

Chapter 1: Fundamentals of computer graphics

Introduction

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer with help from specialized software and hardware. The development of computer graphics has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

Computer graphics is a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Although the term often refers to the study of three-dimensional computer graphics, it also encompasses two-dimensional graphics and image processing.

History of Computer Graphics

The phrase “Computer Graphics” was coined in 1960 by William Fetter, a graphic designer for Boeing. The field of computer graphics developed with the emergence of computer graphics hardware. Early projects like the Whirlwind and SAGE Projects introduced the CRT as a viable display and interaction interface and introduced the light pen as an input device.

Initial 1960s developments

Further advances in computing led to greater advancements in interactive computer graphics. In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software. Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location.

Sutherland seemed to find the perfect solution for many of the graphics problems he faced. Even today, many standards of computer graphics interfaces got their start with this early Sketchpad program. One example of this is in drawing constraints. If one wants to draw a square for

example, they do not have to worry about drawing four lines perfectly to form the edges of the box. One can simply specify that they want to draw a box, and then specify the location and size of the box. The software will then construct a perfect box, with the right dimensions and at the right location. Another example is that Sutherland's software modeled objects - not just a picture of objects. In other words, with a model of a car, one could change the size of the tires without affecting the rest of the car. It could stretch the body of the car without deforming the tires.

Further 1961 developments

Also in 1961 another student at MIT, Steve Russell, created the first video game, Spacewar. Written for the DEC PDP-1, Spacewar was an instant success and copies started flowing to other PDP-1 owners and eventually even DEC got a copy. E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-gyro gravity attitude control system" in 1963. In this computer generated film, Zajac showed how the attitude of a satellite could be altered as it orbits the Earth. He created the animation on an IBM 7090 mainframe computer. Also at BTL, Ken Knowlton, Frank Sindon and Michael Noll started working in the computer graphics field. Sindon created a film called Force, Mass and Motion illustrating Newton's laws of motion in operation. Around the same time, other scientists were creating computer graphics to illustrate their research. At Lawrence Radiation Laboratory, Nelson Max created the films, "Flow of a Viscous Fluid" and "Propagation of Shock Waves in a Solid Form." Boeing Aircraft created a film called "Vibration of an Aircraft."

Ralph Baer, a supervising engineer at Sanders Associates, came up with a home video game in 1966 that was later licensed to Magnavox and called the Odyssey. While very simplistic, and requiring fairly inexpensive electronic parts, it allowed the player to move points of light around on a screen. It was the first consumer computer graphics product.

David C. Evans was director of engineering at Bendix Corporation's computer division from 1953 to 1962, after which he worked for the next five years as a visiting professor at Berkeley. There he continued his interest in computers and how they interfaced with people. In 1966, the University of Utah recruited Evans to form a computer science program, and computer graphics quickly became his primary interest. This new department would become the world's primary research center for computer graphics.

Also in 1966, Sutherland at MIT invented the first computer controlled head-mounted display (HMD). Called the Sword of Damocles because of the hardware required for support, it displayed two separate wireframe images, one for each eye. This allowed the viewer to see the computer scene in stereoscopic 3D. After receiving his Ph.D. from MIT, Sutherland became Director of Information Processing at ARPA (Advanced Research Projects Agency), and later became a professor at Harvard.

In 1967 Sutherland was recruited by Evans to join the computer science program at the University of Utah. There he perfected his HMD. Twenty years later, NASA would re-discover his techniques in their virtual reality research. At Utah, Sutherland and Evans were highly sought after consultants by large companies but they were frustrated at the lack of graphics hardware available at the time so they started formulating a plan to start their own company.

1970s

Many of the most important early breakthroughs in computer graphics research occurred at the University of Utah in the 1970s. A student by the name of Edwin Catmull started at the University of Utah in 1970 and signed up for Sutherland's computer graphics class. Catmull had just come from The Boeing Company and had been working on his degree in physics. Growing up on Disney, Catmull loved animation yet quickly discovered that he did not have the talent for drawing. Now Catmull (along with many others) saw computers as the natural progression of animation and they wanted to be part of the revolution. The first animation that Catmull saw was his own. He created an animation of his hand opening and closing. It became one of his goals to produce a feature length motion picture using computer graphics. In the same class, Fred Parke created an animation of his wife's face. Because of Evan's and Sutherland's presence, UU was gaining quite a reputation as the place to be for computer graphics research so Catmull went there to learn 3D animation.

As the UU computer graphics laboratory was attracting people from all over, John Warnock was one of those early pioneers; he would later found Adobe Systems and create a revolution in the publishing world with his PostScript page description language. Tom Stockham led the image processing group at UU which worked closely with the computer graphics lab. Jim Clark was also there; he would later found Silicon Graphics, Inc.

1980s

In the early 1980s, the availability of bit-slice and 16-bit microprocessors started to revolutionise high resolution computer graphics terminals which now increasingly became intelligent, semi-standalone and standalone workstations. Graphics and application processing were increasingly migrated to the intelligence in the workstation, rather than continuing to rely on central mainframe and mini-computers. Typical of the early move to high resolution computer graphics intelligent workstations for the computer-aided engineering market were the Orca 1000, 2000 and 3000 workstations, developed by Orcatech of Ottawa, a spin-off from Bell-Northern Research, and led by an early workstation pioneer David John Pearson. The Orca 3000 was based on Motorola 68000 and AMD bit-slice processors and had Unix as its operating system. It was targeted squarely at the sophisticated end of the design engineering sector. Artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology.

1990s

3D graphics became more popular in the 1990s in gaming, multimedia and animation. At the end of the 80s and beginning of the nineties were created, in France, the very first computer graphics TV series: "La Vie des bêtes" by studio Mac Guff Ligne (1988), Les Fables Géométriques J.-Y. Grall, Georges Lacroix and Renato (studio Fantome, 1990–1993) and Quarxs, the first HDTV computer graphics series by Maurice Benayoun and François Schuiten (studio Z-A production, 1991–1993). In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas worldwide. In 1996, Quake, one of the first fully 3D games, was released. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modeling software.

Applications of Computer Graphics

1. Computer Aided Design - also known as computer-aided design and drafting (CADD) , is the use of computer technology for the process of design and design-documentation. Computer Aided Drafting describes the process of drafting with a computer. CADD software, or environments, provides the user with input-tools for the purpose of streamlining design processes; drafting, documentation, and manufacturing processes. CADD output is often in the form of electronic files for print or machining operations. The development of CADD-based software is in direct correlation with the processes it seeks to economize; industry-based software (construction, manufacturing, etc.) typically uses vector-based (linear) environments whereas graphic-based software utilizes raster-based (pixelated) environments.

CADD environments often involve more than just shapes. As in the manual drafting of technical and engineering drawings, the output of CAD must convey information, such as materials, processes, dimensions, and tolerances, according to application-specific conventions.

CAD is an important industrial art extensively used in many applications, including automotive, shipbuilding, and aerospace industries, industrial and architectural design, prosthetics, and many more. CAD is also widely used to produce computer animation for special effects in movies, advertising and technical manuals. The modern ubiquity and power of computers means that even perfume bottles and shampoo dispensers are designed using techniques unheard of by engineers of the 1960s. Because of its enormous economic importance, CAD has been a major driving force for research in computational geometry, computer graphics (both hardware and software), and discrete differential geometry.

The design of geometric models for object shapes, in particular, is occasionally called computer-aided geometric design (CAGD).

2. Computer art - is any art in which computers play a role in production or display of the artwork. Such art can be an image, sound, animation, video, CD-ROM, DVD-ROM, videogame, web site, algorithm, performance or gallery installation. Many traditional disciplines are now integrating digital technologies and, as a result, the lines between traditional works of art and new media works created using computers has been blurred.

For instance, an artist may combine traditional painting with algorithm art and other digital techniques. As a result, defining computer art by its end product can thus be difficult. Computer art is by its nature evolutionary since changes in technology and software directly affect what is possible.

3. Image processing - in computer science, image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or, a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it. The Digital image processing is the use of computer algorithms to perform image processing on digital images. Digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and signal distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modeled in the form of Multidimensional Systems.
4. Presentation graphics program - is a computer software package used to display information, normally in the form of a slide show. It typically includes three major functions: an editor that allows text to be inserted and formatted, a method for inserting and manipulating graphic images and a slide-show system to display the content. Examples are Microsoft PowerPoint, Corel Presentations and Google Docs
5. Visualization - is any technique for creating images, diagrams, or animations to communicate a message. Visualization through visual imagery has been an effective way to communicate both abstract and concrete ideas since the dawn of man.

Visualization today has ever-expanding applications in science, education, engineering (e.g. product visualization), interactive multimedia, medicine, etc. Typical of a visualization application is the field of computer graphics. The invention of computer graphics may be the most important development in visualization since the invention of central perspective in the Renaissance period. Scientific visualization is the use of interactive, sensory representations, typically visual, of abstract data to reinforce cognition, hypothesis building and reasoning. Data visualization is a related subcategory

of visualization dealing with statistical graphics and geographic or spatial data (as in thematic cartography) that is abstracted in schematic form.

6. Entertainment – Computer graphics are now used in creating motion pictures, music videos and television shows. Sometimes the graphics scenes are displayed by themselves and sometimes graphics objects are combined with actors and live scenes.
7. Computational biology involves the development and application of data-analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological, behavioral, and social systems.
8. A computer simulation, a computer model, or a computational model - is a computer program, or network of computers, that attempts to simulate an abstract model of a particular system. Computer simulations have become a useful part of mathematical modeling of many natural systems in physics (computational physics), astrophysics, chemistry and biology, human systems in economics, psychology, social science, and engineering. Simulations can be used to explore and gain new insights into new technology, and to estimate the performance of systems too complex for analytical solutions. Computer simulations vary from computer programs that run a few minutes, to network-based groups of computers running for hours, to ongoing simulations that run for days. The scale of events being simulated by computer simulations has far exceeded anything possible (or perhaps even imaginable) using the traditional paper-and-pencil mathematical modeling.
9. Virtual reality (VR) - is a term that applies to computer-simulated environments that can simulate physical presence in places in the real world, as well as in imaginary worlds. Most current virtual reality environments are primarily visual experiences, displayed either on a computer screen or through special stereoscopic displays, but some simulations include additional sensory information, such as sound through speakers or headphones.
10. Computer animation - is the process used for generating animated images by using computer graphics. The more general term computer generated imagery encompasses both static scenes and dynamic images, while computer animation only refers to moving images. To create the illusion of movement, an image is displayed on the computer screen and repeatedly replaced by a new image that is similar to the previous image, but

advanced slightly in the time domain (usually at a rate of 24 or 30 frames/second). This technique is identical to how the illusion of movement is achieved with television and motion pictures.

11. A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a video device. The word video in video game traditionally referred to a raster display device, but following popularization of the term "video game", it now implies any type of display device. The electronic systems used to play video games are known as platforms; examples of these are personal computers and video game consoles. These platforms range from large mainframe computers to small handheld devices. Specialized video games such as arcade games, while previously common, have gradually declined in use.

Chapter 2: Concepts, Terms and Definitions

Introduction

Before examining to different specialisms that make up computer graphics, this chapter provides a quick review of the basic terms and definitions that are common to most areas of computer graphics.

Low level concepts

All of computer graphics is based upon the properties of the screen or display device. The fundamental thing that you need to know about displays is that they are divided into lots of small squares called pixels (“PICture ELements”).

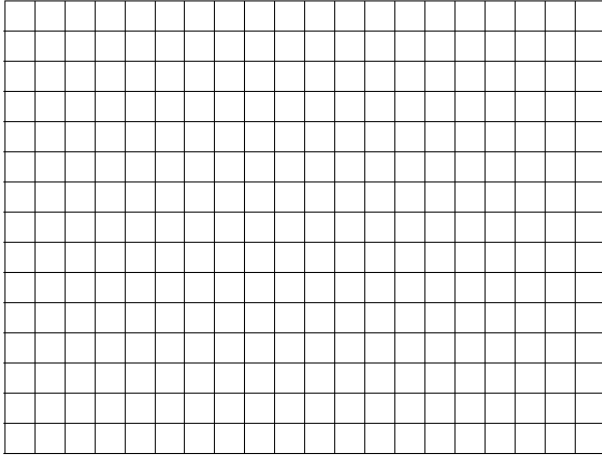


Figure 1 Pixels

The simplest way to think of how this works is to stick to black and white. Each pixel can be set to black or white (i.e. turned on or off). This allows patterns of dots to be created on the screen.

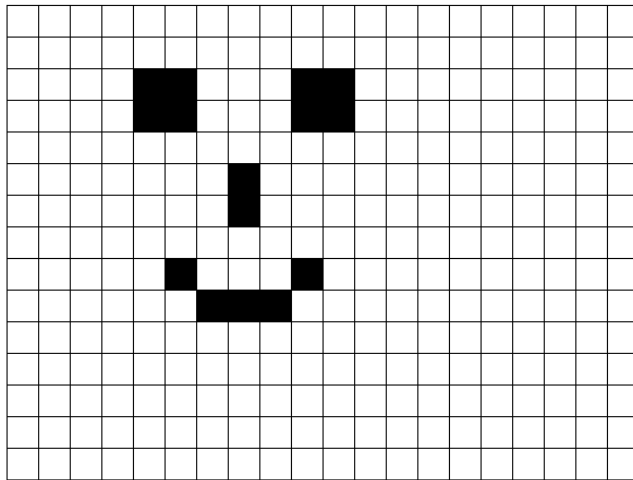
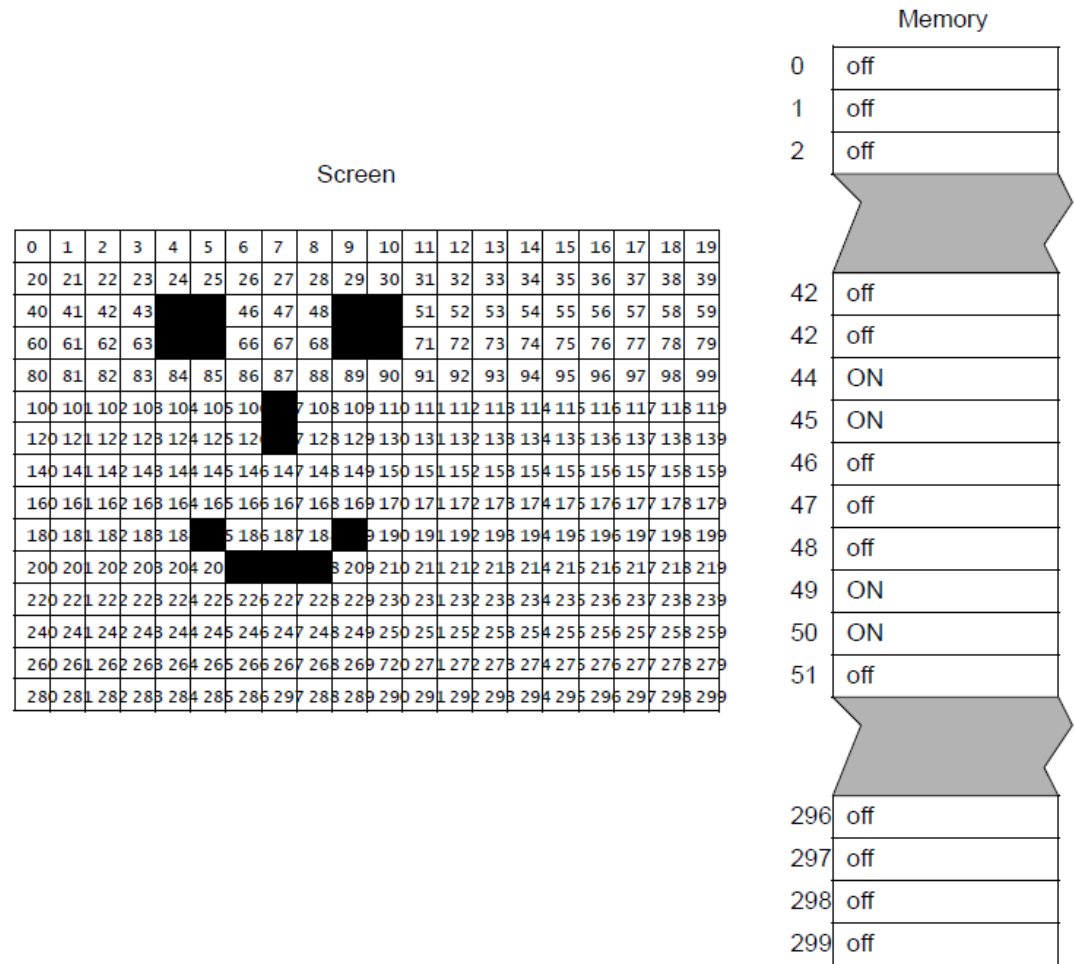


Figure Smiley pixels

Memory Mapping

Drawing on the screen is therefore simply a matter of setting the right pixels either on or off. Each pixel on the screen corresponds to an address in the computer’s memory - this

is known as memory mapping and the display is said to be a “memory mapped display.” Effectively, each pixel is numbered - sequentially.



By writing values to the correct locations in memory (this used to be called “poking the address”) the appearance of the screen can be controlled by a programmer. Conversely, by inspecting the contents of a memory location (“peeking”), a program can find out if a pixel is turned on or off.

The portion of memory that is associated with the display is known as the “video memory”. In the PC architecture, this memory is usually physically located on the graphics card, which is why you can buy 8Mb graphics cards, 16Mb graphics cards etc.

Resolution

The screen in this example is composed of 300 pixels arranged in 15 rows of 20. It is said to have a resolution of “20 by 15” (20 x 15). This is actually very small by today’s standards. Typically, a display will have a resolution of 1024x768 maybe even more.

Screen size

Resolution is NOT the same thing as screen size. Our 20x15 display could have been 5cm across (as a mobile phone display), 40cm across (as a monitor) or 20m (as a projection system) but the resolution would always be 20x15.

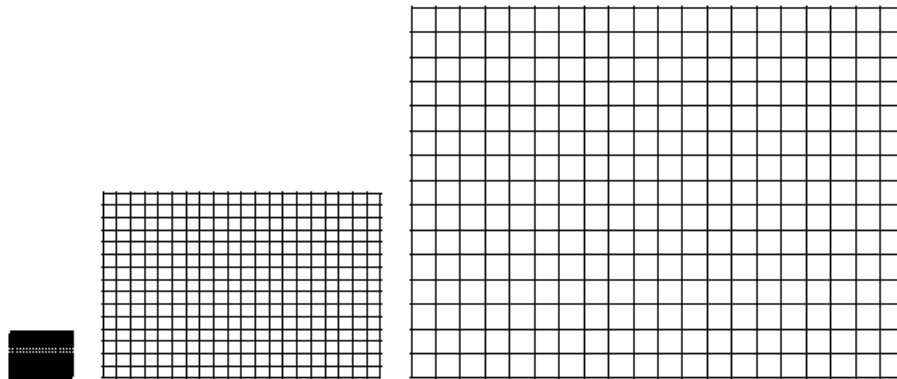
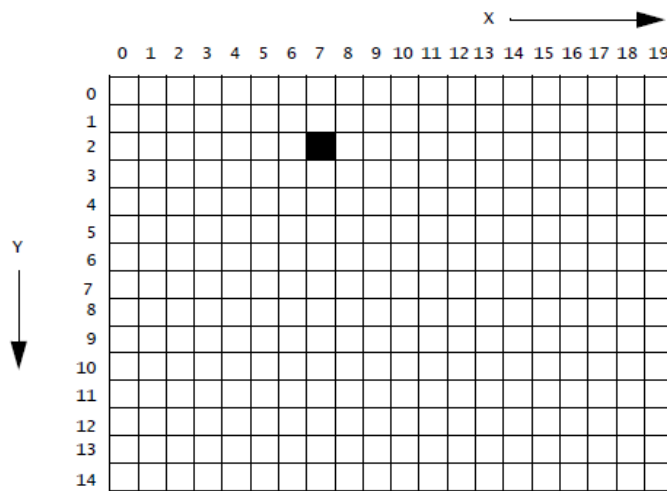


Figure Screen size is different to resolution

Coordinates

Whilst the computer “thinks” of its display as a simple list of addresses, it is much more convenient (for reasons which will become clear later) for us to think in terms of coordinates and leave it to the computer to convert the coordinates to memory location itself.



