: World and Modeling Coordinates

Before going into details let us discuss on 3D coordinate systems. As you all know in 3D graphics, one uses a coordinate system in order to represent points in space by a series of numbers. Usually Cartesian coordinates are used, as these are the easiest to work with. It defines three straight fixed perpendicular lines called axes (usually referred to as X, Y, and Z). A point in 3D can be described by three numbers, which indicate the distance of the point from each axis. This is represented by an ordered triple: \((x,y,z)\). However, there is not just one Cartesian coordinate system. There are many, which have their origins in different places and their axes aligned with different things.

**World Coordinates**

The world coordinate system forms the "base" on which you can think of all other coordinates being defined. Usually, its origin is at the center of your game world, and its axes might be aligned with directions like north/south, east/west, and up/down. (For example, it is very common for the X axis to be east, the Y axis to be north, and the Z axis to be up. Another way is for X to be east, Y to be up, and Z to be south.) This is the coordinate system in which most of your level geometry - such as rooms, hallways, and landscapes - will probably be defined.

**Modeling coordinates**

Modeling coordinates is the one in all drawing primitives do their drawing. The user can select the position and orientation of the modeling space with regard to the world space by means of translations, rotations, scales, or generalized transformations. The relation between modeling coordinates and world coordinates is determined by the modeling matrix. Modeling coordinates are a useful conceptual device when drawing complex or repetitive scenes. For instance, a paper clip can be defined once in modeling coordinates, and then drawn hundreds of times by moving the modeling coordinate around in world space.

5.3) Concept #3: Different types of Transformation

There are three basic 2D Transformation functions:
Now let’s consider the figures given below. Here we are trying to transform a picture in the world coordinate into an object in the modeling coordinates.

Modeling Coordinates

![Modeling Coordinates Diagram]

World Coordinates

**Figure 1**

Modeling Coordinates
Let’s look at this in detail.

World Coordinates

Figure 2

Modeling Coordinates

Figure 1

Initial location at (0, 0) with x- and y-axes aligned

Modeling Coordinates

Figure 3

Scale .3, .3
World Coordinates

Figure 4: Scaling Transformation is done

Modeling Coordinates

Figure 5: Rotating the object by 90°
5.4) Concept #4: Translation

Translations with 2 dimensional points are not very difficult. All we need to do is determine how much we want to move the object in the x and y direction, and add those variables to the point's x and y respectively.

Translation means a shift by $T_x$ in the x direction and $T_y$ in the y direction is

\[
x' = x + T_x \\
y' = y + T_y
\]

$T_x$ and $T_y$ are translation amounts in the x direction in the y direction respectively. For example let us see the figure given below. An object at position (4,5) and (7,5) has been shifted to a new position (7,1) and (10,1).
5.5) Concept #5: Scaling

A scaling by $S_x$ in the $x$ direction and $S_y$ in the $y$ directions about the origin is

$$x' = S_x x$$
$$y' = S_y y$$

If $S_x$ and $S_y$ are not equal this results in a stretching along the axis of the larger scale factor. To scale about a particular point, first translate to the origin, scale, and translate back to the original position. For example, to scale about the point $(x_0, y_0)$

$$x' = x_0 + S_x (x - x_0)$$
$$y' = y_0 + S_y (y - y_0)$$

In other words scaling a coordinate means multiplying each of its components by a scalar. Uniform scaling means this scalar is the same for all components. For example, we have an object initially at $(4, 5)$ and $(7, 5)$. A scaling Factor of $\frac{1}{4}$ is applied to this object. Look at the figure given below.
Note that the house changes position, since the scaling is about the origin. If scaling were uniform, the amount of scaling in each dimension would be equal.

5.6) Concept #6: Rotation

Rotation about the origin by an angle $A$ in a clockwise direction is

$$
x' = x \cos(A) + y \sin(A) $$
$$
y' = y \cos(A) - x \sin(A) $$

To rotate about a particular point apply the same technique as described for scaling, translate the coordinate system to the origin, rotate, and the translate back. Putting it in other words to rotate about another point, first translate the point to the origin, rotate, and then translate back.

The following transforms perform rotation:

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

The angle of rotation is about the origin. Positive angles are measured counterclockwise. You should be able to reformulate the rotational transformations for negative angles also.

Use the identities:
\[ \cos(-\theta) = \cos \theta \]
\[ \sin(-\theta) = -\sin \theta \]

**Proof:**

Consider:

By simple trigonometry:

\[ x = r \cos \phi \] and \[ y = r \cos \phi \]

\[ x' = r \cos(\theta + \phi) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \]

\[ y' = r \sin(\theta + \phi) = r \cos \phi \sin \theta + r \sin \phi \cos \theta \]

By substitution

\[ x' = x \cos \theta - y \sin \theta \]

\[ y' = x \sin \theta + y \cos \theta \]

**5.7) Concept #7: Matrix representation of Transformations**
The matrix representation of different transformations are given below

**Translation**

\[
P = \begin{bmatrix} x \\ y \end{bmatrix}
\]

\[
P' = \begin{bmatrix} x' \\ y' \end{bmatrix}
\]

and \( T = \begin{bmatrix} dx \\ dy \end{bmatrix} \)

so \( P' = P + T \)

**Scaling**

\[x' = s_x x\]

\[y' = s_y y\]

which has a matrix form:

\[
P' = \begin{bmatrix} x' \\ y' \end{bmatrix} S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} P = \begin{bmatrix} x \\ y \end{bmatrix}
\]

**Rotation**

\[
P' = R P \text{ where}
\]

\[
P' = \begin{bmatrix} x' \\ y' \end{bmatrix}
\]

\[
P = \begin{bmatrix} x \\ y \end{bmatrix}
\]

\[
R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}
\]

**5.8) Concept #8: Concatenation**

Now we have discussed about different types of transformations. Now is it possible to combine one transformation into another? The answer is yes. We can combine a sequence of transformation into one transformation. This process is called concatenation. Look again at transformations in terms of vectors.