Xsize = 20 (columns)
Ysize = 15 (rows)

Figure Coordinates

Each pixel can be referred to by a pair of numbers known as its coordinates - an x coordinate (which gives the column's number and a y coordinate which gives the row number. The coordinates are always written as a pair of numbers, separated by a comma and enclosed in brackets. This system of geometry is known as "Cartesian geometry" and the coordinates are spoken of as being "Cartesian Coordinates."

Example. Simple coordinate example

In Figure above - "Coordinates" - pixel (7,2) is set ON.

The computer converts coordinates to memory addresses by subtracting the y coordinate from the number of rows on the display minus 1 (i.e. Ysize -1, this effectively turns the y axis upside-down to make sure that the origin is at the bottom), multiplying that by the number of columns on the display

(Xsize), and adding the x coordinate.

location = ((Y * Xsize) + x).

(7,12) becomes ((2*20) + 7) = 47

The origin

In this above example, the origin or (0,0) is in the bottom-left of the screen. It is also possible to have the origin in the top-left, in which case the rows (y coordinates) are numbered downwards

Colour

In discussion so far, we have been restricted to black and white i.e. each pixel can only be on or off. Colour can be represented in most of today's computers. Typically, instead of each pixel being represented by one bit (on or off) each pixel will be represented by 24 bits - 3 x 8 bit bytes. Each byte represents a number between 0 and 255 and each byte is associated with one primary colour - red, blue, green.



	red	green	blue	
	0000 0000	0000 0000	1111 1111	binary
	0	0	255	decimal
	00	00	FF	hex
	1100 0000	1100 0000	0000 0000	binary
	192	192	0	decimal
	C0	C0	00	hex

Figure 24 bit colour representation

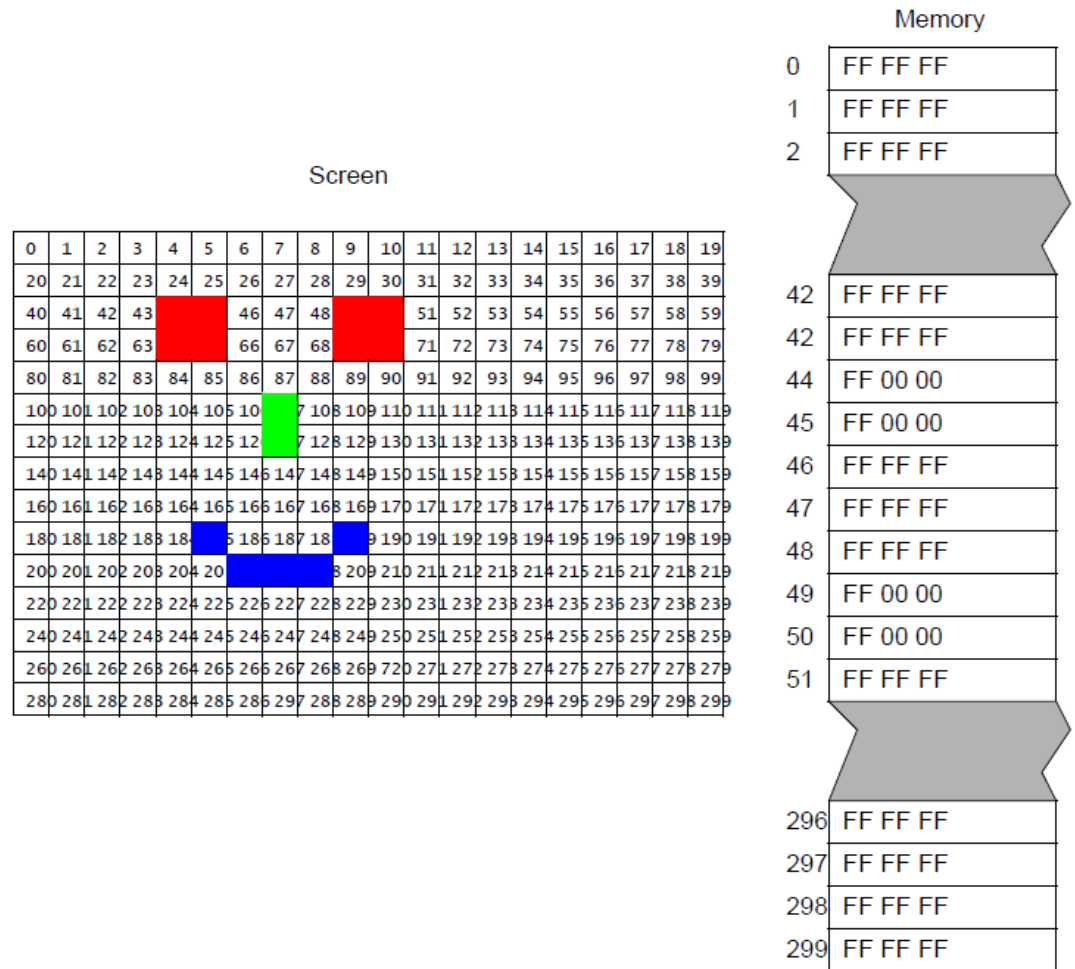


Image files

From Figure above - “Colour smiley” it is hopefully only a small step to see how Figure below - “smiley.bmp” works. The pattern of bits in the computer’s memory forms what is know as a bitmap.

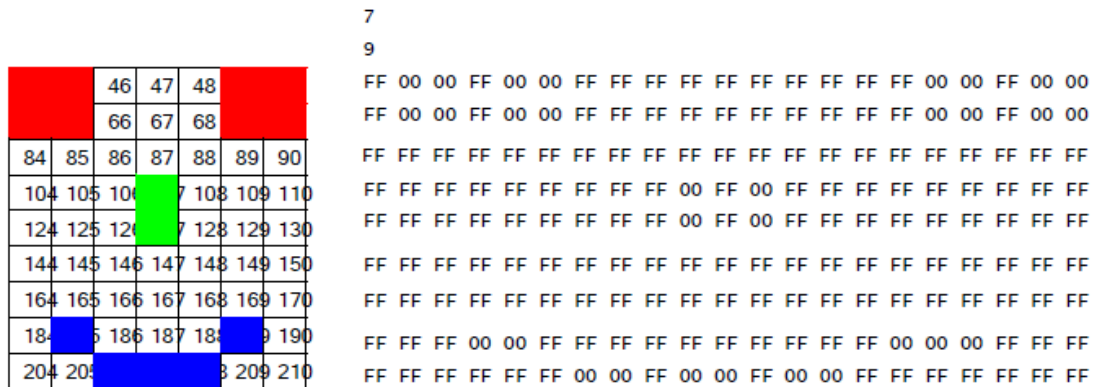


Figure smiley.bmp

By simply writing those values out to a data file and prepending information about the width and depth of the image, a picture can be saved onto disk and this is indeed the approach taken by all the common graphical formats seen on the WWW such as.gifs,.jpgs, etc.

It is then an even smaller step (increase the number of pixels (i.e the resolution)) to see how Figure below - “Carol Smiley” works.



Figure Carol Smiley ¹

Image file formats

Image file formats are standardized means of organizing and storing digital images. Image files are composed of digital data in one of these formats that can be *rasterized* for use on a computer display or printer. An image file format may store data in uncompressed, compressed, or vector formats. Once rasterized, an image becomes a grid of pixels, each of which has a number of bits to designate its color equal to the color depth of the device displaying it.

Image file sizes

In raster images, Image file size is positively correlated to the number of pixels in an image and the color depth, or bits per pixel, of the image. Images can be compressed in various ways, however. Compression uses an algorithm that stores an exact representation or an approximation of the original image in a smaller number of bytes that can be expanded back to its uncompressed form with a corresponding decompression algorithm. Considering different compressions, it is common for two images of the same number of pixels and color depth to have a very different compressed file size. Considering exactly the same compression, number of pixels, and color depth for two images, different graphical complexity of the original images may also result in very different file sizes after compression due to the nature of compression algorithms. With some compression formats, images that are less complex may result in smaller compressed file sizes. This characteristic sometimes results in a smaller file size for some lossless formats than lossy formats. For example, simple images may be losslessly

compressed into a GIF or PNG format and result in a smaller file size than a lossy JPEG format.

Vector images, unlike raster images, can be any dimension independent of file size. File size increases only with the addition of more vectors.

Image file compression

There are two types of image file compression algorithms: lossless and lossy.

1. ***Lossless compression algorithms*** reduce file size while preserving a perfect copy of the original uncompressed image. Lossless compression generally, but not exclusively, results in larger files than lossy compression. Lossless compression should be used to avoid accumulating stages of re-compression when editing images.
2. ***Lossy compression algorithms*** preserve a representation of the original uncompressed image that may appear to be a perfect copy, but it is not a perfect copy. Oftentimes lossy compression is able to achieve smaller file sizes than lossless compression. Most lossy compression algorithms allow for variable compression that trades image quality for file size.

Major graphic file formats

1. Raster formats

- a. GIF- stands for graphics interchange format, a bit-mapped graphics file format used by the World Wide Web, CompuServe and many bulletin board system. GIF supports color and various resolutions. It also includes data compression, but because it is limited to 256 colors, it is more effective for scanned images such as illustrations rather than color photos.
- b. JPEG - Short for Joint Photographic Experts Group, and pronounced jay-peg. JPEG is a lossy compression technique for color images. Although it can reduce files sizes to about 5% of their normal size, some detail is lost in the compression.
- c. TIFF - Acronym for tagged image file format, one of the most widely supported file formats for storing bit-mapped images on personal computers (both PCs and Macintosh computers). Other popular formats are BMP and PCX. TIFF graphics can be any resolution, and they can be black and white, gray-scaled, or color. Files in TIFF format often end with a .tif extension.
- d. MPEG - Short for Moving Picture Experts Group, and pronounced m-peg, is a working group of the ISO. The term also refers to the family of digital video compression standards and file formats developed by the group. MPEG generally produces better-quality video than competing formats, such as Video for Windows, Indeo and QuickTime. MPEG files previously on PCs needed hardware decoders (codecs) for MPEG processing. Today, however, PCs can use software-only codecs including products from RealNetworks, QuickTime or Windows Media Player. MPEG algorithms compress data to form small bits that can be easily transmitted and then decompressed. MPEG achieves its high compression rate by storing only the changes from one frame to another, instead of each entire frame. The video information is then encoded using a technique called Discrete Cosine Transform (DCT). MPEG uses a type of lossy compression, since some data is removed. But the diminishment of data is generally imperceptible to the human eye.
- e. PNG - Short for Portable Network Graphics, and pronounced ping, a new bit-mapped graphics format similar to GIF. In fact, PNG was approved as a standard

by the World Wide Web consortium to replace GIF because GIF uses a patented data compression algorithm called LZW. In contrast, PNG is completely patent- and license-free. The most recent versions of Netscape Navigator and Microsoft Internet Explorer now support PNG.

- f. BMP - The standard bit-mapped graphics format used in the Windows environment. By convention, graphics files in the BMP format end with a.BMP extension. BMP files store graphics in a format called device-independent bitmap (DIB).

2. *Vector formats*

As opposed to the raster image formats above (where the data describes the characteristics of each individual pixel), vector image formats contain a geometric description which can be rendered smoothly at any desired display size. At some point, all vector graphics must be rasterized in order to be displayed on digital monitors. However, vector images can be displayed with analog CRT technology such as that used in some electronic test equipment, medical monitors, radar displays, laser shows and early video games. Plotters are printers that use vector data rather than pixel data to draw graphics.

- a. CGM - CGM (Computer Graphics Metafile) is a file format for 2D vector graphics, raster graphics, and text, and is defined by ISO/IEC 8632. All graphical elements can be specified in a textual source file that can be compiled into a binary file or one of two text representations. CGM provides a means of graphics data interchange for computer representation of 2D graphical information independent from any particular application, system, platform, or device. It has been adopted to some extent in the areas of technical illustration and professional design, but has largely been superseded by formats such as SVG and DXF.
- b. Gerber Format (RS-274X) - RS-274X Extended Gerber Format was developed by Gerber Systems Corp., now Ucamco. This is a 2D bi-level image description format. It is the de-facto standard format used by printed circuit board or PCB software. It is also widely used in other industries requiring high-precision 2D bi-level images.
- c. SVG - SVG (Scalable Vector Graphics) is an open standard created and developed by the World Wide Web Consortium to address the need (and attempts of several corporations) for a versatile, scriptable and all-purpose vector format for the web and otherwise. The SVG format does not have a compression scheme of its own, but due to the textual nature of XML, an SVG graphic can be compressed using a program such as gzip. Because of its scripting potential, SVG is a key component in web applications: interactive web pages that look and act like applications.

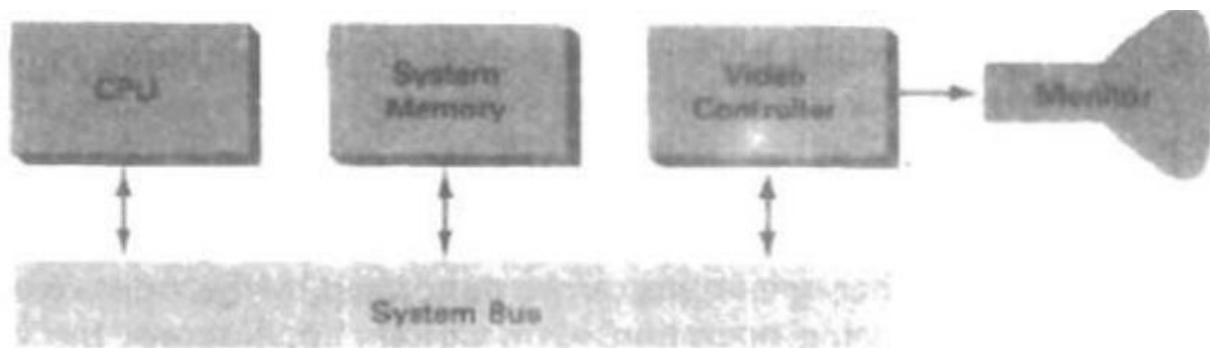
Chapter 3: Overview of graphic systems

Output devices

An output device is any piece of computer hardware equipment used to communicate the results of data processing carried out by an information processing system (such as a computer) to the outside world. Input/output, or I/O, refers to the communication between an information processing system (such as a computer), and the outside world. Inputs are the signals or data sent to the system, and outputs are the signals or data sent by the system to the outside. Examples include Speakers, Headphones, Screen (Monitor) and Printer.

Raster-Scan Systems

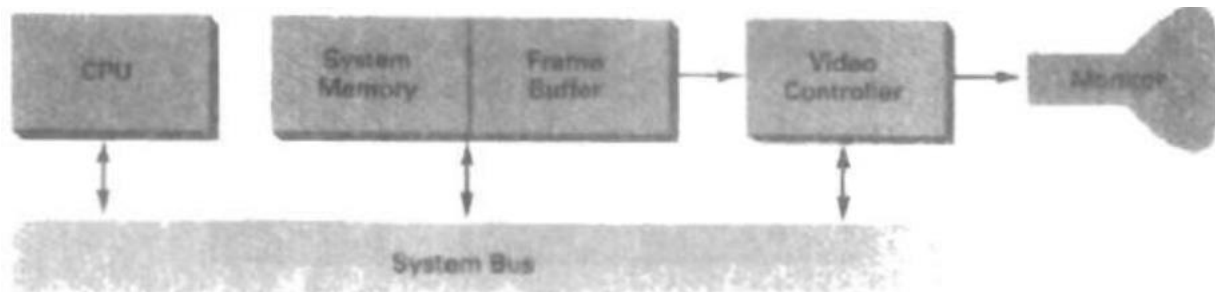
Interactive raster graphics systems typically employ several processing units. In addition to the central processing unit, or CPU, a special-purpose processor, called the video controller or display controller, is used to control the operation of the display device. Organization of a simple raster system is shown in below. Here, the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen.



Architecture of a simple raster graphics system

In addition to the video controller, more sophisticated raster systems employ other processors as coprocessors and accelerators to implement various graphics operations. Video Controller in the figure below shows a commonly used organization for raster systems. A fixed area of the system memory 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. For many graphics monitors, the coordinate originate at the lower left screen comer.



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

Screen (Monitor)

A monitor or display or visual display unit, is an electronic visual display for computers. The monitor comprises the display device, circuitry, and an enclosure. The display device in modern monitors is typically a thin film transistor liquid crystal display (TFT-LCD) thin panel, while older monitors use a cathode ray tube about as deep as the screen size. The first computer monitors used Cathode ray tubes (CRTs), which was the dominant technology until they were replaced by LCD monitors in the 21st Century.

Originally, computer monitors were used for data processing while television receivers were used for entertainment. From the 1980s onwards, computers (and their monitors) have been used for both data processing and entertainment, while televisions have implemented some computer functionality.

The performance of a monitor is measured by the following parameters:

- Luminance is measured in candelas per square meter (cd/m² also called a Nit).
- Aspect ratios are the ratio of the horizontal length to the vertical length. Monitors usually have the aspect ratio 4:3, 5:4, 16:10 or 16:9.
- Viewable image size is usually measured diagonally, but the actual widths and heights are more informative since they are not affected by the aspect ratio in the same way. For CRTs, the viewable size is typically 1 in (25 mm) smaller than the tube itself.

- Display resolution is the number of distinct pixels in each dimension that can be displayed. Maximum resolution is limited by dot pitch.
- Dot pitch is the distance between subpixels of the same color in millimeters. In general, the smaller the dot pitch, the sharper the picture will appear.
- Refresh rate is the number of times in a second that a display is illuminated. Maximum refresh rate is limited by response time.
- Response time is the time a pixel in a monitor takes to go from active (white) to inactive (black) and back to active (white) again, measured in milliseconds. Lower numbers mean faster transitions and therefore fewer visible image artifacts.
- Contrast ratio is the ratio of the luminosity of the brightest color (white) to that of the darkest color (black) that the monitor is capable of producing.
- Power consumption is measured in watts.
- Viewing angle is the maximum angle at which images on the monitor can be viewed, without excessive degradation to the image. It is measured in degrees horizontally and vertically.

Display size

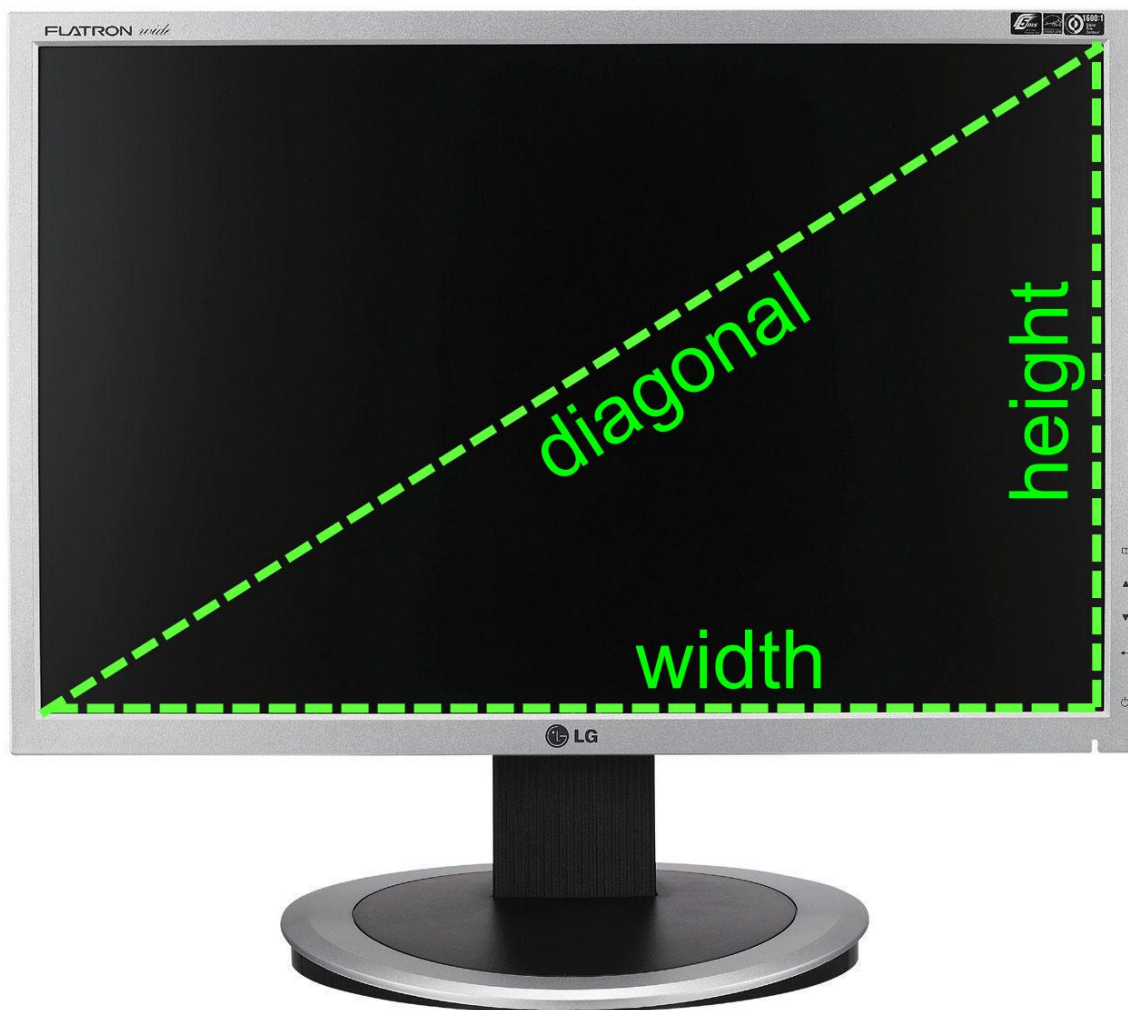
On two-dimensional display devices such as computer monitors and television sets, the display size (or viewable image size or VIS) is the actual amount of screen space that is available to display a picture, video or working space, without obstruction from the case or other aspects of the unit's design. The viewable image size (often measured in inches or millimeters) is sometimes called the physical image size, to distinguish it from the "logical image size" (display resolution, measured in pixels) of various computer display standards.

The size of an approximately rectangular display is usually given by monitor manufacturers as the distance between two opposite screen corners, that is, the diagonal of the rectangle. This method of measurement is inherited from the method used for the first generation of CRT television, when picture tubes with circular faces were in common use. Being circular, only their diameter was needed to describe their size. Since these circular tubes were used to display rectangular images, the diagonal measurement of the rectangle was equivalent to the diameter of the tube's face. This method continued even when cathode ray tubes were manufactured as rounded rectangles; it had the advantage of being a single number specifying the size, and was not confusing when the aspect ratio was universally 4:3.

Liquid crystal displays use 100% of their surface to display, thus the VIS is the actual size presented to the viewer.

The main measurements for display devices are:

- Width and height
- Total area
- The diagonal, usually measured in inches. The diagonal measurement of a display as the actual viewable area is also dependent on the aspect ratio of the display. For a display device of the same area, the diagonal achieves a larger value in a widescreens than compared to more rectangular screen. There is a risk of manufacturers making displays wider than what is actually most convenient in order to make the diagonal achieve a larger value.



Aspect ratio

The aspect ratio of an image is the ratio of the width of the image to its height, expressed as two numbers separated by a colon. That is, for an $x:y$ aspect ratio, no matter how big or small the image is, if the width is divided into x units of equal length and the height is measured using this same length unit, the height will be measured to be y units. For example, consider a group of images, all with an aspect ratio of 16:9. One image is 16 inches wide and 9 inches high. Another image is 16 centimeters wide and 9 centimeters high. A third is 8 yards wide and 4.5 yards high. Aspect ratios are mathematically expressed as $x:y$ (pronounced "x-to-y") and $x \times y$ (pronounced "x-by-y"), with the latter particularly used for pixel dimensions, such as 640×480. Cinematographic aspect ratios are usually denoted as a (rounded) decimal multiple of width vs unit height, while photographic and videographic aspect ratios are usually defined and denoted by whole number ratios of width to height. In digital images there is a subtle distinction between the Display Aspect Ratio (the image as displayed) and the Storage Aspect Ratio (the ratio of pixel dimensions).

The most common aspect ratios used today in the presentation of films in movie theaters are 1.85:1 and 2.39:1. Two common videographic aspect ratios are 4:3 (1.33:1), the universal video format of the 20th century and ; 16:9 (1.77:1), universal for high-definition television and European digital television. Other cinema and video aspect ratios exist, but are used infrequently. As of 2010, nominally 21:9 (2.33) aspect TVs have been introduced by Philips and Vizio (the latter using an LCD from AU Optronics) as "cinema" displays, though the resolution is more precisely $2560 / 1080 = 2.37$, and the aspect ratio is not standardized in HDTV.

In still camera photography, the most common aspect ratios are 4:3, 3:2, and more recently being found in consumer cameras 16:9. Other aspect ratios, such as 5:4, 6:7, and 1:1 (square format), are used in photography as well, particularly in medium format and large format.

With television, DVD and Blu-ray, converting formats of unequal ratios is achieved by either:

1. enlarging the original image (by the same factor in both directions) to fill the receiving format's display area and cutting off any excess picture information (zooming and cropping),
2. by adding horizontal mattes (letterboxing) or vertical mattes (pillarboxing) to retain the original format's aspect ratio, or (for TV and DVD) by stretching (hence distorting) the image to fill the receiving format's ratio, by scaling by different factors in both directions, possibly scaling by a different factor in the center and at the edges (as in Wide Zoom mode).

Display resolution

The display resolution of a digital television or display device is the number of distinct pixels in each dimension that can be displayed. It can be an ambiguous term especially as the displayed resolution is controlled by all different factors in cathode ray tube (CRT), flat panel or projection displays using fixed picture-element (pixel) arrays.

It is usually quoted as width \times height, with the units in pixels: for example, "1024x768" means the width is 1024 pixels and the height is 768 pixels. This example would normally be spoken as "ten twenty-four by seven sixty-eight".

One use of the term "display resolution" applies to fixed-pixel-array displays such as plasma display panels (PDPs), liquid crystal displays (LCDs), digital light processing (DLP) projectors, or similar technologies, and is simply the physical number of columns and rows of pixels creating the display (e.g., 1920 \times 1080). A consequence of having a fixed grid display is that, for multi-format video inputs, all displays need a "scaling engine" (a digital video processor that includes a memory array) to match the incoming picture format to the display.

Note that the use of the word resolution here is a misnomer, though common. The term "display resolution" is usually used to mean pixel dimensions, the number of pixels in each dimension (e.g., 1920 \times 1080), which does not tell anything about the resolution of the display on which the image is actually formed: resolution properly refers to the pixel density, the number of pixels per unit distance or area, not total number of pixels. In digital measurement, the display resolution would be given in pixels per inch. In analog measurement, if the screen is 10 inches high, then the horizontal resolution is measured across a square 10 inches wide. This is typically stated as "lines horizontal resolution, per picture height;" for example, analog NTSC TVs can typically display 486 lines of "per picture height" horizontal resolution, which is equivalent to 648 total lines of actual picture information from left edge to right edge. Which would give NTSC TV a display resolution of 648 \times 486 in actual lines/picture information, but in "per picture height" a display resolution of 486 \times 486.

Other features

- Power saving - Most modern monitors will switch to a power-saving mode if no video-input signal is received. This allows modern operating systems to turn off a monitor after a specified period of inactivity. This also extends the monitor's service life. Some

monitors will also switch themselves off after a time period on standby. Most modern laptops provide a method of screen dimming after periods of inactivity or when the battery is in use. This extends battery life and reduces wear.

- Integrated accessories - Many monitors have other accessories (or connections for them) integrated. This places standard ports within easy reach and eliminates the need for another separate hub, camera, microphone, or set of speakers. These monitors have advanced microprocessors which contain codec information, Windows Interface drivers and other small software which help in proper functioning of these functions.
- Glossy screen - Some displays, especially newer LCD monitors, replace the traditional anti-glare matte finish with a glossy one. This increases color saturation and sharpness but reflections from lights and windows are very visible.
- Directional screen - Narrow viewing angle screens are used in some security conscious applications.
- Polarized 3D monitor - The LG computer monitor 1920x1080 with screen 21.5-inches to 25-inches have feature the company's Film Pattern Retarder coating that ditches the heavy active shutter glasses used by many other manufacturers for the lighter passive variety will give flicker-free 3D. Although the screen is Full HD 1080p, but to see 3D images with 3D polarizing glasses we may only get a half portion of it (HD 720p) due to the pixels should be divided for the left and right eyes.
- Autostereoscopic (3D) screen - A directional screen which generates 3D images without headgear.
- Touch screen - These monitors use touching of the screen as an input method. Items can be selected or moved with a finger, and finger gestures may be used to convey commands. The screen will need frequent cleaning due to image degradation from fingerprints.
- Tablet screens - A combination of a monitor with a graphics tablet. Such devices are typically unresponsive to touch without the use of one or more special tools' pressure. Newer models however are now able to detect touch from any pressure and often have the ability to detect tilt and rotation as well. Touch and tablet screens are used on LCD displays as a substitute for the light pen, which can only work on CRTs.

Types of Monitors

a. Cathode ray tube

The cathode ray tube (CRT) is a vacuum tube containing an electron gun (a source of electrons) and a fluorescent screen, with internal or external means to accelerate and deflect the electron beam, used to create images in the form of light emitted from the fluorescent screen. The image may represent electrical waveforms (oscilloscope), pictures (television, computer monitor), radar targets and others. CRTs have also been used as memory devices, in which case the visible light emitted from the fluorescent material (if any) is not intended to have significant meaning to a visual observer (though the visible pattern on the tube face may cryptically represent the stored data).

The CRT uses an evacuated glass envelope which is large, deep (i.e. long from front screen face to rear end), fairly heavy, and relatively fragile. As a matter of safety, the face is typically made of thick lead glass so as to be highly shatter-resistant and to block most X-ray emissions, particularly if the CRT is used in a consumer product.

A cathode ray tube is a vacuum tube which consists of one or more electron guns, possibly internal electrostatic deflection plates, and a phosphor target. In television sets and computer monitors, the entire front area of the tube is scanned repetitively and systematically in a fixed pattern called a raster. An image is produced by controlling the intensity of each of the three electron beams, one for each additive primary color (red, green, and blue) with a video signal as a reference. In all modern CRT monitors and televisions, the beams are bent by magnetic deflection, a varying magnetic field generated by coils and driven by electronic circuits around the neck of the tube.

Advantages of CRT

- High dynamic range (up to around 15,000:1), excellent color, wide gamut and low black level. The color range of CRTs is unmatched by any display type except OLED.
- Can display in almost any resolution and refresh rate
- No input lag
- Sub-millisecond response times
- Near zero color, saturation, contrast or brightness distortion. Excellent viewing angle.
- Allows the use of light guns/pens

Disadvantages of CRT

- Large size and weight, especially for bigger screens (a 20-inch (51 cm) unit weighs about 50 lb (23 kg))
- High power consumption
- Generates a considerable amount of heat when running
- Geometric distortion caused by variable beam travel distances
- Can suffer screen burn-in
- Produces noticeable flicker at low refresh rates
- Small color displays, less than 7 inches diagonal measurement, are relatively costly. *The maximum practical size for CRTs is around 24 inches for computer monitors; most direct view CRT televisions are 36 inches or smaller, with regular-production models limited to about 40 inches.

b. A liquid crystal display

A liquid crystal display (LCD) is a flat panel display, electronic visual display, video display that uses the light modulating properties of liquid crystals (LCs). LCs do not emit light directly.

They are used in a wide range of applications, including computer monitors, television, instrument panels, aircraft cockpit displays, signage, etc. They are common in consumer devices such as video players, gaming devices, clocks, watches, calculators, and telephones. LCDs have displaced cathode ray tube (CRT) displays in most applications. They are usually more compact, lightweight, portable, less expensive, more reliable, and easier on the eyes. They are available in a wider range of screen sizes than CRT and plasma displays, and since they do not use phosphors, they cannot suffer image burn-in.

LCDs are more energy efficient and offer safer disposal than CRTs. Its low electrical power consumption enables it to be used in battery-powered electronic equipment. It is an electronically modulated optical device made up of any number of segments filled with liquid crystals and arrayed in front of a light source (backlight) or reflector to produce images in color or monochrome. The most flexible ones use an array of small pixels. The earliest discovery leading to the development of LCD technology, the discovery of liquid crystals, dates from 1888. By 2008, worldwide sales of televisions with LCD screens had surpassed the sale of CRT units.

Each pixel of an LCD typically consists of a layer of molecules aligned between two transparent electrodes, and two polarizing filters, the axes of transmission of which are (in most of the cases) perpendicular to each other. With no actual liquid crystal between the polarizing filters, light

passing through the first filter would be blocked by the second (crossed) polarizer. In most of the cases the liquid crystal has double refraction.

The surface of the electrodes that are in contact with the liquid crystal material are treated so as to align the liquid crystal molecules in a particular direction. This treatment typically consists of a thin polymer layer that is unidirectionally rubbed using, for example, a cloth. The direction of the liquid crystal alignment is then defined by the direction of rubbing. Electrodes are made of a transparent conductor called Indium Tin Oxide (ITO).

Before applying an electric field, the orientation of the liquid crystal molecules is determined by the alignment at the surfaces of electrodes. In a twisted nematic device (still the most common liquid crystal device), the surface alignment directions at the two electrodes are perpendicular to each other, and so the molecules arrange themselves in a helical structure, or twist. This reduces the rotation of the polarization of the incident light, and the device appears grey. If the applied voltage is large enough, the liquid crystal molecules in the center of the layer are almost completely untwisted and the polarization of the incident light is not rotated as it passes through the liquid crystal layer. This light will then be mainly polarized perpendicular to the second filter, and thus be blocked and the pixel will appear black. By controlling the voltage applied across the liquid crystal layer in each pixel, light can be allowed to pass through in varying amounts thus constituting different levels of gray. This electric field also controls (reduces) the double refraction properties of the liquid crystal.

The optical effect of a twisted nematic device in the voltage-on state is far less dependent on variations in the device thickness than that in the voltage-off state. Because of this, these devices are usually operated between crossed polarizers such that they appear bright with no voltage (the eye is much more sensitive to variations in the dark state than the bright state). These devices can also be operated between parallel polarizers, in which case the bright and dark states are reversed. The voltage-off dark state in this configuration appears blotchy, however, because of small variations of thickness across the device.

Both the liquid crystal material and the alignment layer material contain ionic compounds. If an electric field of one particular polarity is applied for a long period of time, this ionic material is attracted to the surfaces and degrades the device performance. This is avoided either by applying an alternating current or by reversing the polarity of the electric field as the device is addressed

(the response of the liquid crystal layer is identical, regardless of the polarity of the applied field).

Displays for a small number of individual digits and/or fixed symbols (as in digital watches, pocket calculators etc.) can be implemented with independent electrodes for each segment. In contrast full alphanumeric and/or variable graphics displays are usually implemented with pixels arranged as a matrix consisting of electrically connected rows on one side of the LC layer and columns on the other side which makes it possible to address each pixel at the intersections. The general method of matrix addressing consists of sequentially addressing one side of the matrix, for example by selecting the rows one-by-one and applying the picture information on the other side at the columns row-by-row.

The various matrix addressing schemes are Passive-matrix and active-matrix addressed LCDs.

Advantages of LCD

- Very compact and light.
- Low power consumption.
- No geometric distortion.
- Little or no flicker depending on backlight technology.
- Not affected by screen burn-in.
- No high voltage or other hazards present during repair/service.
- Can be made in almost any size or shape.
- No theoretical resolution limit.

Disadvantages of LCD

- Limited viewing angle, causing color, saturation, contrast and brightness to vary, even within the intended viewing angle, by variations in posture.
- Bleeding and uneven backlighting in some monitors, causing brightness distortion, especially toward the edges.
- Smearing and ghosting artifacts caused by slow response times (>8 ms) and "sample and hold" operation.
- Only one native resolution. Displaying resolutions either requires a video scaler, lowering perceptual quality, or display at 1:1 pixel mapping, in which images will be physically too large or won't fill the whole screen.

- Fixed bit depth, many cheaper LCDs are only able to display 262,000 colors. 8-bit S-IPS panels can display 16 million colors and have significantly better black level, but are expensive and have slower response time.
- Input lag - Display lag is a phenomenon associated with some types of LCD displays, and nearly all types of HDTVs, that refers to latency, or lag measured by the difference between the time a signal is input into a display and the time it is shown by the display
- Dead or stuck pixels may occur either during manufacturing or through use.
- In a constant on situation, thermalization may occur, which is when only part of the screen has overheated and therefore looks discolored compared to the rest of the screen.
- Not all LCDs are designed to allow easy replacement of the backlight.
- Cannot be used with light guns/pens.

c. Plasma display

A plasma display panel (PDP) is a type of flat panel display common to large TV displays 30 inches (76 cm) or larger. They are called "plasma" displays because the technology utilizes small cells containing electrically charged ionized gases, or what are in essence chambers more commonly known as fluorescent lamps.

Plasma displays are bright (1,000 lux or higher for the module), have a wide color gamut, and can be produced in fairly large sizes—up to 150 inches (3.8 m) diagonally. They have a very low-luminance "dark-room" black level compared to the lighter grey of the unilluminated parts of an LCD screen (i.e. the blacks are blacker on plasmas and greyer on LCDs). LED-backlit LCD televisions have been developed to reduce this distinction. The display panel itself is about 6 cm (2.5 inches) thick, generally allowing the device's total thickness (including electronics) to be less than 10 cm (4 inches). Plasma displays use as much power per square meter as a CRT or an AMLCD (is a type of flat panel display, currently the overwhelming choice of notebook computer manufacturers, due to low weight, very good image quality, wide color gamut and response time.) television. Power consumption varies greatly with picture content, with bright scenes drawing significantly more power than darker ones – this is also true of CRTs. Typical power consumption is 400 watts for a 50-inch (127 cm) screen. 200 to 310 watts for a 50-inch (127 cm) display when set to cinema mode. Most screens are set to 'shop' mode by default, which draws at least twice the power (around 500–700 watts) of a 'home' setting of less extreme brightness. The lifetime of the latest generation of plasma displays is estimated at 100,000 hours

of actual display time, or 27 years at 10 hours per day. This is the estimated time over which maximum picture brightness degrades to half the original value.

Plasma display screens are made from glass, which reflects more light than the material used to make an LCD screen. This causes glare from reflected objects in the viewing area. Companies such as Panasonic coat their newer plasma screens with an anti-glare filter material. Currently, plasma panels cannot be economically manufactured in screen sizes smaller than 32 inches. Although a few companies have been able to make plasma Enhanced-definition televisions (EDTV) this small, even fewer have made 32in plasma HDTVs. With the trend toward large-screen television technology, the 32in screen size is rapidly disappearing.