Scaling: \( x' = x * S_x, y' = y * S_y \), which can be written as:

\[
(x' \ y') = (x \ y) * (S_x \ 0)
\]

\[
(0 \ Sy)
\]
Rotation: \( x' = x \cos q - y \sin q, y' = y \cos q + x \sin q \), which can be written as:

\[
\begin{pmatrix}
    x' \\
    y'
\end{pmatrix} =
\begin{pmatrix}
    x & y
\end{pmatrix} \cdot 
\begin{pmatrix}
    \cos q & \sin q \\
    -\sin q & \cos q
\end{pmatrix}
\]

So if we wanted to scale and then rotate the object we could do the following:

\[
\begin{pmatrix}
    x' \\
    y'
\end{pmatrix} =
\begin{pmatrix}
    x & y
\end{pmatrix} \cdot 
\begin{pmatrix}
    S_x & 0 \\
    0 & S_y
\end{pmatrix}
\]

\[
\begin{pmatrix}
    x'' \\
    y''
\end{pmatrix} =
\begin{pmatrix}
    x' & y'
\end{pmatrix} \cdot 
\begin{pmatrix}
    \cos q & \sin q \\
    -\sin q & \cos q
\end{pmatrix}
\]

But this is the same as:

\[
\begin{pmatrix}
    x'' \\
    y''
\end{pmatrix} =
\begin{pmatrix}
    x & y
\end{pmatrix} \cdot 
\begin{pmatrix}
    S_x & 0 \\
    0 & S_y
\end{pmatrix} \cdot 
\begin{pmatrix}
    \cos q & \sin q \\
    -\sin q & \cos q
\end{pmatrix}
\]

\[
= 
\begin{pmatrix}
    x & y
\end{pmatrix} \cdot 
\begin{pmatrix}
    S_x \cdot \cos q & S_x \cdot \sin q \\
    -S_y \cdot \sin q & S_y \cdot \cos q
\end{pmatrix}
\]

Hence, we can concatenate the Scaling and Rotation matrices, and then multiply the old points by resultant matrix.

**5.9) University Questions**

(a) Explain in detail the principle of 2D transformations (12 marks) [F 3794 Computer Graphics (R,T) November 2005]

(b) Prove that two successive rotations are additive(4 marks) [F 3879 Computer Graphics (R) November 2005]

(c) Prove that rotation and uniform scaling is a commutative pair of operations(4 marks) [F 3879 Computer Graphics (R) November 2005]

**6.0) Summary**
In this lecture we have discussed about modeling transformation principles. We have understood that to change the size of the object we can use scaling transformation. To move the object and to rotate the object we are familiarized with the translation and rotation transformations. We have also discussed how these transformations are represented in the matrix format. We concluded by mentioning how these transformations can be concatenated into one transformation.

7) Exercise questions

7.0) Implement translation of a line, triangle and rectangle.
7.1) Implement rotation of a line, triangle and rectangle.
7.2) Implement scaling of a line, triangle and rectangle.

8.0) Programming / Computational assignments

Design a 2D window, where there is a regular grid, with two coordinate systems shown. Each coordinate system is drawn with two arrows representing the basis vectors. The user can drag the origin of each system, or the tips of the vectors. Movements should snap to grid positions, if possible. The user should also be able to "lock" the angle between the x and y vectors, and rotates the two rigidly about the origin. The user can also choose a point whose coordinates will be calculated.

On a separate text window, display the coordinate-change matrix, as well as the translation, rotation, scaling matrices that make up the transformation.

Lecture # 8

1) Synopsis: Bresenham’s line drawing Algorithm

2) Target: At the completion of this lecture you should be able to answer questions like
   (b) How do you scan convert a circle?
   (c) What do you mean by the symmetrical property of a circle?
   (d) How is a circle drawn using Bresenham’s algorithm?

You should also be able to explain the mathematical and algorithmic aspects of the Bresenham’s scan conversion method.

3) Introduction
In lecture 5 and 6 we discussed about how to scan convert a line. We had also gone through two important algorithms used for plotting lines. As stated above, in this lecture, we plan to explore how to scan convert a circle. Here we shall try to understand an algorithm used to plot the circle. Before we begin on this topic, a revision of the concepts developed earlier in lecture 5 and 6 may be helpful. Hence I advise you to have a look at it before starting off. Once you finish this aspect, we will proceed towards exposition of items listed in the synopsis.

In particular, we will emphasize the following

a) Scan converting a circle
b) Bresenham’s method of scan conversion
c) Bresenham’s algorithm for circles

5.1) Concept #1: Scan converting a circle

Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.

We know that there are 360 degrees in a circle. First we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated. Next we see that it's also symmetrical about the y axis, so now we only need to calculate the first 90 degrees. Finally we see that the circle is also symmetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees

We only need to calculate the values on the border of the circle in the first octant. The other values may be determined by symmetry. Assume a circle of radius r with center at (0,0).
Procedure Circle_Points(x, y: Integer);
   Begin
      Plot(x, y);
      Plot(y, x);
      Plot(y, -x);
      Plot(x, -y);
      Plot(-x, -y);
      Plot(-y, -x);
      Plot(-y, x);
      Plot(-x, y)
   End;

5.2) Concept #2: The Bresenham's Method of scan conversion

Bresenham's circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x, y) it calculates, we draw a pixel in each of the eight octants of the circle. This is done till when the value of the y coordinate equals the x coordinate.

PutPixel (CenterX + X, Center Y + Y)
PutPixel (CenterX + X, Center Y - Y)
5.3) Concept #3 : The Bresenham’s Circle-Drawing Algorithm

The algorithm is given as follows:

Given a radius for the circle we perform this initialization:

\[
\begin{align*}
    d &:= 3 - (2 \times \text{RADIUS}) \\
    x &:= 0 \\
    y &:= \text{RADIUS}
\end{align*}
\]

Now for each pixel we do the following operations:

\[
\begin{align*}
    \text{Draw the 8 circle pixels} \\
    \text{if } d < 0 \text{ then} \\
    \quad d &:= d + (4 \times x) + 6 \\
    \text{else} \\
    \quad \text{begin} \\
    \quad \quad d &:= d + 4 \times (x - y) + 10 \\
    \quad \quad y &:= y - 1; \\
    \quad \text{end;}
\end{align*}
\]

6.0) Summary

In this lecture we have discussed about scan converting circles and an efficient method to plot circles. We have looked into the Bresenham’s algorithm to draw circles. By using C language we can easily implement this algorithm and plot the circles with different centers.

7) Exercise questions
7.0) Implement the Bresenham’s circle drawing algorithm.

**Mid point circle algorithm**

In midpoint circle algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step. For a given radius \( r \) and screen center position \((x_c, y_c)\), this algorithm first calculate pixel positions around a circle path centered at the coordinate origin \((0, 0)\). Then each calculated position \((x, y)\) is moved to its proper position by adding \( x_c \) to \( x \) and \( y_c \) to \( y \). Along the circle section from \( x = 0 \) to \( x = y \) in the first quadrant, the slope of the curve varies from 0 to -1. Therefore, we can take steps in the positive \( x \) direction over this octant and use a decision parameter to determine which of the two possible \( y \) positions is closer to the circle path at each step. Positions of the other seven octants are then obtained by symmetry.

\[
f_{\text{circle}}(x, y) = x^2 + y^2 - r^2
\]

Any point on the boundary of the circle with radius \( r \) satisfies the equation \( f_{\text{circle}}(x, y) = 0 \). If the point is in the interior of the circle, the circle function is negative. If the point is outside the circle, the circle function is positive. This test is performed for the mid positions between pixels near the circle path at each sampling step.

Assuming we have plotted the pixel at \((X_k, Y_k)\), we next need to determine whether the pixel at position \((X_k + 1, Y_k - 1)\) is closer to the circle. For that we calculate the circle function at the midpoint between these points.

\[
P_k = f_{\text{circle}}(X_k + 1, Y_k - \frac{1}{2})
\]

If \( P_k < 0 \), this midpoint is inside the circle and the pixel on the scan line \( Y_k \) is closer to the circle boundary, and we select the pixel on the scan line \( Y_k - 1 \). Successive parameters are obtained using incremental calculations. The initial decision parameter is obtained by evaluating the circle function at the staring position \((X_0, Y_0) = (0, r)\).

\[
= f_{\text{circle}}(1, r - \frac{1}{2})
\]

\[
= 1 + (r - \frac{1}{2})^2 - r^2
\]
The algorithm is given as follows:

Initialize x=0, y = r and p = 1-r
For x<=y do the following
   Plot the pixel (x,y)
   If (p <0)
      p = p+2x + 3
   Else
      p=p + 2(x-y) + 5
   Decrement the value of y

3D OBJECT REPRESENTATION
One of the major concepts in computer graphics is modeling of objects. Modeling of objects means describing the objects in terms of their geometric properties (like shape and size) and how they interact with light (reflect, transmit). A graphics system usually uses a set of primitives or geometric forms to model variety of objects. Geometric forms that are often used as primitives include points, lines, polylines, polygons and polyhedra. More complex geometric form includes curves, curved surface patches, and quadratic surfaces.

SIMPLE GEOMETRIC FORMS

Points and lines
The basic building blocks of computer graphics can be termed as points and lines. How do we specify a point? We do it by specifying its coordinate in three or two dimensional space. For example, a point can be represented as P(x, y, z). How do we specify a line segment? It can be done by specifying the two end points P1(x1, y1, z1) and P2(x2, y2, z2).

Polylines
What is a polyline? It’s a chain of connected line segments. It is specified by giving the vertices or nodes that defines the line segment. For example, a polyline can be defined as P0, P1…, Pn. The first vertex of the
Polyline is called the starting point or the initial point and the last vertex is called the terminal point or the final point.

**Polygons**

How do you specify a polygon? A polygon can be treated as a closed polyline, that is, a polyline where the initial and the terminal vertices coincide. It is specified by its vertex list as P0, P1, …, Pn, P0. The edges of the polygon are specified as line segments P0P1, P1P2, …, PnP0. What is a planar polygon? A polygon in which all vertices lie on the same plane can be called as a planar polygon.

**WIRE FRAME MODELS**

A wire frame model consists of edges, vertices, and polygons. In a wire frame the vertices are connected by edges, and the polygons are sequences of vertices and edges. The edges may be curves or it may be straight line segments. If the edges are straight line segments then the wire frame model is called a polygonal net or polygonal mesh.

The wire frame models are used in engineering applications. The advantage of wire frame model is that they are easy to construct. They are easy to clip and manipulate if they are composed of straight lines. But they cannot be used in building up realistic models. For modeling highly curved objects we will be forced to use very large number of polygons to achieve the illusions of roundness and smoothness.

**Polyhedron**

A polyhedron is a closed polygonal net in which each polygon is planar. A closed polygonal net means it encloses a definite volume. The polygons are called the faces of the polyhedron. In modeling, polyhedrons are quite often treated as solid objects (block).