Plasma display advantages

- Picture quality
 - Capable of producing deeper blacks allowing for superior contrast ratio
 - Wider viewing angles than those of LCD; images do not suffer from degradation at high angles like LCDs
 - Less visible motion blur, thanks in large part to very high refresh rates and a faster response time, contributing to superior performance when displaying content with significant amounts of rapid motion
- Physical
 - Slim profile
 - Can be wall mounted
 - Less bulky than rear-projection televisions

Plasma display Disadvantages

- Picture quality
 - Earlier generation displays were more susceptible to screen burn-in and image retention, although most recent models have a pixel orbiter that moves the entire picture faster than is noticeable to the human eye, which reduces the effect of burn-in but does not prevent it.
 - Earlier generation displays had phosphors that lost luminosity over time, resulting in gradual decline of absolute image brightness (newer models are less susceptible to this, having lifespans exceeding 100,000 hours, far longer than older CRT technology)

- Earlier generation models were susceptible to "large area flicker"
- Heavier screen-door effect when compared to LCD or OLED based TVs
- Physical
 - Generally do not come in smaller sizes than 37 inches
 - Heavier than LCD due to the requirement of a glass screen to hold the gases
- Other
 - Use more electricity, on average, than an LCD TV
 - Do not work as well at high altitudes due to pressure differential between the gases inside the screen and the air pressure at altitude. It may cause a buzzing noise. Manufacturers rate their screens to indicate the altitude parameters.
 - For those who wish to listen to AM radio, or are Amateur Radio operators (Hams) or Shortwave Listeners (SWL), the Radio Frequency Interference (RFI) from these devices can be irritating or disabling.
 - Due to the strong infrared emissions inherent with the technology, standard IR repeater systems can not be used in the viewing room. A more expensive "plasma compatible" sensor must be used.

d. Organic light-emitting diode

An organic light emitting diode (OLED) is a light-emitting diode (LED) in which the emissive electroluminescent layer is a film of organic compounds which emit light in response to an electric current. This layer of organic semiconductor material is situated between two electrodes. Generally, at least one of these electrodes is transparent.

OLEDs are used in television set screens, computer monitors, small, portable system screens such as mobile phones and PDAs, watches, advertising, information, and indication. OLEDs are also used in large-area light-emitting elements for general illumination. Due to their low thermal conductivity, they typically emit less light per area than inorganic LEDs.

An OLED display works without a backlight. Thus, it can display deep black levels and can be thinner and lighter than liquid crystal displays. In low ambient light conditions such as dark rooms, an OLED screen can achieve a higher contrast ratio than an LCD—whether the LCD uses either cold cathode fluorescent lamps or the more recently developed LED backlight.

There are two main families of OLEDs: those based on small molecules and those employing polymers. Adding mobile ions to an OLED creates a Light-emitting Electrochemical Cell or LEC, which has a slightly different mode of operation.

OLED displays can use either passive-matrix (PMOLED) or active-matrix addressing schemes. Active-matrix OLEDs (AMOLED) require a thin-film transistor backplane to switch each individual pixel on or off, but allow for higher resolution and larger display sizes.

Advantages

- Lower cost in the future: OLEDs can be printed onto any suitable substrate by an inkjet printer or even by screen printing, theoretically making them cheaper to produce than LCD or plasma displays. However, fabrication of the OLED substrate is more costly than that of a TFT LCD, until mass production methods lower cost through scalability.
- Light weight & flexible plastic substrates: OLED displays can be fabricated on flexible plastic substrates leading to the possibility of flexible organic light-emitting diodes being fabricated or other new applications such as roll-up displays embedded in fabrics or clothing. As the substrate used can be flexible such as PET (Polyethylene terephthalate), the displays may be produced inexpensively.
- Wider viewing angles & improved brightness: OLEDs can enable a greater artificial contrast ratio (both dynamic range and static, measured in purely dark conditions) and viewing angle compared to LCDs because OLED pixels directly emit light. OLED pixel colours appear correct and unshifted, even as the viewing angle approaches 90° from normal.
- Better power efficiency: LCDs filter the light emitted from a backlight, allowing a small fraction of light through so they cannot show true black, while an inactive OLED element does not produce light or consume power.
- Response time: OLEDs can also have a faster response time than standard LCD screens. Whereas LCD displays are capable of between 2 and 8 ms response time offering a frame rate of ~200 Hz, an OLED can theoretically have less than 0.01 ms response time enabling 100,000 Hz refresh rates.

Disadvantages

- Current costs: OLED manufacture currently requires process steps that make it extremely expensive. Specifically, it requires the use of Low-Temperature Polysilicon backplanes;

LTPS backplanes in turn require laser annealing from an amorphous silicon start, so this part of the manufacturing process for AMOLEDs starts with the process costs of standard LCD, and then adds an expensive, time-consuming process that cannot currently be used on large-area glass substrates.

- **Lifespan:** The biggest technical problem for OLEDs was the limited lifetime of the organic materials. In particular, blue OLEDs historically have had a lifetime of around 14,000 hours to half original brightness (five years at 8 hours a day) when used for flat-panel displays. This is lower than the typical lifetime of LCD, LED or PDP technology—each currently rated for about 25,000 – 40,000 hours to half brightness, depending on manufacturer and model. However, some manufacturers' displays aim to increase the lifespan of OLED displays, pushing their expected life past that of LCD displays by improving light outcoupling, thus achieving the same brightness at a lower drive current. In 2007, experimental OLEDs were created which can sustain 400 cd/m² of luminance for over 198,000 hours for green OLEDs and 62,000 hours for blue OLEDs.
- **Color balance issues:** Additionally, as the OLED material used to produce blue light degrades significantly more rapidly than the materials that produce other colors, blue light output will decrease relative to the other colors of light. This differential color output change will change the color balance of the display and is much more noticeable than a decrease in overall luminance. This can be partially avoided by adjusting colour balance but this may require advanced control circuits and interaction with the user, which is unacceptable for some users. In order to delay the problem, manufacturers bias the colour balance towards blue so that the display initially has an artificially blue tint, leading to complaints of artificial-looking, over-saturated colors. More commonly, though, manufacturers optimize the size of the R, G and B subpixels to reduce the current density through the subpixel in order to equalize lifetime at full luminance. For example, a blue subpixel may be 100% larger than the green subpixel. The red subpixel may be 10% smaller than the green.
- **Efficiency of blue OLEDs:** Improvements to the efficiency and lifetime of blue OLEDs is vital to the success of OLEDs as replacements for LCD technology. Considerable research has been invested in developing blue OLEDs with high external quantum efficiency as well as a deeper blue color. External quantum efficiency values of 20% and

19% have been reported for red (625 nm) and green (530 nm) diodes, respectively. However, blue diodes (430 nm) have only been able to achieve maximum external quantum efficiencies in the range of 4% to 6%.

- Water damage: Water can damage the organic materials of the displays. Therefore, improved sealing processes are important for practical manufacturing. Water damage may especially limit the longevity of more flexible displays.
- Outdoor performance: As an emissive display technology, OLEDs rely completely upon converting electricity to light, unlike most LCDs which are to some extent reflective; e-ink leads the way in efficiency with ~ 33% ambient light reflectivity, enabling the display to be used without any internal light source. The metallic cathode in an OLED acts as a mirror, with reflectance approaching 80%, leading to poor readability in bright ambient light such as outdoors. However, with the proper application of a circular polarizer and anti-reflective coatings, the diffuse reflectance can be reduced to less than 0.1%. With 10,000 fc incident illumination (typical test condition for simulating outdoor illumination), that yields an approximate photopic contrast of 5:1.
- Power consumption: While an OLED will consume around 40% of the power of an LCD displaying an image which is primarily black, for the majority of images it will consume 60–80% of the power of an LCD – however it can use over three times as much power to display an image with a white background such as a document or website. This can lead to reduced real-world battery life in mobile devices.
- Screen burn-in: Unlike displays with a common light source, the brightness of each OLED pixel fades depending on the content displayed. The varied lifespan of the organic dyes can cause a discrepancy between red, green, and blue intensity. This leads to image persistence, also known as burn-in.
- UV sensitivity: OLED displays can be damaged by prolonged exposure to UV light. The most pronounced example of this can be seen with a near UV laser (such as a Blu-ray pointer) and can damage the display almost instantly with more than 20 mW leading to dim or dead spots where the beam is focused. This is usually avoided by installing a UV blocking filter over the panel and this can easily be seen as a clear plastic layer on the glass. Removal of this filter can lead to severe damage and an unusable display after only a few months of room light exposure.

e. 3D display

A 3D display is any display device capable of conveying a stereoscopic perception of 3-D depth to the viewer. The basic requirement is to present offset images that are displayed separately to the left and right eye. Both of these 2-D offset images are then combined in the brain to give the perception of 3-D depth. Although the term "3D" is ubiquitously used, it is important to note that the presentation of dual 2-D images is distinctly different from displaying an image in three full dimensions. The most notable difference is that the observer is lacking any freedom of head movement and freedom to increase information about the 3-dimensional objects being displayed. Holographic displays do not have this limitation, so the term "3D display" fits accurately for such technology.

Similar to how in sound reproduction it is not possible to recreate a full 3-dimensional sound field merely with two stereophonic speakers, it is likewise an overstatement of capability to refer to dual 2-D images as being "3D". The accurate term "stereoscopic" is more cumbersome than the common misnomer "3D", which has been entrenched after many decades of unquestioned misuse.

The optical principles of multiview auto-stereoscopy have been known for over 60 years. Practical displays with a high resolution have recently become available commercially.

Types of 3D displays

- Stereoscopic - Based on the principles of stereopsis, described by Sir Charles Wheatstone in the 1830s, stereoscopic technology provides a different image to the viewer's left and right eyes. Examples of this technology include anaglyph images and polarized glasses. Stereoscopic technologies generally involve special spectacles.
- Autostereoscopic - Autostereoscopic display technologies use optical components in the display, rather than worn by the user, to enable each eye to see a different image. The optics split the images directionally into the viewer's eyes, so the display viewing geometry requires limited head positions that will achieve the stereoscopic effect. Automultiscopic displays provide multiple views of the same scene, rather than just two. Each view is visible from a different range of positions in front of the display. This allows the viewer to move left-right in front of the display and see the correct view from any position. Example technologies include parallax barriers and specular holography.

- Computer-generated holography - Research into holographic displays has produced devices which are able to create a light field identical to that which would emanate from the original scene, with both horizontal and vertical parallax across a large range of viewing angles. The effect is similar to looking through a window at the scene being reproduced; this may make CGH the most convincing of the 3D display technologies, but as yet the large amounts of calculation required to generate a detailed hologram largely prevent its application outside of the laboratory.
- Volumetric displays - Volumetric displays use some physical mechanism to display points of light within a volume. Such displays use voxels instead of pixels. Volumetric displays include multiplanar displays, which have multiple display planes stacked up, and rotating panel displays, where a rotating panel sweeps out a volume.
- Other technologies have been developed to project light dots in the air above a device. An infrared laser is focused on the destination in space, generating a small bubble of plasma which emits visible light.

Hard-Copy Devices

We can obtain hard-copy output for our images in several formats. For presentations or archiving, we can send image files to devices or service bureaus that will produce 35-mm slides or overhead transparencies. To put images on film, we can simply photograph a scene displayed on a video monitor. And we can put our pictures on paper by directing graphics output to a printer or plotter. The quality of the pictures obtained from a device depends on dot size and the number of dots per inch, or Lines per inch, that can be displayed. To produce smooth characters in printed text strings, higher-quality printers shift dot positions so that adjacent dots overlap.

Printers produce output by either impact or nonimpact methods. Impact printer's press formed character faces against an inked ribbon onto the paper. A line printer is an example of an impact device, with the typefaces mounted on bands, chains, drums, or wheels. Nonimpact printers and plotters use laser techniques, ink-jet sprays, xerographic presses' (as used in photocopying machines),electrostatic methods, and electro thermal methods to get images onto Paper.

Character impact printers often have a dot-matrix print head containing a rectangular array of protruding wire pins, with the number of pins depending on the quality of the printer. Individual characters or graphics patterns are obtained by retracting certain pins so that the remaining pins form the pattern to be printed.

In a laser device, a laser beam makes a charge distribution on a rotating drum coated with a photoelectric material, such as selenium. Toner is applied to the drum and then transferred to paper.

Ink-jet methods produce output by squirting ink in horizontal rows across a roll of paper wrapped on a drum. The electrically charged ink stream is deflected by an electric field to produce dot-matrix patterns. Drafting layouts and other drawings are typically generated with ink-jet or pen plotters. A pen plotter has one or more pens mounted on a carriage, or crossbar, that spans a sheet of paper. Pens with varying colors and widths are used to produce a variety of shadings and line styles. Wet-ink, ball-point, and felt-tip pens are all possible choices for use with a pen plotter. Plotter paper can lie flat or be rolled onto a drum or belt. Crossbars can be either moveable or stationary, while the pen moves back and forth along the bar. Clamps, a vacuum, or an electrostatic charge hold the paper in position.

Chapter 4: Output Primitives

Graphic SW and HW provide subroutines to describe a scene in terms of basic geometric structures called output primitives. Those functions in a graphic package that we use to describe the various picture components are called the graphic output primitives or simply primitives. Output primitives are combined to form complex structures. Output primitives describe the geometry of objects and – typically referred to as geometric primitives. Examples: point, line, text, filled region, images, quadric surfaces, spline curves. Each of the output primitives has its own set of attributes.

Graphics Definitions

Point - A location in space, 2D or 3D. Sometimes denotes one pixel

Vertex - Point in 3D

Edge - Line in 3D connecting two vertices

Polygon/Face/Facet - Arbitrary shape formed by connected vertices. Fundamental unit of 3D computer graphics

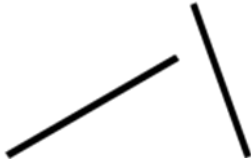
Output primitives attributes

Points

Attributes: Size, Color

Lines

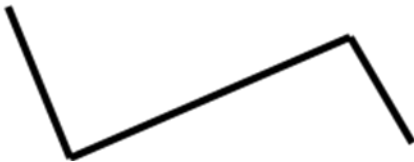
Attributes: Color, Thickness, Type



Polylines (open)

A set of line segments joined end to end.

Attributes: Color, Thickness, Type



Polylines (closed)

A polyline with the last point connected to the first point.

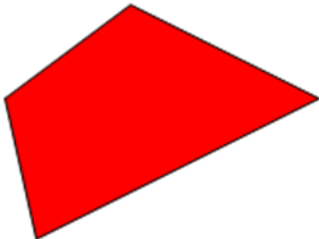
Attributes: Color, Thickness, Type



Polygons

A set of line segments joined end to end.

Attributes: Fill color, Thickness, Fill pattern



Text

Attributes: Font, Color, Size, Spacing, Orientation.

Font:

- Type (Helvetica, Times, Courier etc.)
- Size (10 pt, 14 pt etc.)
- Style (**Bold**, *Italic*, Underlined)

Images

Attributes: Image Size, Image Type, Color Depth.

Image Type:

- Binary (only two levels)
- Monochrome
- Color.



Color Depth:

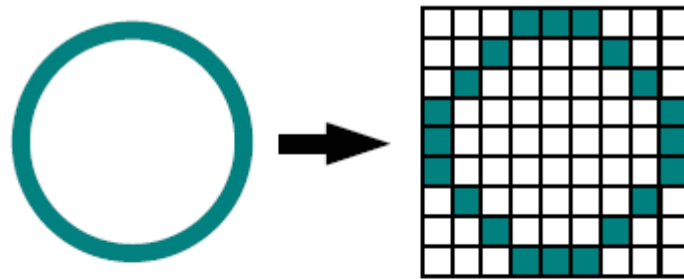
- Number of bits used to represent color.

Scan Conversion

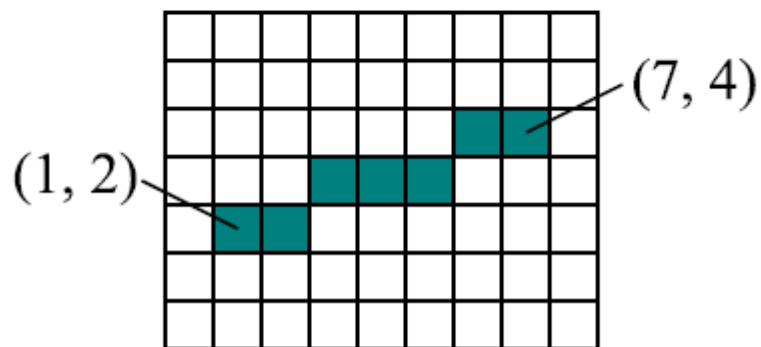
Process of converting output primitives into frame buffer updates. Choose which pixels contain which intensity value.

Constraints

- Straight lines should appear as a straight line
- primitives should start and end accurately
- Primitives should have a consistent brightness along their length
- They should be drawn rapidly



Scan conversion of a circle



Point scan conversion

```
#include <iostream>
#include<conio.h>
#include <windows.h>
void gotoxy(int x, int y)
{
    COORD ord;
    ord.X = x;
    ord.Y = y;
    SetConsoleCursorPosition
(GetStdHandle(STD_OUTPUT_HANDLE), ord);
}
void point(int x, int y)
{
    gotoxy(x,y);
    std::cout<<"x";
}
int main()
{
    point(2,2),
    getch();
}
```

Line Drawing

What is a Line

- Straight path connecting two points.
- Extremely small width, consistent density.
- Beginning and end on points.

A line, or straight line is, (infinitely) thin, (infinitely) long, straight geometrical object, i.e. a curve that is long and straight. Line is an infinitely-extending one-dimensional figure that has no curvature. A line segment is a part of a line that is bounded by two distinct end points and contains every point on the line between its end points.

What is an ideal line?

Must appear straight and continuous.

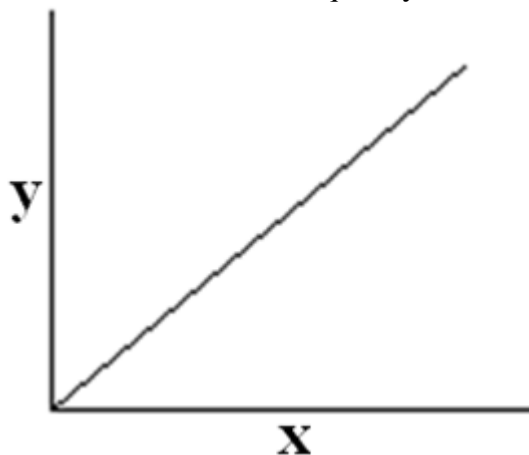
Only possible axis-aligned and 45 degree lines.

Must interpolate (interrupt) both defining end points.

Must have uniform density and intensity.

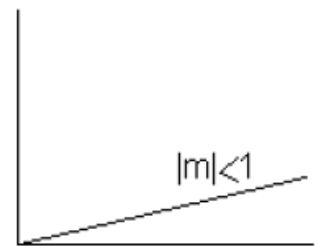
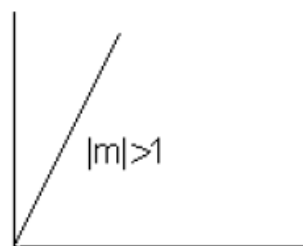
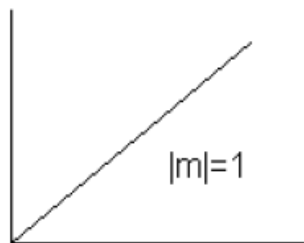
Consistent within a line and over all lines.

Must be efficient, drawn quickly.



A straight line segment is defined by the coordinate positions for the endpoints of the segment. Given two points, in Euclidean geometry, one can always find exactly one line that passes through the two points. A line may have three forms with respect to slope:

- Slope=1
- Slope>1
- Slope<1



Line Equations

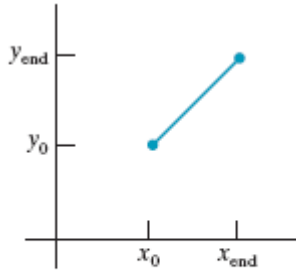
The Cartesian

slope-intercept equation for the straight line is

$$y = m.x + b \text{-----(1)}$$

Where m=Slope of the line b= Constant

Our aim is to scan convert a line segment usually defined by its endpoints: (x_0, y_0) and (x_{end}, y_{end}) .



Line path
between endpoint positions
 (x_0, y_0) and (x_{end}, y_{end}) .