**POLYGON SURFACES**

The boundary of a three dimensional graphics object can be represented by a set of polygon surfaces that enclose the object interior. The object descriptions are stored as sets of surface polygons. Therefore all polygon surfaces can be easily described with linear equations. A polygon representation of a polygon precisely defines the surface features of the object.
Polygon Tables

How do you specify a polygon surface? This can be done by using a set of vertex coordinates and associated attributes parameters. We use polygon tables to store such information. These tables are used in processing, display and manipulation of objects in a scene. The polygon tables can be categorized into two: geometric tables and attribute tables. The geometric table contains vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces. The attribute table contains information about the degree of transparency of the object and its surface reflectivity and texture characteristics.

Now let’s see how the geometric data is stored. This can be done by creating three lists:

1. Vertex table
2. Edge table
3. Polygon table

The coordinate value for each vertex in the object is stored in the vertex table. The edge table contains the list of vertices that forms each edge. The polygon table contains the list of edges that forms the polygon surfaces.

Now how are these list linked inter related? There is a back pointer pointing from the edge table to the vertex table. Similarly there is another pointer that points back to the edge table from the polygon table.

Now how do you construct a polyhedron from these three tables? At first, the polygon surface table will be searched to identify the edges that compose the polygon. From the polygon table the information about the edges are obtained. Now we need to find the details of the edges. The details of the edges are stored in the edge table. Using the pointer from the polygon table we move to the edge table to locate the vertices that compose the edges of the polygon. From the edge table we will get the details of the vertices that compose each edge. But now we need to get the information about each vertex. These details are stored in the vertex table. By using the pointer from the edge table to the vertex table we can get the details about the x, y, and z coordinates that compose each vertex.

We can always add extra information to the data tables for faster information extraction. For example we could expand the edge table to include forward pointers into the polygon table so that common edges between the polygons could be identified more rapidly. Similarly, the vertex table could be expanded so that the vertices are cross reference to corresponding edges. We can also store additional information in the data tables like the slope for each edge and the coordinate extent for each polygon.
We can eliminate the edge table by letting the polygon table reference the vertices directly, but we can run into problems, such as drawing some edges twice, because we don't realize that we have visited the same set of points before, in a different polygon. We could go even further and eliminate the vertex table by listing all the coordinates explicitly in the polygon table, but this wastes space because the same points appear in the polygon table several times.

Let’s consider the example of a polyhedron with two surfaces. The three tables that describes the polygon is as shown in the figure:

Now what about the spatial orientation of the individual surface of the object? Often in the graphics pipeline, we need to know the orientation of an object. This information is obtained from the equation of the plane that describes the polygon.

The equation of the plane can be expressed as $Ax + By + Cz = 0$ where $(x, y, z)$ is any point on the plane and coefficients $A$, $B$, $C$, and $D$ are constants defining the spatial properties of the plane. Here the value of $ABCD$ can be obtained by solving the set of three plane equations using the coordinate for three non collinear points in the plane. For this purpose let us select three consecutive polygon vertices $(x_1, y_1, z_1)$, $(x_2, y_2, z_2)$, $(x_3, y_3, z_3)$ and solve the simultaneous linear plane equations.
(A/D)x_k + (B/D)y_k + (C/D)z_k = -1

Then we can use the equation to determine whether a point is on the inside or outside of the plane formed by this polygon:

Ax + By + Cz + D < 0 = the point (x, y, z) is inside the surface

Ax + By + Cz + D > 0 = the point (x, y, z) is outside the surface

**Normal to the plane**

The coefficients A, B, and C can also be used to determine a vector normal to the plane of the polygon. This vector, called the surface normal, is given simply by:

N = (A, B, C).

If we specify the vertices of a polygon counterclockwise when viewing the outer side, in a right-handed coordinate system, the surface normal N will point from inside to outside. You can verify this from an alternate definition for N, based on three vertices:

N = (V2 - V1) x (V3 - V1) = (A, B, C)

**Polygon Meshes**

Objects can be modeled by several polygon functions. But when object surfaces are to be tiled, it is more convenient to specify the surface faces with mesh function. One type of the polygon mesh is the triangle strip. The function produces n – 2 triangles given the coordinate for n vertices. A triangle list will produce only n/3 triangles for a polygon with n vertices.
Another similar function is the quadrilateral mesh, which generates a mesh of \((n-1)\) by \((m-1)\) quadrilaterals, given the coordinates for an \(n\) by \(m\) array of vertices. A quadrilateral mesh containing 12 quadrilaterals constructed from a 5 by 4 input vertex array is shown in the figure below.

![Quadrilateral Mesh](image)

**CURVES AND SURFACES**

You know that objects with complex shapes occur frequently in our 3D world. Therefore special techniques have to be developed to model these objects so that we get realistic images. These objects can be approximated as plane faced polyhedra. For example the polyhedral approximation of a coffee cup may contain 1000 faces. In this case it will be difficult to generate and modify the shape of the cup. We need a more direct representation of shapes which is tractable both to the computer and to the person trying to make modifications on the shape.

**Representation of curves**

Let’s discuss some of the crucial properties that our representation must have. These properties are important in designing the curves.

1. Control Points
2. Multiple value
3. Axis independence
4. Global and local control
5. Variation Diminishing property
6. Versatility
7. Order of Continuity

**Control Points**
How do you give shape to the curve? How do you control its shape? The answer to these questions is control points. Control points are a set of coordinate positions that indicates the general shape of the curve. It is by the control points the shape of the curve is controlled. The control points influence the shape of the curve. It is through these control points the curve pass. Examples are given below:

The line connecting the control points in order is called the control graph, or control polygon, or even characteristic polygon. If the curve passes exactly through each of the control points then the curve is said to interpolate the set of control points. If the curve doesn’t pass necessarily through the control points then it is said to approximate the set of control points. Examples are given below:
Multiple values: The curve can have multiple values. It is not a graph that is plotted by using a single values function. A curve can be multivalued with respect to all coordinate systems.

Axis independence: The shape of the curve must not change when the control points are measured in a different coordinate system. For example if the control points are rotated $60^\circ$ the curve should rotate $60^\circ$ but the shape of the curve should not change.

Global and local control: We know that control points are used to control the shape of the curve. When we try to manipulate or change the position of a control point the shape of the curve changes. Manipulation of the control point can bring in two effects on the curve. The shape of the curve gets affected entirely or the shape of the curve changes only in the region near to the control point. If the shape of the curve changes entirely then it is said to have global control. If the shape changes only in the region near the control point, it is said to have local control.

Variation-diminishing property: Certain curves have a tendency to amplify small irregularities in the shape outlined by the control points. But there are certain other curves that always smooth the designer’s control points diminishing the irregularities in the shape. A curve that oscillates about its control points is usually undesirable. But there will not be such oscillations in curves that possess variation diminishing property. The curve will always tend to smooth out the sequence of control points.

Versatility: The shapes of the curves formed from the control points should not be limited. It should not lack versatility. How can we bring in versatility in curve designs? This can be done by adding or removing control
points from the framework. For example, a straight line can be formed from two control points. This straight line can be changed into a curve by simply adding one more control point. This third control point can create a large number of additional shapes by just changing its location.

**Order of continuity:** Using simple curves simple object shapes can be modeled but for complex shapes we need to depend on complex curves. Now you know how a curve is formed from the control points. Now let’s discuss how a complex curve is formed? A complex curve can be formed by joining simple curves together. The shape that cannot be described by a simple curve can often be described by several curves joined together.

When the curves are joined the order of continuity has to be maintained. There are three orders of continuity defined. They are:

1. Zero Order
2. First order
3. Second order

Zero order continuity means simply that two curves meet. First order continuity requires the curve to be tangent at the point of intersection. Second order continuity requires that the curvatures be the same.

**BEZIER METHODS**

This spline approximation method was developed by P. Bezier, a French engineer, for use in the design of Renault automobile bodies.

**Bezier Curves**

In general a Bezier curve section can be fitted to any number of control points. The degree of the polynomial is determined by two factors. The number of control points and their relative position are the factors that contribute to the degree of the Bezier curve. The degree of the Bezier curve is one less than the number of control points used. If there are \( n + 1 \) control points then the degree of the Bezier curve will be \( n \). For example, three points will generate a parabola and four control points will generate a cubic curve.

As we know the main form used to model curves and surfaces is the parametric function or vector-valued function. A point on the curve is represented as a vector:
Beziers defines the curve $P(u)$ in terms of the location of $n+1$ control points $p_i$

$P(u) = [x(u) \quad y(u) \quad z(u)]$

where $B_{i,n}(u)$ is a blending function

$$B_{i,n}(u) = C(n,i)u^i(1-u)^{n-i}$$

And $C(n,i)$ is the binomial coefficient, $C(n,i) = n!/(i!(n-i)!)$

Here $p_i$ is the control point where $p_i$ can be defined as $[x_i \quad y_i \quad z_i]$.

$X(u) = \sum X_k B_{E_k,n}(u) \quad 0 < k < n$

$Y(u) = \sum Y_k B_{E_k,n}(u) \quad 0 < k < n$

$Z(u) = \sum Z_k B_{E_k,n}(u) \quad 0 < k < n$

Here there are $n+1$ control points as $p_i$ ranges from 0 to $n$. These control points along with the blending function gives the shape to the curve. The blending function defined influences the control points i.e. it actually blends the control point $p_i$. The blending functions are the key to the behaviour of Bezier curves.

Here let us take an example of a Bezier curve with four control points. So $p_i$ ranges from 0 to 3. Therefore the Bezier parametric function can be written as

$$P(u) = p_0 B_{0,3} + p_1 B_{1,3} + p_2 B_{2,3} + p_3 B_{3,3}$$

Here the control points $p_i$'s are influenced by their respective blending functions. The graph

Algorithm:

a) Compute $C(n,k)$, $k$ ranging from 0 to $n$ where $n =$Control points-1
b) Repeat steps 3,4,5,6 and 7 with $k$ ranging from 1 to $n$

c) blend=$nCk \times u^k \times (1-u)^{n-k}$
d) \( x = x + x \cdot \text{blend} \)
e) \( y = y + y \cdot \text{blend} \)
f) \( z = z + z \cdot \text{blend} \)
g) Join the points
h) End

Properties:

a) Curve always passes through first and last control points
   i.e. \( P(0) = P_0 \)
   \( P(1) = P_n \)

b) Values of the parametric first derivatives at the endpoints can be calculated from control point coordinates as
   \( P'(0) = -nP_0 + nP_1 \)
   \( P'(1) = -nP_{n-1} + nP_n \)

c) Slope at the beginning of curve is along the line joining the first two control points and the slope at the end of the curve is along the line joining the last two endpoints.

d) Curve lies within a convex hull (convex polygon boundary) of the control points. The curve is contained by the polygon.

Applications:

a) Painting and drawing packages-for drawing curves
b) CAD systems- for drawing smooth curves
c) Curve Fitting- to fit a curve along the layout of an object
d) Animation- to transform from one shape to another or to simulate a motion
e) Accelerating 3D Graphics- to simulate transformation of an object
B-Spline Curves

a) B-spline functions of order m are piecewise polynomials of degree m-1. They are m-2 smooth.
b) The B-spline functions form a basis; every spline function can be written as a linear combination of B-splines.
c) Nk,m(t) has support [tk, t k+m].
d) A B-spline curve lies within the convex hull of m consecutive control points.
e) Knots can appear multiple times within the knot vector. A knot with multiplicity m will cause the resulting curve to interpolate the corresponding control point.
f) Affine invariance.