Therefore, we need to draw the line segment: $y = m.x + b$

For $x_0 \leq x \leq x_{end}$ and $y_0 \leq y \leq y_{end}$

Where

$$m = (y_{end} - y_0) / (x_{end} - x_0) = \delta y / \delta x \text{-----(2)}$$

At point (x_0, y_0)

$$b = y_0 - m \cdot x_0$$

$$= y_0 - x_0 \cdot (y_{end} - y_0) / (x_{end} - x_0) \text{-----(3)}$$

At point (x_{end}, y_{end})

$$b = y_{end} - m \cdot x_{end}$$

$$= y_{end} - x_{end} \cdot (y_{end} - y_0) / (x_{end} - x_0) \text{-----(4)}$$

From Eq.2, for any given x interval δx along a line, we can compute the corresponding y interval δy .

$$\delta y = m \cdot \delta x \text{-----(5)}$$

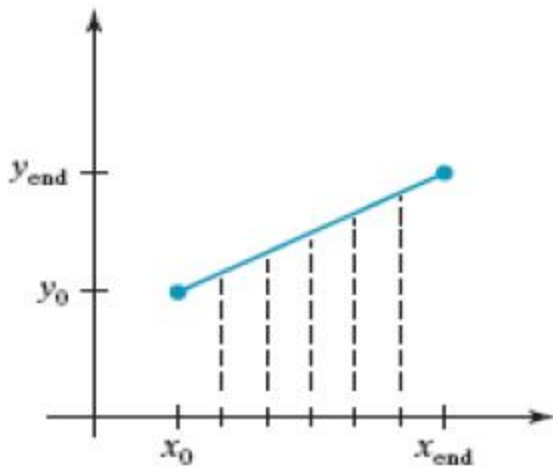
Similarly for any given y interval δy along a line, we can compute the corresponding x interval δx .

$$\delta x = \delta y / m \text{-----(6)}$$

Does it Work?

Now check if $|m| < 1$ then starting at the first point, simply increment x by 1 (unit increment) till it reaches ending point; whereas calculate y point by the equation 5 ($\delta y = m \cdot \delta x$) for each x.

Conversely if $|m| > 1$ then increment y by 1 till it reaches ending point; whereas calculate x point corresponding to each y, by the equation 6 ($\delta x = \delta y / m$).



Straight-line
segment with five sampling
positions along the x axis
between x_0 and x_{end} .

If $|m|$ is less than 1 ($|m| < 1$) Then it means that for every subsequent pixel on the line there will be unit increment in x direction and there will be less than 1 increment in y direction and vice versa for slope greater than 1 ($|m| > 1$).

If $|m|=1$ then $\delta x = \delta y$. In this case a smooth line with slope of m is generated between the specified points.

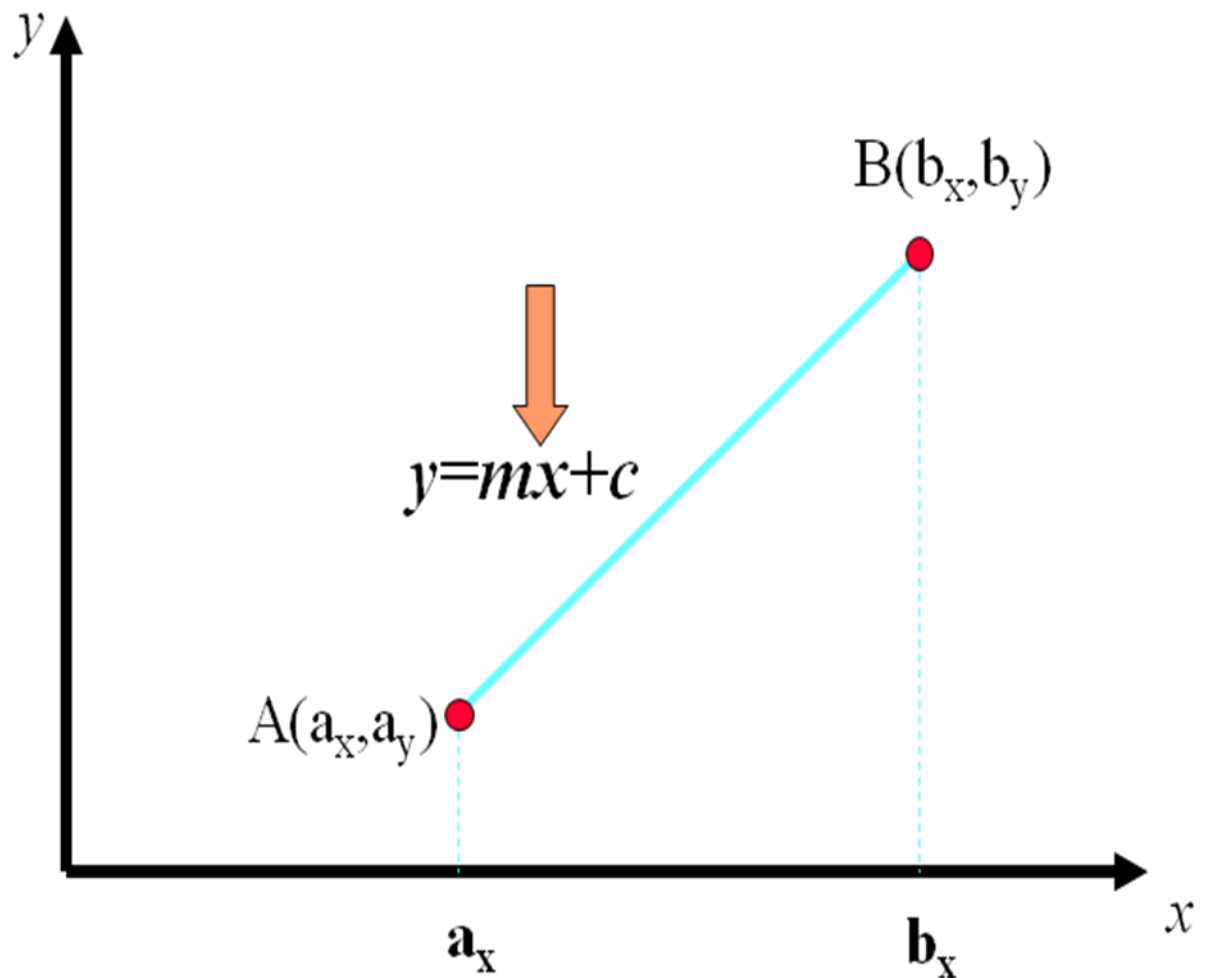
Line Drawing Algorithms

- Line drawing is fundamental to computer graphics.
- We must have fast and efficient line drawing functions.
- Rasterization Problem: Given only the two end points, how to compute the intermediate pixels, so that the set of pixels closely approximate the ideal line.



Analytical Method





Directly based on the analytical equation of a line. Involves floating point multiplication and addition. Requires round-off function.

```
#include <iostream>
#include <windows.h>
#include <conio.h>
void gotoxy(int x, int y)
{
    COORD ord;
    ord.X = x;
    ord.Y = y;
    SetConsoleCursorPosition
(GetStdHandle(STD_OUTPUT_HANDLE), ord);
}
void Draw_Line(int xo, int yo,
int xe, int ye)
{
```

```

    double m = ((double) ye - yo) /
((double) xe - xo);
if(m<1)
{
    for(int x = xo; x <= xe; x++)
    {
        double y = m * (x - xo) + yo;
        int yi=static_cast<int>(y);
        gotoxy(x, yi);
        std::cout << "x";


    }
}
else
{
    for(int y = yo; y <= ye; y++)
    {
        double x=((y-yo)+xo)/m;
        int xi=static_cast<int>(x);
        gotoxy(xi, y);
        std::cout << "x";
    }
}
}
int main()
{
    Draw_Line(0,0,6,3);
    Draw_Line(10,10,15,20);
    gotoxy(0,0);
    getch();
}

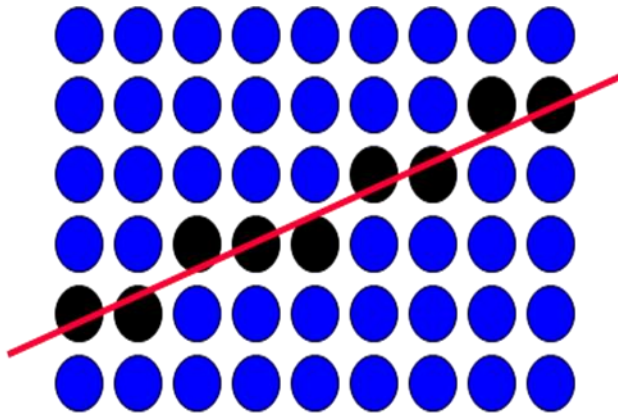
```

Incremental Algorithms

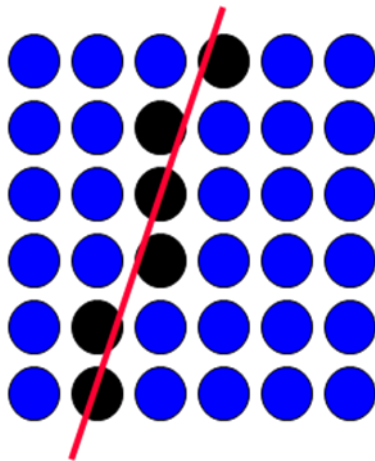
I have got a pixel on the line (Current Pixel). How do I get the next pixel on the line?

Compute one point based on the previous point:

(x0, y0).....(xk, yk)  (xk+1, yk+1)

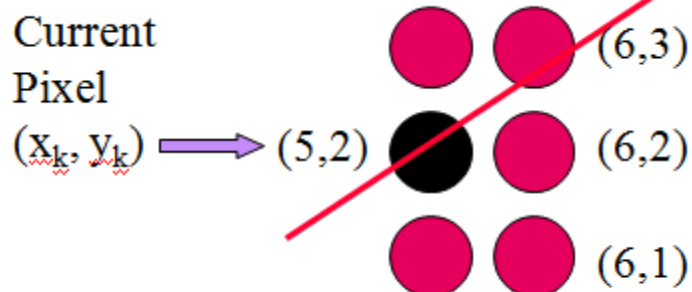


**Next pixel on next column
(when slope is small)**



**Next pixel on next row
(when slope is large)**

Incrementing along x



To find (x_{k+1}, y_{k+1}) :

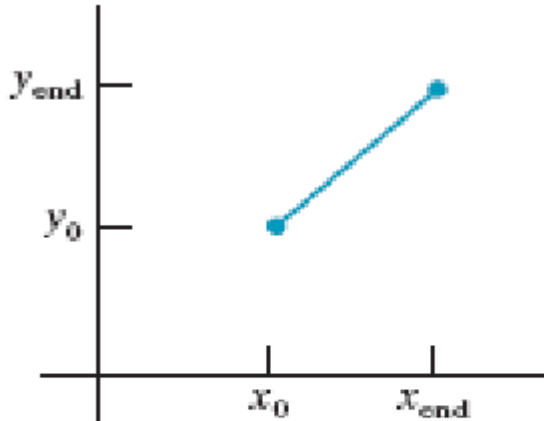
$$x_{k+1} = x_k + 1$$

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

Assumes that the next pixel to be set is on the next column of pixels (Incrementing the value of x !). Not valid if slope of the line is large.

Digital Differential Analyzer (DDA) Algorithm

Digital Differential Analyzer Algorithm is an incremental algorithm. The basis of the DDA method 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 .



Line path
between endpoint positions
(x_0, y_0) and (x_{end}, y_{end}).

As we know that

- $m = \delta y / \delta x$
- $\delta y = m \delta x$
- $\delta x = 1/m \cdot \delta y = \delta y / m$

We consider first a line with positive slope, as shown in figure above (For 1st quadrant). If the slope is less than or equal to 1 ($m \leq 1$), we sample at unit x intervals ($\delta x = 1$) and compute successive y values as

$$y_{k+1} - y_k = m \quad (\text{If } \delta y = y_{k+1} - y_k \text{ \& } \delta x = 1)$$

$$y_{k+1} = y_k + m \text{ -----(1)}$$

Subscript k takes integer values starting from 0, for the first point, and increases by 1 until the final endpoint is reached. Since m can be any real number between 0.0 and 1.0, each calculated y value must be rounded to the nearest integer corresponding to a screen pixel position in the x column we are processing. For lines with a positive slope greater than 1.0 ($m > 1$), we reverse the roles of x and y .

That is, we sample at unit y intervals ($\delta y = 1$) and calculate consecutive x values as:

- $\delta x = 1/m$ (If $\delta x = x_{k+1} - x_k$ \& $\delta y = 1$)
- $x_{k+1} - x_k = 1/m$
- $x_{k+1} = x_k + 1/m \text{ -----(2)}$

In this case, each computed x value is rounded to the nearest pixel position along the current y scan line. Equations 1 and 2 are based on the assumption that lines are to be processed from the left endpoint to the right endpoint of the figure above.

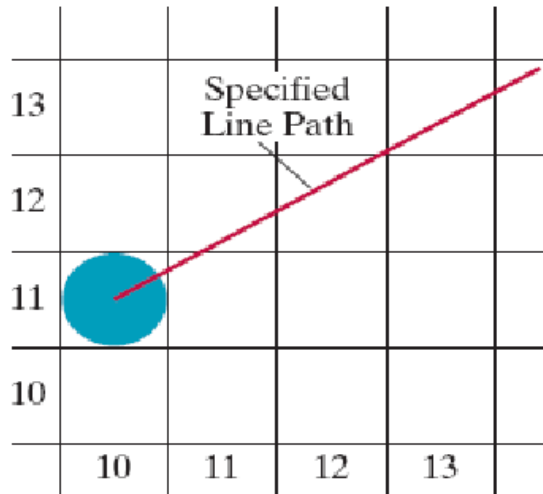
If this processing is reversed, so that the starting endpoint is at the right (4th Quadrant), then

We have $\delta x = -1$ and

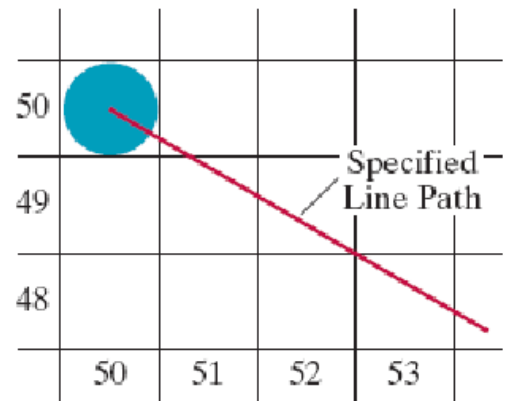
$$y_{k+1} = y_k - m \text{ -----(3)}$$

When the slope is greater than 1, i.e. $m > 1$, we have $\delta y = -1$ and

$$x_{k+1} = x_k - 1/m \text{ -----(4)}$$



A section of a display screen where a straight-line segment is to be plotted, starting from the pixel at column 10 on scan line 11.



A section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

Similar calculations are carried out using equations 1 through 4 to determine pixel positions along a line with negative slope.

- Thus, if the $|m| < 1$ and the starting endpoint is at the left, we set $\delta x = 1$ and calculate y values with Eq#1.
- When the starting endpoint is at the right (for the same slope i.e. $|m| < 1$), we set $\delta x = -1$ and obtain y positions using Eq#3.
- For a negative slope with $|m| > 1$, we use $\delta y = -1$ and Eq#4 or
- We use $\delta y = 1$ and Eq#2.

DDA has very simple technique.

1) Find difference δx and δy between x coordinates and y coordinates respectively ending points of a line.

2) If $|\delta x|$ is greater than $|\delta y|$, then $|\delta x|$ will be step; otherwise $|\delta y|$ will be step.

if $|\delta x| > |\delta y|$ then

step = $|\delta x|$

else

step = $|\delta y|$

Now very simple to say that step is the total number of pixel required for a line.

3) Next step is to divide dx and dy by step to get x Increment and y Increment that is the increment required in each step to find next pixel value.

$xIncrement = dx/step$

$yIncrement = dy/step$

4) Next a loop is required that will run step times.

- In the loop drawPixel and add x Increment in x1 and y Increment in y1.
- To sum-up all above in the algorithm, we will get;


```

dx= xEnd-x0           // find the difference of xEnd and x0
dy= yEnd-y0           // find the difference of yEnd and y0
x1 = x0                // set initial value of x1
y1 = y0                // set initial value of y1
if |dx|>|dy| then      // find step
  step = |dx|
else
  step = |dy|
xIncrement= dx/step     // calculate xIncrement
yIncrement= dy/step     // calculate yIncrement
for counter = 1 to step // loop to draw pixel
  Begin
  draw_Pixel(round(x1), round(y1)) // function to draw pixel
  x1 = x1 + xIncrement           // Increment x1
  y1 = y1 + yIncrement           // Increment y1
      
```

Criticism on Algorithm

- There is serious criticism on the algorithm that is use of floating point calculation.
- They say that when we have to draw points that should have integers as coordinates then why to use floating point calculation, which requires more space as well as they have more computational cost.
- Therefore there is need to develop an algorithm which would be based on integer type calculations.

C++ code implementation of Digital Differential Analyzer (DDA) Algorithm

```

#include <iostream>
#include <windows.h>
#include <conio.h>
void gotoxy(int x, int y)
{
    COORD ord;
    ord.X = x;
    ord.Y = y;
    SetConsoleCursorPosition
(GetStdHandle(STD_OUTPUT_HANDLE), ord);
}
void Draw_Line(int xo, int yo,
int xe, int ye)
{
    gotoxy(xo, yo);
    std::cout << "x";

```

```

        double yk=yo;
        double xk=xo;

        double m = ((double) ye - yo) /
        ((double) xe - xo);

        if(m<1)
        {
            for(int x = (xo+1); x <= xe; x++)
            {
                double yk1 =yk+m ;

                int iyk1i=static_cast<int>(yk1);
                yk=yk1;
                gotoxy(x, iyk1i);
                std::cout << "x";
            }
        }
        else
        {
            for(int y = (yo+1); y <= ye; y++)
            {
                double xk1=(xk+(1/m));
                int ixk1i=static_cast<int>(xk1);
                xk=xk1;
                gotoxy(ixk1i, y);
                std::cout << "x";

            }
        }
    }
}

```

```

int main()
{
    Draw_Line(0,0,6,3);
    Draw_Line(10,10,15,20);
    gotoxy(0,0);
    getch();
}

```

Bresenham Line Drawing Algorithm

The Bresenham line algorithm is an algorithm that determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition and subtraction all of which are very economical

operations in standard computer architectures. Bresenham's algorithm finds the closest integer coordinates to the actual line, using only integer math. Assuming that the slope is positive and less than 1, moving 1 step in the x direction,

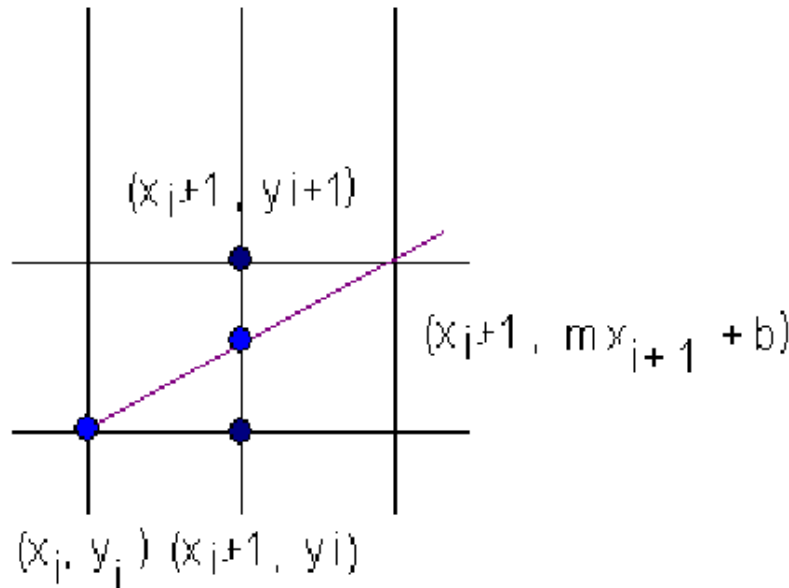
- y either stays the same, or
- Increases by 1.