ANIMATION

Computer animation generally refers to any time sequence of visual changes in a scene. In addition to changing object position with translation or rotations, computer-generated animations could display time variations in object size, color, transparency or surface texture.

Design of an animation sequence

In general an animation sequence is designed with the following steps

a) Storyboard layout
b) Object definitions
c) Key-frame specifications
d) Generation of in between frames
The storyboard is an outline of the action. It defines motion sequences as a set of basic events that are to take place. Depending on the type of animation to be produced, the storyboard could consist of a set of rough sketches or it could be a list of basic ideas for the motion.

An object definition is given for each participant in the action. Objects can be defined in terms of basic shapes, such as polygon or splines. In addition, associated sets of movements are also specified along with the shape.

A key frame is a detailed drawing of the scene at a certain time in the animation sequence. Within each key frame each object is positioned according to the time for that frame. In-betweens are intermediate frames between the key frames. The number of In-betweens needed is determined by the media to be used to display animation. Film requires 24 frames per second, and graphics terminal are refreshed at the rate of 30 to 60 frames per second. Typically time intervals for the motion are set up so that there are from three to five In-betweens for each pair of key frames.

When someone creates a 3D animation on a computer, they usually don't specify the exact position of any given object on every single frame. They create key frames. Key frames are important frames during which an object changes its size, direction, shape or other properties. The computer then figures out all the in between frames and saves an extreme amount of time for the animator.

Two key frames drawn by the user

In between frames generated by the computer
On raster scan systems we can generate real time animation in limited applications using raster operations. A simple method for translation in the xy plane is to transfer a rectangular blocks of pixels through arbitrary angles using anti-aliasing procedures. To rotate a block of pixels we need to determine the percent of area coverage for those pixels that overlap the rotated block. Sequences of raster operations can be executed to produce real time animation of either two-dimensional or three-dimensional objects, as long as we restrict the animation to motions in the projection plane. Then no viewing or visible surface algorithms need be invoked.

**Morphing**

Transformation of object shapes from one form to another is called morphing. We generate set of in-betweens from the specification of two or more key frames. Given the animation paths we can interpolate the positions of individual objects between any two times or key frames. With complex object transformations the shapes of the object may change over time. If all surfaces are described with polygon meshes then the number of edges per polygon can change from one frame to the next. Thus the total number of line segments can be different in different frames.

Transformation of object shapes from one form to another is called morphing. Morphing methods can be applied to any motion or transition involving a change of shape.

Given two key frames for an object transformation we first adjust the object specification in one if the frames so that the number of polygon edges (or the number of vertices) is the same for the two frames. This is illustrated below

![Diagram of Frame K and Frame K+1](image-url)
A straight-line segment in key frame \( k \) is transformed into two line segments in key frame \( k+1 \). Since key frame \( k+1 \) has an extra vertex, we add a vertex between 1 and 2 in key frame \( K \) to balance the number of vertices and edges in the two key frames. Using linear interpolation to generate the in betweens we transition the added vertex in key frame \( k \) into vertex 3’ along the straight-line path as shown.

We can state general preprocessing rules for equalizing key frames in terms of either the number of edges or the number of vertices to be added to a key frame.

**Case 1: Equalizing edge count.**

Let the parameters \( L_k \) and \( L_{k+1} \) denote the number of line segments in two consecutive frames. We then define

\[
\begin{align*}
L_{\text{max}} &= \max(L_k, L_{k+1}) \\
L_{\text{min}} &= \min(L_k, L_{k+1}) \\
N_e &= L_{\text{max}} \mod L_{\text{min}} \\
N_s &= \text{int}(L_{\text{max}} / L_{\text{min}})
\end{align*}
\]

Then the preprocessing is accomplished by

1. Dividing the \( N_e \) edges of keyframe_{\text{min}} into \( N_s \) sections.
2. Dividing the remaining lines of keyframe_{\text{min}} into \( N_s \) sections

**Case 2: Equalizing vertex count**
Let the parameters be \( V_k \) and \( V_{k+1} \) denote the number of vertices in the two consecutive frames. We define

\[
\begin{align*}
V_{\text{max}} &= \max(V_k, V_{k+1}) \\
V_{\text{min}} &= \min(V_k, V_{k+1}) \\
N_{ls} &= (V_{\text{max}}-1) \mod (V_{\text{min}}-1) \\
N_p &= \text{int} \left( \frac{(V_{\text{max}}-1)}{(V_{\text{min}}-1)} \right)
\end{align*}
\]

Preprocessing using vertex count is performed by
1. Adding \( N_p \) points to \( N_{ls} \) line sections of keyframe
2. Adding \( N_p-1 \) points to the remaining edges of keyframe

**Light Sources**

Every object in a scene is potentially a source of light. Light may be either or reflected from objects. Generally, in computer graphics we make a distinction between light emitters and light reflectors. The emitters are called light sources, and the reflectors are usually the objects being rendered. Light sources are characterized by their intensities while reflectors are characterized by their material properties.

**Point Light Sources**

The rays emitted from a point light radially diverge from the source. A point light source is a fair approximation to a local light source such as a light bulb. The direction of the light to each point on a surface changes when a point light source is used. Thus a normalized vector to the light emitter must be computed for each point that is illuminated.

**Basic illumination models**

Illumination models model the interaction of light with the surface and range from simple to very complex.

a) **Local Illumination** = direct illumination - considers light traveling directly from source to surface
b) **Global Illumination** = indirect illumination - takes into account reflection of light from other surfaces

**Ambient Light**

Even though an object in a scene is not directly lit it will still be visible. This is because light is reflected from nearby objects. Ambient light refers to any outside light such as sunlight coming through windows or overhead room light. i.e. Ambient light is the general background light all around us. Ambient light has no spatial or directional characteristics. The amount of ambient light incident on each object is a constant for all surfaces in the scene.