A decision function is required to resolve this choice.

If the current point is (x_i, y_i) , the next point can be either (x_i+1, y_i) or (x_i+1, y_i+1) .

The actual position on the line is

$(x_i+1, m(x_i+1)+b)$

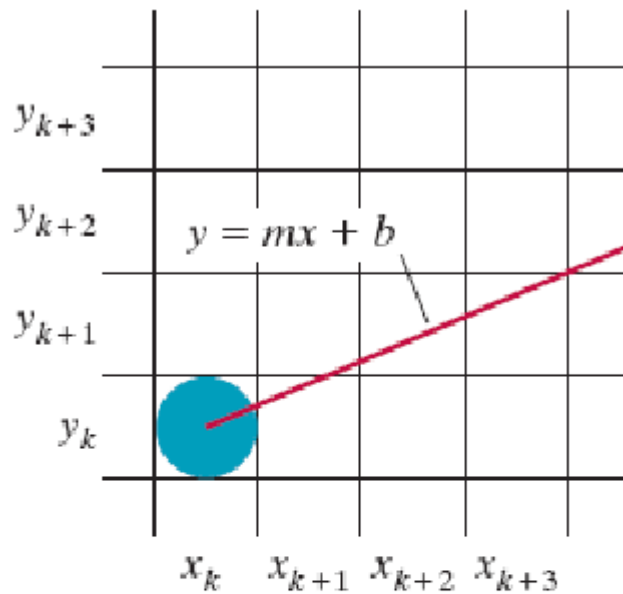


To illustrate Bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.0. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x_0, y_0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.

Figure below demonstrates the k^{th} step in this process. Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column $x_{k+1} = x_k + 1$?-----(1)

Our choices are the pixels at positions

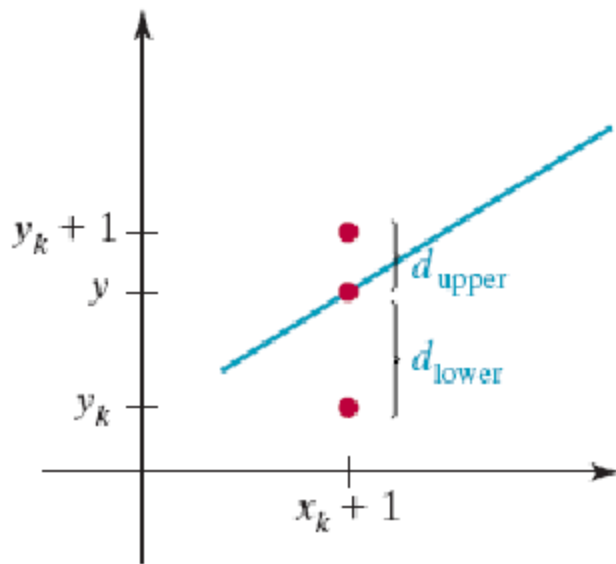
- (x_k+1, y_k) and
- (x_k+1, y_k+1) .



A section of the screen showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < m < 1$.

At sampling position x_{k+1} , we label vertical pixel separations from the mathematical line path as d_{lower} and d_{upper} . The y coordinate on the mathematical line at pixel column position x_{k+1} is calculated as:

- $y = mx + b$
- $y = m(x_{k+1}) + b$
- {Replace x by x_{k+1} }



Vertical
distances between pixel
positions and the line y
coordinate at sampling
position $x_k + 1$.

Then

$$d_{\text{lower}} = y - y_k \\ = m(x_k + 1) + b - y_k \text{-----}(2)$$

{ Replace y by $m(x_k + 1) + b$ }

And

$$d_{\text{upper}} = y_{k+1} - y \\ = y_{k+1} - [m(x_k + 1) + b] \\ \text{{ Replace } y \text{ by } m(x_k + 1) + b \text{ } } \\ = y_{k+1} - m(x_k + 1) - b \text{-----}(3)$$

To find which pixel is closest???, a test is performed that is based on the difference of both separations. i.e.; $d_{\text{lower}} - d_{\text{upper}}$

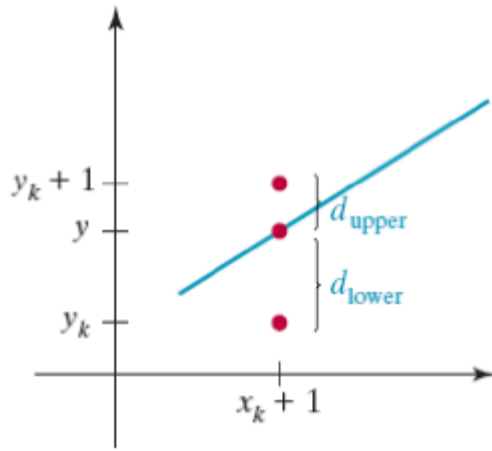


FIGURE 3-11 Vertical distances between pixel positions and the line y coordinate at sampling position $x_k + 1$.

So subtracting Eq#3 from Eq#2.

$$\begin{aligned} d_{\text{lower}} - d_{\text{upper}} &= m(x_k + 1) + b - y_k - [y_k + 1 - m(x_k + 1) - b] \\ &= m(x_k + 1) + b - y_k - y_k - 1 + m(x_k + 1) + b \\ &= 2m(x_k + 1) - 2y_k + 2b - 1 \end{aligned} \quad \text{-----(4)}$$

Decision parameter P_k for the k^{th} step can be obtained by re-arranging Eq#4.

$$\begin{aligned} d_{\text{lower}} - d_{\text{upper}} &= 2m(x_k + 1) - 2y_k + (2b - 1) \\ &= 2 \Delta y / \Delta x (x_k + 1) - 2y_k + (2b - 1) \\ \Delta x (d_{\text{lower}} - d_{\text{upper}}) &= 2 \Delta y (x_k + 1) - 2 \Delta x y_k + \Delta x (2b - 1) \end{aligned} \quad \text{-----(5)}$$

As $\Delta x (d_{\text{lower}} - d_{\text{upper}}) = P_k$, Eq#5 becomes:

$$\begin{aligned} P_k &= 2 \Delta y (x_k + 1) - 2 \Delta x y_k + \Delta x (2b - 1) \\ P_k &= 2 \Delta y x_k + 2 \Delta y - 2 \Delta x y_k + \Delta x (2b - 1) \\ P_k &= 2 \Delta y x_k - 2 \Delta x y_k + 2 \Delta y + \Delta x (2b - 1) \end{aligned} \quad \text{-----(6)}$$

As $+ 2 \Delta y + \Delta x (2b - 1)$ is a constant term, so Eq#6 becomes:

$$P_k = 2 \Delta y x_k - 2 \Delta x y_k + C \quad \text{-----(7)}$$

Note that Δx and Δy are independent of pixel positions, so are considered as Constant.

Coordinate changes along the line occurs in unit step in either x or y directions. Therefore we can obtain the value of successive decision parameters using incremental integer calculations.

At step $k+1$, the decision parameter P_{k+1} can be calculated from Eq#7 as:

$$P_{k+1} = 2 \Delta y x_{k+1} - 2 \Delta x y_{k+1} + C \quad \text{-----(8)}$$

Perform subtraction of Eq#7 from Eq#8.

$$P_{k+1} - P_k = 2 \Delta y x_{k+1} - 2 \Delta x y_{k+1} + C - [2 \Delta y x_k - 2 \Delta x y_k + C]$$

$$P_{k+1} - P_k = 2 \Delta y x_{k+1} - 2 \Delta x y_{k+1} + C - 2 \Delta y x_k + 2 \Delta x y_k - C$$

$$P_{k+1} - P_k = 2 \Delta y x_{k+1} - 2 \Delta y x_k - 2 \Delta x y_{k+1} + 2 \Delta x y_k$$

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

And

$$P_{k+1} = P_k + 2 \Delta y (x_{k+1} - x_k) - 2 \Delta x (y_{k+1} - y_k) \text{ -----(9)}$$

But from Eq#1, $x_{k+1} = x_k + 1$

So Eq#9 becomes:

$$P_{k+1} = P_k + 2 \Delta y (x_k + 1 - x_k) - 2 \Delta x (y_{k+1} - y_k) \text{ -----(10)}$$

Where $y_{k+1} - y_k$ is either 0 or 1.

So

$$P_{k+1} = P_k + 2 \Delta y \text{ -----(11)}$$

For $y_{k+1} - y_k = 0$ and

Point is $(x_k + 1, y_k)$.

And

$$P_{k+1} = P_k + 2 \Delta y - 2 \Delta x \text{ -----(12)}$$

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

Point is $(x_k + 1, y_k + 1)$.

How to calculate the 1st decision parameter P_0 ???

Consider Eq#6.

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

$$P_k = 2 \Delta y (x_k + 1) - 2 \Delta x y_k + \Delta x (2b - 1) \text{ ----[A]}$$

For P_0 , use point/pixel (x_1, y_1) , so Eq#A becomes:

$$P_0 = 2 \Delta y (x_1 + 1) - 2 \Delta x y_1 + \Delta x (2b - 1) \text{ ----[B]}$$

As $b = y - mx$

$b = y_1 - mx_1$ for (x_1, y_1)

So Eq#B changes as

$$P_0 = 2 \Delta y x_1 + 2 \Delta y - 2 \Delta x y_1 + \Delta x [2(y_1 - mx_1) - 1]$$

$$P_0 = 2 \Delta y x_1 + 2 \Delta y - 2 \Delta x y_1 + \Delta x (2y_1 - 2mx_1 - 1) \text{ -----[C]}$$

But $m = \Delta y / \Delta x$

So Eq#C becomes

$$P_0 = 2 \Delta y x_1 + 2 \Delta y - 2 \Delta x y_1 + \Delta x (2y_1 - 2(\Delta y / \Delta x)x_1 - 1)$$

$$P_0 = 2 \Delta y x_1 + 2 \Delta y - 2 \Delta x y_1 + 2 \Delta x y_1 - 2 \Delta y x_1 - \Delta x$$

$$P_0 = 2 \Delta y - \Delta x \text{ -----[D]}$$

Bresenham Algorithm

BresenhamLine(int x1, y1, x2, y2)

//Line drawing procedure for

$m < 1$

Begin

//Start of Line drawing procedure

int dx = x2 - x1

//Calculate dx

int dy = y2 - y1

//Calculate dy

int p = 2 * dy - dx

//Calculate initial decision parameter

int increE = 2 * dy

//for $p \leq 0$

int incrNE = 2 * (dy - dx)

//for $p = 1$

if ($x_1 > x_2$)

//Determine the start points

Begin

//Start of if body

$x = x_2$

$y = y_2$

$x_2 = x_1$

End

//End of if body

else

```

Begin                                     //Start of else body
x=x1
y=y1
End                                     //End of else body
Plot_Pixel(x, y)                         //Function to draw pixel
while (x < x2)
Begin                                   //Start of loop body
if (p <= 0)
Begin                                   //Start of if body
p+=incrE
x++
End                                     //End of if body
else
Begin                                   //Start of else body
p+=incrNE
x++
y++
End                                     //End of else body
Plot_Pixel(x, y)                         //Function to draw pixel
End                                     //End of loop
End                                     //End of Line drawing procedure

```

C++ implementation for Bresenhams' line drawing Algorithm

```

#include <iostream>
#include <conio.h>
#include <windows.h>
void gotoxy(int x, int y)
{
COORD ord;
ord.X = x;
ord.Y = y;
SetConsoleCursorPosition(GetStdHandle
(STD_OUTPUT_HANDLE), ord);
}

```

```

void Bresenham_Line(int left, int top,
int right, int bottom)
{

```

```

    int x, y;
    int Cx=right-left;
    int Cy=bottom-top;
    int twoCy=2*Cy;
    int twoCx=2*Cx;
    int twoCxy=twoCy-twoCx;

```



```

    int P;

    gotoxy(left, top);std::cout << "i";

    P=twoCy-Cx;

    if(P<0)
    { x=left+1; y=top;}
    else
    { x=left+1; y=top+1;}

    gotoxy(x, y);std::cout << "a";


    int stop=Cx+x-1;

    for(int X = x; X < stop; X++)
    {
        if(P<0)
        {
            P=P+twoCy;
            x=x+1; y=y;
        }
        else
        {
            x=x+1; y=y+1;
            P=P+twoCxy;
        }

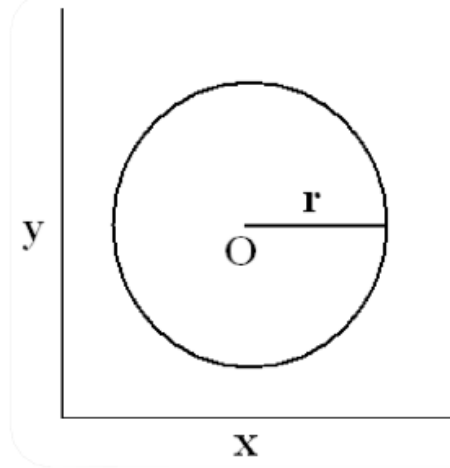
        gotoxy(x, y);
        std::cout << "x";

    }
}

int main()
{
    Bresenham_Line(0,0,10,6);
    gotoxy(100,100);
    getch();
}
Circle Drawing Algorithms
What is Circle?

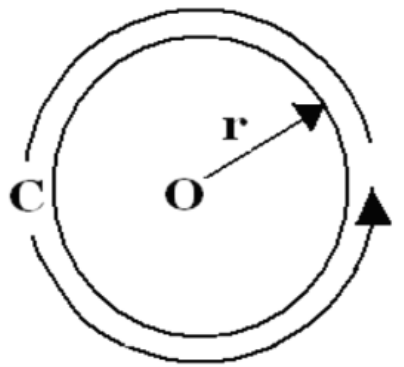
```

A circle is the set of points in a plane that are equidistant from a given point O. The distance r from the center is called the radius, and the point O is called the center. Twice the radius is known as the diameter. The angle a circle subtends from its center is a full angle, equal to 360° or 2π radians.



Circumference of Circle

A circle has the maximum possible area for a given perimeter, and the minimum possible perimeter for a given area. The perimeter C of a circle is called the circumference, and is given by $C = 2\pi r$



Circle Drawing Techniques

First of all for circle drawing we need to know what the input will be. Well, the input will be one center point (x, y) and radius r. Therefore, using these two inputs there are a number of ways to draw a circle. They involve understanding level very simple to complex and reversely time complexity inefficient to efficient. We see them one by one giving comparative study.

Circle Drawing Using Cartesian Coordinates

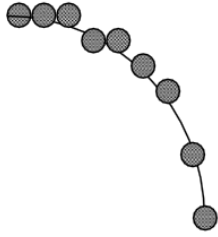
This technique uses the equation for a circle with radius r centered at (0,0):

$$x^2 + y^2 = r^2 \text{-----(I)}$$

An obvious choice is to plot

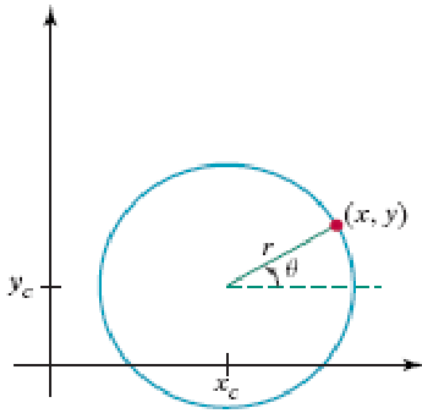
$$y = \pm \sqrt{r^2 - x^2} \quad \text{Against different values of } x.$$

Using above equation a circle can be easily drawn.



The value of x varies from $r - x_c$ to $r + x_c$, and y is calculated using the formula given below.

$$y = \pm \sqrt{r^2 - x^2}$$



Circle with center coordinates (x_c, y_c) and radius r .

Eq#I for points (x, y) and (x_c, y_c) can be modified as:

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \text{-----(II)}$$

To calculate y :

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

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

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

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

Algorithm

Using this technique a simple algorithm will be:

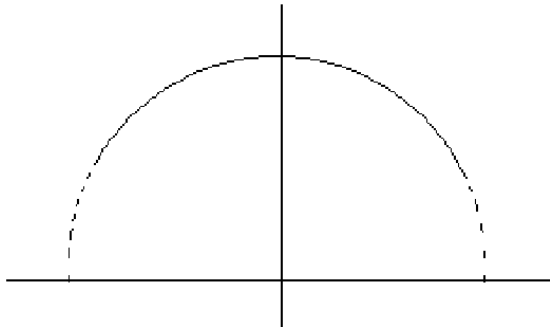
```

Circle1 (xc, yc, r)           // procedure to draw circle
Begin
  for x= r - xc to r + xc
    Begin
      y = yc +  $\sqrt{r^2 - (x - xc)^2}$ 
      drawPixel (x, y)
      y = yc -  $\sqrt{r^2 - (x - xc)^2}$ 
      drawPixel (x, y)
      x++
    End
  End
End

```

Drawbacks/ Shortcomings

- This works, but is inefficient because of the multiplications and square root operations.
- It also creates large gaps in the circle for values of x close to r as shown in the figure.



Circle Drawing Using Polar Coordinates

A better approach, to eliminate unequal spacing as shown in previous figure is to calculate points along the circular boundary using polar coordinates r and θ . Expressing the circle equation in parametric polar form yields the pair of equations

$$\left. \begin{aligned} x &= x_c + r \cos \theta \\ y &= y_c + r \sin \theta \end{aligned} \right\} \text{-----(I)}$$

Using Eq#I circle can be plotted by calculating x and y coordinates as θ takes values from 0 to 360 degrees or 0 to 2π .

When these equations are using a fixed angular step size, a circle is plotted with equidistant points on the circumference. The step size chosen for θ depends on the application and the display device. Larger angular separations along the circumference can be connected with straight-line segments to approximate the circular path. For a more continuous boundary on a raster display, we can set the step size at $1/r$. This plots pixel positions that are approximately one unit apart.

Algorithm

Now let us see how this technique can be sum up in algorithmic form.

```

Circle2 (xc, yc, r)      // procedure to draw circle
Begin
  for  $\theta = 0$  to  $2\pi$  step  $1/r$ 
  Begin
     $x = x_c + r * \cos \theta$ 
     $y = y_c + r * \sin \theta$ 
    drawPixel (x, y)
  End
End

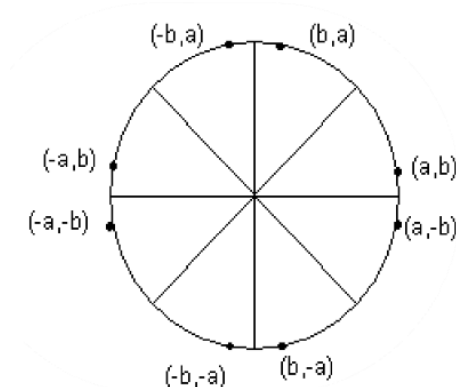
```

Again this is very simple technique. And also it solves problem of unequal space. But unfortunately this technique is still inefficient in terms of calculations because it involves

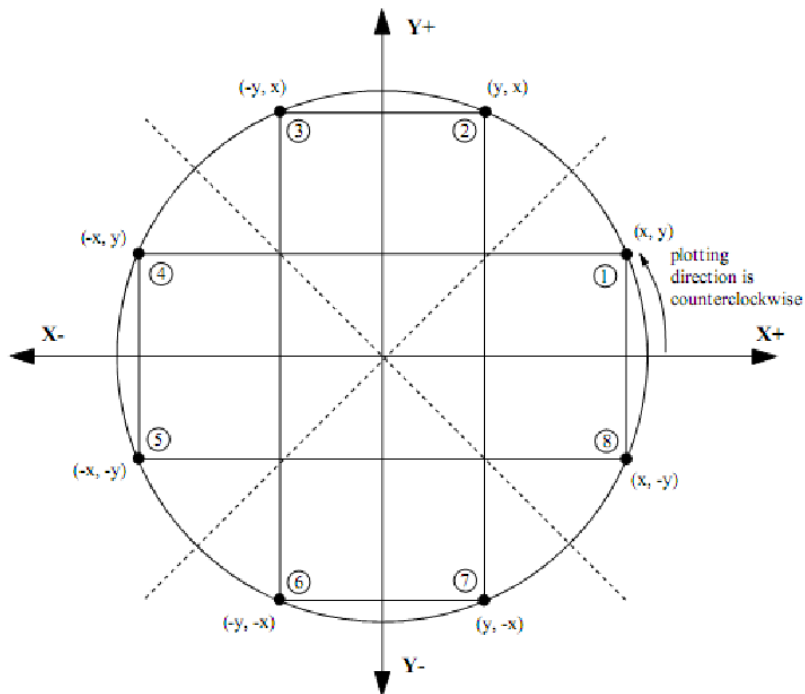
- Floating point calculations and
 - Trigonometric calculations
- both of which are time consuming.

Calculations can be reduced by considering the symmetry of circles.

The shape of circle is similar in each quadrant. We can generate the circle section in the second quadrant of the xy-plane by noting that the two circle sections are symmetric with respect to the y axis and Circle sections in the third and fourth quadrants can be obtained from sections in the first and second quadrants by considering symmetry about the x axis. We can take this one step further and note that there is also symmetry between octants. Circle sections in adjacent octants within one quadrant are symmetric with respect to the 45° line dividing the two octants. These symmetry conditions are illustrated in given figure.



Eight Octants Symmetry



Optimizing the Algorithm

Now this algorithm can be optimized by using symmetric octants as:

```

Circle3 (xc, yc, r)           // procedure to draw circle
Begin
  for  $\theta = 0$  to  $\pi/4$  step  $1/r$ 
  Begin
     $x = x_c + r * \cos \theta$ 
     $y = y_c + r * \sin \theta$ 
    DrawSymmetricPoints(xc, yc, x, y) //call to other procedure
  End
End

```

DrawSymmetricPoints (x_c, y_c, x, y)

Begin

```
Plot ( $x + x_c, y + y_c$ )           // Point in octant 1
Plot ( $y + x_c, x + y_c$ )           // Point in octant 2
Plot ( $-y + x_c, x + y_c$ )         // Point in octant 3
Plot ( $-x + x_c, y + y_c$ )         // Point in octant 4
Plot ( $-x + x_c, -y + y_c$ )        // Point in octant 5
Plot ( $-y + x_c, -x + y_c$ )        // Point in octant 6
Plot ( $y + x_c, -x + y_c$ )        // Point in octant 7
Plot ( $x + x_c, -y + y_c$ )        // Point in octant 8
```

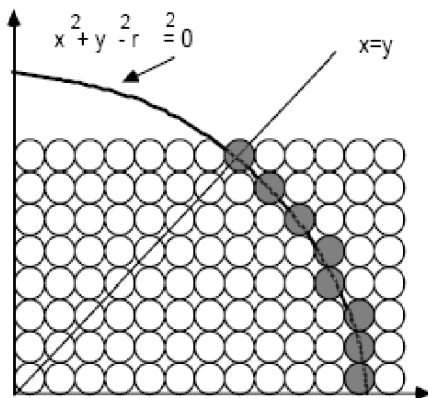
End

Inefficiency Still Prevails

- Hence we have reduced half the calculations by considering symmetric octants of the circle.
- But as we discussed earlier, inefficiency is still there and that is due to the use of floating point calculations.
- In next algorithm we will try to remove this problem.

Midpoint Circle Algorithm

As in the Bresenham line drawing algorithm we derive a decision parameter that helps us to determine whether or not to increment in the y coordinate against increment of x coordinate or vice versa for slope > 1 . Similarly here we will try to derive decision parameter which can give us closest pixel position. Let us consider only the first octant of a circle of radius r centered on the origin. We begin by plotting point $(0, r)$ and end when $x < y$.



The decision at each step is whether to choose:

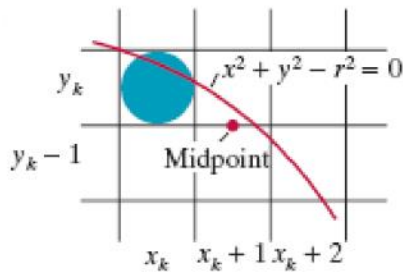
- The pixel to the right of the current pixel or
- The pixel which is to the right and below the current pixel. (8-way stepping)

Assume:

$P=(x_k, y_k)$ is the current pixel.

$Q=(x_k+1, y_k)$ is the pixel to the right

$R=(x_k+1, y_k-1)$ is the pixel to the right and below.



Midpoint
between candidate pixels at
sampling position $x_k + 1$
along a circular path.

To apply the midpoint method, we define a circle function:

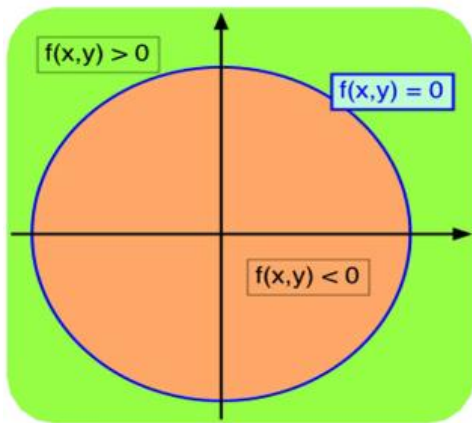
$$f_{\text{circle}}(x, y) = x^2 + y^2 - r^2$$

The following relations can be observed:

$f_{\text{circle}}(x, y) < 0$, if (x, y) is inside the circle boundary.

$f_{\text{circle}}(x, y) = 0$, if (x, y) is on the circle boundary.

$f_{\text{circle}}(x, y) > 0$, if (x, y) is outside the circle boundary.



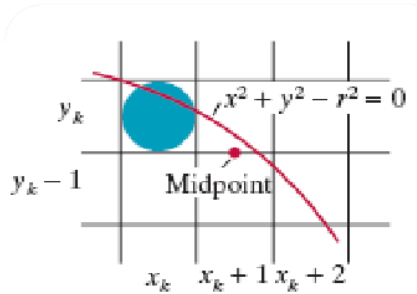
The circle function tests given above are performed for the midpoints between pixels near the circle path at each sampling step. Thus, the circle function is the decision parameter in the midpoint algorithm, and we can set up incremental calculations for this function as we did in the line algorithm. Figure given below shows the midpoint between the two candidate pixels at sampling position x_k+1 . Assuming we have just plotted the pixel at (x_k, y_k) , we next need to determine whether the pixel at position (x_k+1, y_k) , or the one at position (x_k+1, y_k-1) is closer to the circle. Our decision parameter is the circle function evaluated at the midpoint between these two pixels:

$$P_k = f_{\text{circle}}(x_k + 1, y_k - 1/2)$$

$$P_k = (x_k + 1)^2 + (y_k - 1/2)^2 - r^2 \dots \dots \dots (1)$$

If $p_k < 0$, this midpoint is inside the circle and the pixel on scan line y_k is closer to the circle boundary.

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



Midpoint
between candidate pixels at
sampling position $x_k + 1$
along a circular path.

Successive decision parameters are obtained using incremental calculations.

We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$.

$$P_{k+1} = f_{\text{circle}}(x_{k+1} + 1, y_{k+1} - 1/2)$$

$$P_{k+1} = [(x_k + 1) + 1]^2 + (y_{k+1} - 1/2)^2 - r^2 \dots\dots\dots(2)$$

Subtracting (1) from (2), we get:

$$P_{k+1} - P_k = [(x_k + 1) + 1]^2 + (y_{k+1} - 1/2)^2 - r^2 - [(x_k + 1)^2 + (y_k - 1/2)^2 - r^2]$$

$$P_{k+1} = P_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_k - y_{k+1}) + 1 \dots(3)$$

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

Therefore, if $P_k < 0$ or negative then y_{k+1} will be y_k and the formula to calculate P_{k+1} will be:

$$\begin{aligned} P_{k+1} &= P_k + 2(x_k + 1) + (y_k^2 - y_k^2) - (y_k - y_k) + 1 \\ P_{k+1} &= P_k + 2(x_k + 1) + 1 \end{aligned} \dots\dots\dots(4)$$

Otherwise, if $P_k > 0$ or positive then y_{k+1} will be $y_k - 1$ and the formula to calculate P_{k+1} will be:

$$\begin{aligned} P_{k+1} &= P_k + 2(x_k + 1) + [(y_k - 1)^2 - y_k^2] - (y_k - 1 - y_k) + 1 \\ P_{k+1} &= P_k + 2(x_k + 1) + (y_k^2 - 2y_k + 1 - y_k^2) - (y_k - 1 - y_k) + 1 \\ P_{k+1} &= P_k + 2(x_k + 1) - 2y_k + 1 + 1 + 1 \\ P_{k+1} &= P_k + 2(x_k + 1) - 2y_k + 2 + 1 \\ P_{k+1} &= P_k + 2(x_k + 1) - 2(y_k - 1) + 1 \end{aligned} \dots\dots\dots(5)$$

How to calculate initial Decision Parameter P_0 ?

Now a similar case that we observed in line algorithm is that how would starting P_k be calculated.

For this, the starting pixel position will be $(0, r)$.

Therefore, putting this value in equation, we get:

$$P_0 = f_{\text{circle}}(x_k + 1, y_k - 1/2)$$

$$P_0 = (0 + 1)^2 + (r - 1/2)^2 - r^2$$

$$P_0 = 1 + r^2 - r + 1/4 - r^2$$

$$P_0 = 5/4 - r$$

If radius r is specified as an integer, we can simply round p_0 to:

$$P_0 = 1 - r \dots\dots\dots(6)$$

Since all increments are integer, finally the algorithm can be summed up as:

Midpoint Circle Algorithm

MidpointCircle (xc, yc, radius) // procedure to draw circle

Begin

x = 0;

y = r;

p = 1 - r;

do

DrawSymmetricPoints (xc, yc, x, y) // call to another function

x = x + 1

If p < 0 Then

p = p + 2 * (x + 1) + 1

else

y = y - 1

p = p + 2 * (x + 1) - 2 * (y - 1) + 1

while (x < y)

End

DrawSymmetricPoints (xc, yc, x, y)

Begin

Plot (x + xc, y + yc) // Point in octant 1

Plot (y + xc, x + yc) // Point in octant 2

Plot (-y + xc, x + yc) // Point in octant 3

Plot (-x + xc, y + yc) // Point in octant 4

Plot (-x + xc, -y + yc) // Point in octant 5

Plot (-y + xc, -x + yc) // Point in octant 6

Plot (y + xc, -x + yc) // Point in octant 7

Plot (x + xc, -y + yc) // Point in octant 8

End

MIDPOINT CIRCLE DRAWING ALGORITHM

Step 1: Input radius r and circle center (Xc, Yc) and obtain the first point on the circumference of a circle centered on the origin as (X₀, Y₀) = (0, r)

Step 2: Calculate the initial values of the decision parameter as

$$P_0 = 5/4 - r$$

Step 3: At each position starting at k perform the following test:

If $P_k < 0$, the next point to plot is (X_{k+1}, Y_k) and

$$P_{k+1} = P_k + 2X_{k+1} + 1$$

Otherwise the next point is (X_{k+1}, Y_k-1) and

$$P_{k+1} = P_k + 2X_{k+1} + 1 - 2Y_{k+1}$$

where $2X_{k+1} = 2X_k + 2$ and $2Y_{k+1} = 2Y_k - 2$

Step 4: Determine symmetry points in the other seven octants

Step 5: Move each pixel position (X, Y) onto the circular path centered on (X_c, Y_c) and plot the coordinate values as

$$X = X + X_c \quad Y = Y + Y_c$$

Step 6: Repeat steps 3 through until $X \geq Y$

C++ code implementing midpoint circle drawing algorithm

```
#include <iostream>
#include <windows.h>
#include <conio.h>
void gotoxy(int x, int y)
{
    COORD ord;
    ord.X = x;
    ord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), ord);
}

void Circle(int Radius,int xC,int yC)
{
    int P;
    int x,y,X,Y;
        std::cout << std::endl;

        gotoxy(xC, yC);
        std::cout << "a";

    P = 1 - Radius;
    x = 0;
    y = Radius;
        X=xC+x; Y=yC+y;gotoxy(X, Y);std::cout << "i";
        X=xC+x; Y=yC-y;gotoxy(X, Y);std::cout << "i";
        X=xC-x; Y=yC+y;gotoxy(X, Y);std::cout << "i";
        X=xC-x; Y=yC-y;gotoxy(X, Y);std::cout << "i";
        X=xC+y; Y=yC+x;gotoxy(X, Y);std::cout << "i";
        X=xC-y; Y=yC+x;gotoxy(X, Y);std::cout << "i";
        X=xC+y; Y=yC-x;gotoxy(X, Y);std::cout << "i";
        X=xC-y; Y=yC-x;gotoxy(X, Y);std::cout << "i";

    while (x<=y)
    {
        x++;
        if (P<0)
        {
            P += 2 * x + 1;
        }
        else
        {
            P += 2 * (x - y) + 1;
            y--;
        }
    }
}
```

```

    }
    X=xC+x; Y=yC+y;gotoxy(X, Y);std::cout << "x";
    X=xC+x; Y=yC-y;gotoxy(X, Y);std::cout << "x";
    X=xC-x; Y=yC+y;gotoxy(X, Y);std::cout << "x";
    X=xC-x; Y=yC-y;gotoxy(X, Y);std::cout << "x";
    X=xC+y; Y=yC+x;gotoxy(X, Y);std::cout << "x";
    X=xC-y; Y=yC+x;gotoxy(X, Y);std::cout << "x";
    X=xC+y; Y=yC-x;gotoxy(X, Y);std::cout << "x";
    X=xC-y; Y=yC-x;gotoxy(X, Y);std::cout << "x";

}

}

int main()
{
    Circle(7,5,5);
    Circle(10,20,20);
    gotoxy(0,0);
    getch();
}

```

MIDPOINT ELLIPSE DRAWING ALGORITHM

Step 1: Input radius r_x , r_y and ellipse center (X_c, Y_c) and obtain the first point on the circumference of a circle centered on the origin as $(X_0, Y_0) = (0, r_y)$

Step 2: Calculate the initial values of the decision parameter in region 1 as

$$P1_0 = r_y^2 - r_x^2 r_y + 1/4 r_x^2$$

Step 3: At each X_k position in region 1, starting at $k = 0$, perform the following test:

If $P1_k < 0$, the next point to plot is (X_{k+1}, Y_k) and

$$P1_{k+1} = P1_k + 2 r_y^2 X_{k+1} + r_y^2$$

Otherwise the next point is (X_{k+1}, Y_{k-1}) and

$$P1_{k+1} = P1_k + 2 r_y^2 X_{k+1} - 2 r_x^2 Y_{k+1} + r_y^2$$

With

$$2 r_y^2 X_{k+1} = 2 r_y^2 X_k + 2 r_y^2$$

$$2 r_x^2 Y_{k+1} = 2 r_x^2 Y_k - 2 r_x^2$$

Step 4: Calculate the initial values of the decision parameter in region 2 as

$$P2_0 = r_y^2 (X_0 + 1/2)^2 + r_x^2 (Y_0 - 1)^2 - r_x^2 r_y^2$$

Step 5: At each position starting at Y_k position in region 2, starting at $k = 0$, perform the following test:

If $P2_k > 0$, the next point to plot is (X_k, Y_{k-1}) and

$$P2_{k+1} = P2_k - 2 r_y^2 Y_{k+1} + r_x^2$$

Otherwise the next point is (X_{k+1}, Y_{k-1}) and

$$P2_{k+1} = P2_k + 2 r_y^2 X_{k+1} - 2 r_x^2 Y_{k+1} + r_x^2$$

Step 6: Determine symmetry points in the other three octants

Step 7: Move each pixel position (X, Y) onto the circular path centered on

(X_c, Y_c) and plot the coordinate values as

$$X = X + X_c \quad Y = Y + Y_c$$

Step 8: Repeat steps for region 1 until $2 r_y^2 X \geq 2 r_x^2 Y$

Chapter 5: Transformations

Design applications and facility layouts are created by arranging the orientations and sizes of the component parts of the scene. And animations are produced by moving the "camera" or the objects in a scene along animation paths. Changes in orientation, size, and shape are accomplished with geometric transformations that alter the coordinate descriptions of objects. The basic geometric transformations are translation, rotation, and scaling.

2D Transformations

Given a point cloud, polygon, or sampled parametric curve, we can use transformations for several purposes:

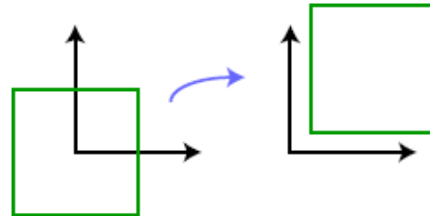
1. Change coordinate frames (world, window, viewport, device, etc).
2. Compose objects of simple parts with local scale/position/orientation of one part defined with regard to other parts. For example, for articulated objects.
3. Use deformation to create new shapes.
4. Useful for animation.

There are three basic classes of transformations:

1. **Rigid body** - Preserves distance and angles.
 - Examples: translation and rotation.
2. **Conformal** - Preserves angles.
 - Examples: translation, rotation, and uniform scaling.
3. **Affine** - Preserves parallelism. Lines remain lines.
 - Examples: translation, rotation, scaling, shear, and reflection.

types of transformations:

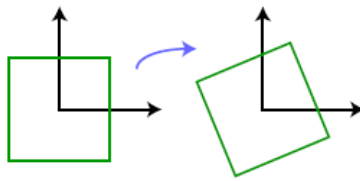
- **Translation** by vector \vec{t} : $\bar{p}_1 = \bar{p}_0 + \vec{t}$.



Translation moves every point a constant distance in a specified direction. Like any rigid motion, a translation may be decomposed into two reflections that are two indirect isometries. A translation can also be interpreted as the addition of a constant vector to every point, or as shifting the origin of the coordinate system.

- **Rotation** clockwise by θ : $\bar{p}_1 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \bar{p}_0$.

Rotation is a transformation in a plane or in space that describes the motion of a rigid body around a fixed point.



In two dimensions, only a single angle is needed to specify a rotation in two dimensions – the angle of rotation. To carry out a rotation using matrices the point (x, y) to be rotated is written as a vector, then multiplied by a matrix calculated from the angle, θ , like so:

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

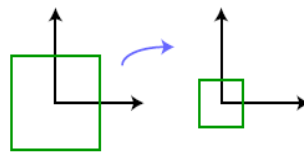
where (x', y') are the co-ordinates of the point after rotation, and the formulae for x' and y' can be seen to be

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

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

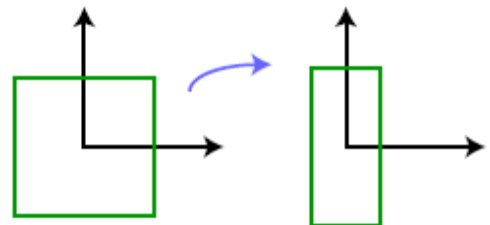
The vectors $\begin{bmatrix} x \\ y \end{bmatrix}$ and $\begin{bmatrix} x' \\ y' \end{bmatrix}$ have the same magnitude and are separated by an angle θ as expected.

- **Uniform scaling** by scalar a : $\bar{p}_1 = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \bar{p}_0$.



Uniform scaling is a linear transformation that enlarges (increases) or shrinks (diminishes) objects by a scale factor that is the same in all directions. The result of uniform scaling is similar (in the geometric sense) to the original. A scale factor of 1 is normally allowed, so that congruent shapes are also classed as similar.

- **Nonuniform scaling** by a and b : $\bar{p}_1 = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \bar{p}_0$.