Let

\[ I_a = \text{Ambient light intensity} \]
\[ k_a = \text{Ambient light reflected} \]

Then intensity on the surface is described by

\[ I = I_a k_a \]

**Ideal Diffuse Reflection**

First, we will consider a particular type of surface called an ideal diffuse reflector. An ideal diffuse surface is, at the microscopic level, a very rough surface. Chalk is a good approximation to an ideal diffuse surface.

Because of the microscopic variations in the surface, an incoming ray of light is equally likely to be reflected in any direction over the hemisphere. Ambient light reflection is an approximation of global diffuse lighting effects. Diffuse reflections are constant over each surface in a scene, independent of the viewing direction. The intensity of the diffuse reflection at any point is given by

\[ I_{amb\text{diff}} = K_d I_a \]

where, \( K_d \rightarrow \text{Coefficient of diffused reflection.} \)
\( I_a \rightarrow \text{intensity of ambient light} \)

\( K_d \) is assigned a constant value in the interval 0 to 1.

**Lambert's cosine law**
Ideal diffuse reflectors are also called Lambertian reflectors. Ideal diffuse reflectors reflect light according to Lambert's cosine law, (these are sometimes called Lambertian reflectors).

Lambert's law states that the reflected energy from a small surface area in a particular direction is proportional to the cosine of the angle between that direction and the surface normal. Lambert's law determines how much of the incoming light energy is reflected. The amount of energy that is reflected in any one direction is constant in this model. In other words, the reflected intensity is independent of the viewing direction. The intensity does, however, depend on the light source's orientation relative to the surface, and it is this property that is governed by Lambert's law.

If $I_l$ is the intensity of the point light source, then the diffuse reflection equation for a point on the surface can be written as

$$I_{l,diff} = K_d \ I_l \ \cos \theta$$

A surface is illuminated by a point source only if the angle of incidence is in the range 0 to 90 ($\cos \theta$ is in the interval from 0 to 1). When $\cos \theta$ is negative, the light source is behind the surface.
If \( N \) is the unit normal vector to a surface and \( L \) is the unit direction vector to the point light source from a position on the surface, then

\[
\cos \theta = N \cdot L
\]

The diffuse reflection equation for a point source illumination is

\[
I_{\text{diff}} = K_d \, l \cdot (N \cdot L)
\]

According to Lambert’s cosine law the intensity of the reflected light depends on the angle of illumination. A surface that is perpendicular to the direction of the incident light appears brighter than a surface that is at an angle to the direction of the incoming light. As an angle increases less of the incident light falls on the surface.

We can combine the ambient and point source intensity calculations to obtain an expression for the total diffuse reflection.

Total diffuse reflection equation is given by

\[
I_{\text{diff}} = K_a \, I_a + K_d \, l \cdot (N \cdot L)
\]

**Specular Reflection**

\[
I_{\text{spec}} = W(q) \, l \cdot \cos \theta
\]
Specular reflection, on the other hand, is viewpoint dependent. Light striking a specular surface, by Snell's Law, will be reflected at an angle which mirrors the incident light angle, so the viewing angle is very important. Specular reflection forms tight, bright highlights, making the surface appear glossy.

It is the perfect, mirror-like reflection of light (or sometimes other kinds of wave) from a surface, in which light from a single incoming direction is reflected into a single outgoing direction. Such behaviour is described by the law of reflection, which states that the direction of outgoing reflected light and the direction of incoming light make the same angle with respect to the surface normal; this is commonly stated as $\theta_i = \theta_r$.

This is in contrast to diffuse reflection, where incoming light is reflected in a broad range of directions. The most familiar example of the distinction between specular and diffuse reflection would be matte and glossy paints. While both exhibit a combination of specular and diffuse reflection, matte paints has a higher proportion of diffuse reflection and glossy paints have a greater proportion of specular reflection. Very highly polished surfaces, such as high quality mirrors, can exhibit almost perfect specular reflection.

Even when a surface exhibits only specular reflection with no diffuse reflection, not all of the light is necessarily reflected. Some of the light may be absorbed by the materials. Additionally, depending on the type of material behind the surface, some of the light may be transmitted through the surface. For most interfaces
between materials, the fraction of the light that is reflected increases with increasing angle of incidence $\theta_i$. If the light is propagating in a material with a higher index of refraction than the material whose surface it strikes, then total internal reflection may occur.

**Phong Model**

This is an empirical model, which is not based on physics, but physical observation. Phong observed that for very shiny surfaces the specular highlight was small and the intensity fell off rapidly, while for duller surfaces it was larger and fell off more slowly. He decided to let the reflected intensity be a function of $(\cos \alpha)^n$ with $n \geq 200$ for a shiny surface and $n$ small for a dull surface. For a perfect reflector $n$ equals infinity, and for a piece of cardboard $n$ equals 0 or 1. In the diagram below we can see how the function $(\cos \alpha)^n$ behaves for different values of $n$.

![Diagram of Phong Model](image)

Specular reflection is also a function of the light incidence angle $\theta$. An example is glass which has almost no specular reflectance for $\theta = 0$ degrees but a very high specular reflectance for $\theta > 80$ degrees. Some substances, such as copper, actually change color with change in the incidence angle, as shown in the following plot of the reflectance curve as a function of the incident angle for copper.
Warn model

- Simulates studio lighting effects by controlling light intensity in different directions
- control light direction using reflecting surface (only specular)
- control light direction using reflecting surface (only specular)

- Flaps are used to control the amount of light emitted by a source in various directions
• Spot lighting is simulated in warn model

**RGB color model**

The RGB color model is an additive model in which red, green, and blue (often used in additive light models) are combined in various ways to reproduce other colors. The name of the model and the abbreviation ‘RGB’
comes from the three primary colors, red, green, and blue. These three colors should not be confused with the
primary pigments of red, blue, and yellow, known in the art world as ‘primary colors’.

The RGB color model itself does not define what is meant by ‘red’, ‘green’ and ‘blue’, and the results of mixing
them are not exact unless the exact spectral make-up of the red, green and blue primaries are defined. The color
model then becomes an absolute color space, such as RGB or Adobe RGB; see RGB color space for more
details. This article discusses concepts common to all the different RGB color spaces that use the RGB color
model.

Rendering Methods

**Constant Shading**

The simplest shading model for a polygon is 'constant shading', also known as 'faceted shading' or 'flat
shading'. This approach applies an illumination model once to determine a single intensity value that is then
used to shade an entire polygon, and holding the value across the polygon to reconstruct the polygon's shade.
This approach is valid if several assumptions are true:

1. The light source is at infinity, so N.L is constant across the polygon face.
2. The viewer is at infinity, so N.V is constant across the polygon face.
3. The polygon represents the actual surface being modeled, and is not an approximation to a curved
   surface.

**Gouraud Shading**

'Gouraud shading also called intensity interpolation shading' or 'color interpolation shading', eliminates
intensity discontinuities.

Gouraud shading extends the concept of interpolated shading applied to individual polygons by interpolating
polygon vertex illumination values that take into account the surface being approximated. The Gouraud
shading process requires that the normal {perpendicular vector} be known for each vertex of the polygonal
mesh. Gouraud was able to compute these 'vertex normals' directly from an analytical description of the
surface. Alternatively, if the vertex normals are not stored with the mesh and cannot be determined directly
from the actual surface, then, Gouraud suggested, we can approximate them by averaging the surface normals
of all polygonal facets sharing each vertex. If an edge is meant to be visible (as at the joint between a plane's wing and body), then we find two vertex normals, one for each side of the edge, by averaging the normals of polygons on each side of the edge separately.

The next step in Gouraud shading is to find 'vertex intensities' by using the vertex normals with any desired illumination model. Finally, each polygon is shaded by linear interpolation of vertex intensities along each edge and then between edges along each scan line. The term 'Gouraud shading' is often generalized to refer to intensity interpolation shading of even a single polygon in isolation, or to the interpolation of arbitrary colors associated with polygon vertices (as is the case in computer games).

The algorithm can be stated as follows:

1. Compute a normal \( N \) for each vertex of the polygon.
2. From \( N \) compute intensity \( I \) for each vertex of the polygon.
3. From bi-linear interpolation compute intensity \( I_i \) for each pixel.
4. Paint pixel to shade corresponding to \( I_i \).

**Phong Shading**

The third shading model, Phong shading, is similar to Gouraud shading except that the Normals are interpolated. Thus, the specular highlights are computed much more precisely than in the Gouraud shading model.

The algorithm is as follows:

1. Compute a normal \( N \) for each vertex of the polygon.
2. From bi-linear interpolation compute a normal, \( N_i \) for each pixel. (This must be renormalized each time)
3. From \( N_i \) compute intensity \( I_i \) for each pixel of the polygon.
4. Paint pixel to shade corresponding to \( I_i \).

Note that this method is much more computationally intensive than Gouraud shading:
Ray Tracing Method

Our goal is find the color of each point on the view window. We subdivide the view window into small squares, where each square corresponds to one pixel in the final image. If you want to create an image at the resolution of 640x400, you would break up the view window into a grid of 640 squares across and 400 square down. The real problem, then, is assigning a color to each square. This is what ray tracing does.

The objective is to determine the color of each light ray that strikes the view window before reaching the eye. Natural assumption would be that rays are traced starting at their point of origin, the light source, and towards their destination, the eye. Some will reach the eye directly, others will bounce around some and then reach the eye, and many, many more will probably never hit the eye at all. For all the rays that never reach the eye, the effort tracing them was wasted.

In order to save ourselves this wasted effort, we trace only those rays that are guaranteed to hit the view window and reach the eye. It seems at first that it is impossible to know beforehand which rays reach the eye.
After all, any given ray can bounce around the room many times before reaching the eye. However, if we look at the problem backwards, we see that it has a very simple solution. Instead of tracing the rays starting at the light source, we trace them backwards, starting at the eye. Consider any point on the view window whose color we're trying to determine. Its color is given by the color of the light ray that passes through that point on the view window and reaches the eye. We can just as well follow the ray backwards by starting at the eye and passing through the point on its way out into the scene. The two rays will be identical, except for their direction: if the original ray came directly from the light source, then the backwards ray will go directly to the light source. So the backwards method does the same thing as the original method, except it doesn't waste any effort on rays that never reach the eye.

This, then, is how ray tracing works in computer graphics. For each pixel on the view window, we define a ray that extends from the eye to that point. We follow this ray out into the scene and as it bounces off of different objects. The final color of the ray (and therefore of the corresponding pixel) is given by the colors of the objects hit by the ray as bit travels through the scene.

Just as in the light-source-to-eye method it might take a very large number of bounces before the ray ever hits the eye, in backwards method it might take many bounces before the ray every hits the light. Since we need to establish some limit on the number of bounces to follow the ray on, we make the following approximation: every time a ray hits an object, we follow a single new ray from the point of intersection directly towards the light source.

We trace a new ray from each ray-object intersection directly towards the light source.

In the figure we see two rays, a and b, which intersect the purple sphere. To determine the color of a, we follow the new ray a' directly towards the light source. The color of a will then depend on several factors,
discussed in Color and Shading below. As you can see, \( \mathbf{b} \) will be shadowed because the ray \( \mathbf{b}' \) towards the light source is blocked by the sphere itself. Ray \( \mathbf{a} \) would have also been shadowed if another object blocked the ray \( \mathbf{a}' \).