Non-uniform scaling is obtained when at least one of the scaling factors is different from the others; a special case is directional scaling or stretching (in one direction). Non-uniform scaling changes the shape of the object; e.g. a square may change into a rectangle, or into a parallelogram if the sides of the square are not parallel to the scaling axes (the angles between lines parallel to the axes are preserved, but not all angles). Matrix representation, A scaling can be represented by a scaling matrix. To scale an

object by a vector $v = (v_x, v_y, v_z)$, each point $p = (p_x, p_y, p_z)$ would need to be multiplied with this scaling matrix:

$$S_v = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix}.$$

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

$$S_v p = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} v_x p_x \\ v_y p_y \\ v_z p_z \end{bmatrix}.$$

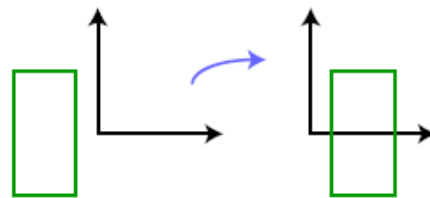
Such a scaling changes the diameter of an object by a factor between the scale factors, the area by a factor between the smallest and the largest product of two scale factors, and the volume by the product of all three.

The scaling is uniform if and only if the scaling factors are equal $(v_x = v_y = v_z)$. If all except one of the scale factors are equal to 1, we have directional scaling.

In the case where $v_x = v_y = v_z = k$, the scaling is also called an enlargement or dilation by a factor k , increasing the area by a factor of k^2 and the volume by a factor of k^3 .

A scaling in the most general sense is any affine transformation with a diagonalizable matrix. It includes the case that the three directions of scaling are not perpendicular. It includes also the case that one or more scale factors are equal to zero (projection), and the case of one or more negative scale factors. The latter corresponds to a combination of scaling proper and a kind of reflection: along lines in a particular direction we take the reflection in the point of intersection with a plane that need not be perpendicular; therefore it is more general than ordinary reflection in the plane.

- **Reflection about the y -axis:** $\bar{p}_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \bar{p}_0$.



3.2 Linear transformation

Linear transformation is a function between two vector spaces that preserves the operations of vector addition and scalar multiplication. As a result, it always maps straight lines to straight lines or 0. The expression "linear operator" is commonly used for linear maps from a vector space to itself (i.e., endomorphisms).

Examples of linear transformation matrices

- rotation by 90 degrees clockwise:

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

- rotation by θ degrees clockwise:

$$\mathbf{A} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

- reflection against the x axis:

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

- reflection against the y axis:

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

- scaling by 2 in all directions:

$$\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

- Vertical shear mapping:

$$\mathbf{A} = \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix}$$

- squeezing:

$$\mathbf{A} = \begin{bmatrix} k & 0 \\ 0 & 1/k \end{bmatrix}$$

- projection onto the y axis:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

3.3 Homogeneous Coordinates

Homogeneous coordinates are another way to represent points to simplify the way in which we express affine transformations. Normally, bookkeeping would become tedious

when affine transformations of the form $A\bar{p} + \vec{t}$ are composed. With homogeneous

coordinates, affine transformations become matrices, and composition of transformations is as simple as matrix multiplication.

Formulas involving homogeneous coordinates are often simpler and more symmetric than their Cartesian counterparts. If the homogeneous coordinates of a point are multiplied by a non-zero scalar then the resulting coordinates represent the same point. An additional condition must be added on the coordinates to ensure that only one set of coordinates corresponds to a given point, so the number of coordinates required is, in general, one more than the dimension of the projective space being considered. For example, two homogeneous coordinates are required to specify a point on the projective line and three homogeneous coordinates are required to specify a point on the projective plane

- Any point in the projective plane is represented by a triple (X, Y, Z) , called the homogeneous coordinates of the point, where X, Y and Z are not all 0.
- The point represented by a given set of homogeneous coordinates is unchanged if the coordinates are multiplied by a common factor.
- Conversely, two sets of homogeneous coordinates represent the same point if and only if one is obtained from the other by multiplying by a common factor.
- When Z is not 0 the point represented is the point $(X/Z, Y/Z)$ in the Euclidean plane.
- When Z is 0 the point represented is a point at infinity.
- Note that the triple $(0, 0, 0)$ is omitted and does not represent any point. The origin is represented by $(0, 0, 1)$.

3.4 Hierarchical Transformations

It is often convenient to model objects as hierarchically connected parts. For example, a robot arm might be made up of an upper arm, forearm, palm, and fingers. Rotating at the shoulder on the upper arm would affect all of the rest of the arm, but rotating the forearm at the elbow would affect the palm and fingers, but not the upper arm. A reasonable hierarchy, then, would have the upper arm at the root, with the forearm as its only child, which in turn connects only to the palm, and the palm would be the parent to all of the fingers.

Each part in the hierarchy can be modeled in its own local coordinates, independent of the other parts. For a robot, a simple square might be used to model each of the upper arm, forearm, and so on. Rigid body transformations are then applied to each part relative to its parent to achieve the proper alignment and pose of the object. For example, the fingers are positioned to be in the appropriate places in the palm coordinates, the fingers and palm together are positioned in forearm coordinates, and the process continues up the hierarchy. Then a transformation applied to upper arm coordinates is also applied to all parts down the hierarchy.

Chapter 6: Introduction to OpenGL

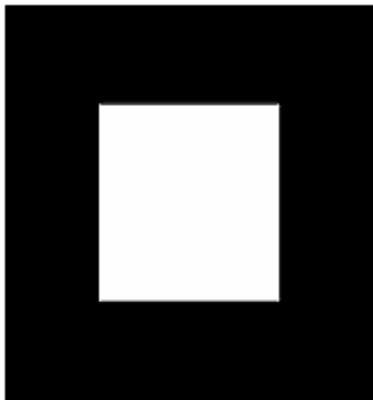
What Is OpenGL?

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitives - points, lines, and polygons.

A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

Example of OpenGL code: White Rectangle on a Black Background



```
#include <whateverYouNeed.h>

main() {

    InitializeAWindowPlease();

    glClearColor (0.0, 0.0, 0.0, 0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
```

```

    glVertex3f (0.25, 0.75, 0.0);
glEnd();
glFlush();

```

```

    UpdateTheWindowAndCheckForEvents();
}

```

The first line of the **main()** routine initializes a *window* on the screen: The **InitializeAWindowPlease()** routine is meant as a placeholder for window system-specific routines, which are generally not OpenGL calls. The next two lines are OpenGL commands that clear the window to black: **glClearColor()** establishes what color the window will be cleared to, and **glClear()** actually clears the window. Once the clearing color is set, the window is cleared to that color whenever **glClear()** is called. This clearing color can be changed with another call to **glClearColor()**. Similarly, the **glColor3f()** command establishes what color to use for drawing objects - in this case, the color is white. All objects drawn after this point use this color, until it's changed with another call to set the color.

The next OpenGL command used in the program, **glOrtho()**, specifies the coordinate system OpenGL assumes as it draws the final image and how the image gets mapped to the screen. The next calls, which are bracketed by **glBegin()** and **glEnd()**, define the object to be drawn - in this example, a polygon with four vertices. The polygon's "corners" are defined by the **glVertex3f()** commands. As you might be able to guess from the arguments, which are (*x*, *y*, *z*) coordinates, the polygon is a rectangle on the *z*=0 plane.

Finally, **glFlush()** ensures that the drawing commands are actually executed rather than stored in a *buffer* awaiting additional OpenGL commands. The

UpdateTheWindowAndCheckForEvents() placeholder routine manages the contents of the window and begins event processing.

OpenGL Command Syntax

OpenGL commands use the prefix **gl** and initial capital letters for each word making up the command name (recall **glClearColor()**, for example). Similarly, OpenGL defined constants begin with **GL_**, use all capital letters, and use underscores to separate words (like **GL_COLOR_BUFFER_BIT**).

OpenGL as a State Machine

OpenGL is a state machine. You put it into various states (or modes) that then remain in effect until you change them. You can set the current color to white, red, or any other color, and thereafter every object is drawn with that color until you set the current color to something else. The current color is only one of many state variables that OpenGL maintains. Others control such things as the current viewing and projection transformations, line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many state variables refer to modes that are enabled or disabled with the command **glEnable()** or **glDisable()**.

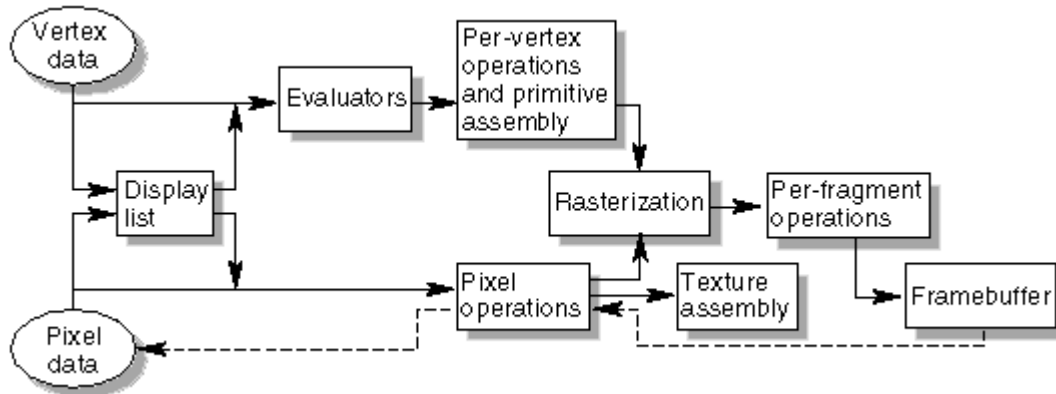
Each state variable or mode has a default value, and at any point you can query the system for each variable's current value. Typically, you use one of the six following commands to do this: **glGetBooleanv()**, **glGetDoublev()**, **glGetFloatv()**, **glGetIntegerv()**, **glGetPointerv()**, or **glIsEnabled()**. Which of these commands you

select depends on what data type you want the answer to be given in. Some state variables have a more specific query command (such as **glGetLight*()**, **glGetError()**, or **glGetPolygonStipple()**). In addition, you can save a collection of state variables on an attribute stack with **glPushAttrib()** or **glPushClientAttrib()**, temporarily modify them, and later restore the values with **glPopAttrib()** or **glPopClientAttrib()**. For temporary state changes, you should use these commands rather than any of the query commands, since they're likely to be more efficient.

OpenGL Rendering Pipeline

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline.

The following diagram shows the Henry Ford assembly line approach, which OpenGL takes to processing data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the framebuffer.



Display Lists - All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (The alternative to retaining data in a display list is processing the data immediately - also known as *immediate mode*.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators - All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

Per-Vertex Operations - For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting

calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

Primitive Assembly - Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped.

In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines.

The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

Pixel Operations - While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.

If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the framebuffer.

Texture Assembly - An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

Rasterization - Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the framebuffer. Line and polygon stippling, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

Fragment Operations - Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

The first operation which may be encountered is texturing, where a texel (texture element) is generated from texture memory for each fragment and applied to the fragment. Then fog calculations may be applied, followed by the scissor test, the alpha test, the stencil test, and the depth-buffer test (the depth buffer is for hidden-surface removal). Failing an enabled test may end the continued processing of a fragment's

square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed. Finally, the thoroughly processed fragment is drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.

OpenGL-Related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

- i) **GLUT** is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL Programming.
- ii) **GLX** is used on Unix OpenGL implementation to manage interaction with the X Window System and to encode OpenGL onto the X protocol stream for remote rendering.
- iii) **GLU** is the OpenGL Utility Library. This is a set of functions to create texture mipmaps from a base image, map coordinates between screen and object space, and draw quadric surfaces and NURBS.
- iv) **DRI** is the Direct Rendering Infrastructure for coordinating the Linux kernel, X window system, 3D graphics hardware and an OpenGL-based rendering engine.

Coordinate System

A coordinate system is a system which uses one or more numbers, or coordinates, to uniquely determine the position of a point or other geometric element. The order of the coordinates is significant and they are sometimes identified by their position in an ordered tuple and sometimes by a letter, as in 'the x-coordinate'. In elementary mathematics the coordinates are taken to be real numbers, but in more advanced applications coordinates can be taken to be complex numbers or elements of a more abstract system such as a commutative ring. The use of a coordinate system allows problems in geometry to be translated into problems about numbers and vice versa; this is the basis of analytic geometry.

An example in everyday use is the system of assigning longitude and latitude to geographical locations. In physics, a coordinate system used to describe points in space is called a frame of reference.

- World coordinate system- The world coordinate system contains the simulated world (the scene to be rendered) in other words represents a coordinate system in given units, that represents a given application program of the world. Also known as the "universe" or sometimes "model" coordinate system. This is the base reference system for the overall model, (generally in 3D), to which all other model coordinates relate.
- Cartesian Coordinate system - specifies each point uniquely in a plane by a pair of numerical coordinates, which are the signed distances from the point to two fixed perpendicular directed lines, measured in the same unit of length.
- Screen coordinates system: The grid or bit map a computer uses to turn monitor pixels on and off. This 2D coordinate system refers to the physical coordinates of the pixels on the computer screen, based on current screen resolution. (E.g. 1024x768)

- **Object Coordinate System** - When each object is created in a modelling program, the modeller must pick some point to be the origin of that particular object, and the orientation of the object to a set of model axes. For example when modelling a desk, the modeller might choose a point in the center of the desk top for the origin, or the point in the center of the desk at floor level, or the bottom of one of the legs of the desk. When this object is moved to a point in the world coordinate system, it is really the origin of the object (in object coordinate system) that is moved to the new world coordinates, and all other points in the model are moved by an equal amount. Note that while the origin of the object model is usually somewhere on the model itself, it does not have to be. For example, the origin of a doughnut or a tire might be in the vacant space in the middle.
- **Hierarchical Coordinate Systems** - Sometimes objects in a scene are arranged in a hierarchy, so that the "position" of one object in the hierarchy is relative to its parent in the hierarchy scheme, rather than to the world coordinate system. For example, a hand may be positioned relative to an arm, and the arm relative to the torso. When the arm moves, the hand moves with it, and when the torso moves, all three objects move together.
- **Viewpoint Coordinate System** - Also known as the "camera" coordinate system. This coordinate system is based upon the viewpoint of the observer, and changes as they change their view. Moving an object "forward" in this coordinate system moves it along the direction that the viewer happens to be looking at the time.
- **Model Window Coordinate System** - Not to be confused with desktop windowing systems (MS Windows or X Windows), this coordinate system refers to the subset of the overall model world that is to be displayed on the screen. Depending on the viewing parameters selected, the model window may be rectilinear or a distorted viewing frustum of some kind.
- **Viewport Coordinate System** - This coordinate system refers to a subset of the screen space where the model window is to be displayed. Typically the viewport will occupy the entire screen window, or even the entire screen, but it is also possible to set up multiple smaller viewports within a single screen window.

Chapter 7: Lighting and Shading

This is one of the primary elements of generating realistic images. Light sources generate energy, which we may think of as being composed of extremely tiny packets of energy, called *photons*. The photons are reflected and transmitted in various ways throughout the environment. They bounce off various surfaces and may be scattered by smoke or dust in the air. Eventually, some of them enter our eye and strike our retina. We perceive the resulting amalgamation of photons of various energy levels in terms of *color*. The more accurately we can simulate this physical process, the more realistic lighting will be. Unfortunately, computers are not fast enough to produce a truly realistic simulation of indirect reflections in real time, and so we will have to settle for much simpler approximations.

Illumination

local illumination model, which means that the shading of a point depends *only* on its relationship to the light sources, without considering the other objects in the scene.

global illumination model, in which light reflected or passing through one object might affects the illumination of other objects. Global illumination models deal with many affects, such as shadows, indirect illumination, color bleeding (colors from one object reflecting and altering the color of a nearby object), caustics (which result when light passes through a lens and is focused on another surface).

Light: we can imagine a simple model of light consisting of a large number of photons being emitted continuously from each light source. Each photon has an associated energy, which (when aggregated over millions of different reflected photons) we perceive as *color*. Although color is complex phenomenon, for our purposes it is sufficient to consider color to be modeled as a triple of red, green, and blue components. The strength or *intensity* of the light at any location can be modeled in terms of the *flux*, that is, the amount of illumination energy passing through a fixed area over a fixed amount of time. Assuming that the atmosphere is a vacuum (in particular there is no smoke or fog), a photon of light travels unimpeded until hitting a surface, after which one of three things can happen

Reflection: The photon can be reflected or scattered back into the atmosphere. If the surface were perfectly smooth (like a mirror or highly polished metal) the reflection would satisfy the rule “angle of incidence equals angle of reflection” and the result would be a mirror-like and very shiny in appearance. On the other hand, if the surface is rough at a microscopic level (like foam rubber, say) then the photons are scattered nearly uniformly in all directions. We can further distinguish different varieties of reflection:

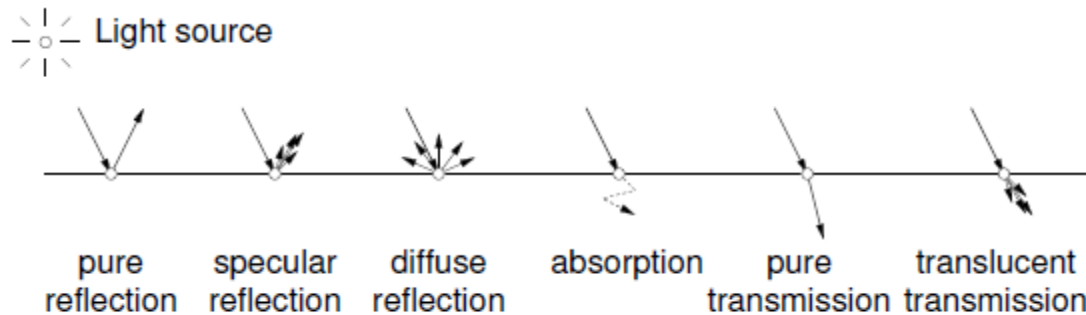
Pure reflection: Perfect mirror-like reflection

Specular reflection: Imperfect reflection like brushed metal and shiny plastics.

Diffuse reflection: Uniformly scattering, and hence not shiny.

Absorption: The photon can be absorbed into the surface (and hence dissipates in the form of heat energy). We do not see this light. Thus, an object appears to be green, because it reflects photons in the green part of the spectrum and absorbs photons in the other regions of the visible spectrum.

Transmission: The photon can pass through the surface. This happens perfectly with transparent objects (like glass and polished gem stones) and with a significant amount of scattering with translucent objects (like human skin or a thin piece of tissue paper).



The ways in which a photon of light can interact with a surface.

Light Sources: In reality, light sources come in many sizes and shapes. They may emit light in varying intensities and wavelengths according to direction. The intensity of light energy is distributed across a continuous spectrum of wavelengths.

Lighting in real environments usually involves a considerable amount of indirect reflection between objects of the scene. If we were to ignore this effect and simply consider a point to be illuminated only if it can see the light source, then the resulting image in which objects in the shadows are totally black. In indoor scenes we are accustomed to seeing much softer shading, so that even objects that are hidden from the light source are partially illuminated.

scattering of light modeled by breaking the light source's intensity into two components: *ambient emission* and *point emission*.

Ambient emission: Refers to light that does not come from any particular location. Like heat, it is assumed to be scattered uniformly in all locations and directions. A point is illuminated by ambient emission even if it is not visible from the light source.

Point emission: Refers to light that originates from a single point. In theory, point emission only affects points that are directly visible to the light source. That is, a point p is illuminate by light source q if and only if the open line segment pq does not intersect any of the objects of the scene.

Attenuation: The light that is emitted from a point source is subject to *attenuation*, that is, the decrease in strength of illumination as the distance to the source increases. Physics tells us that the intensity of light falls off as the inverse square of the distance.

Types of light reflection: The next issue needed to determine how objects appear is how this light is *reflected* off of the objects in the scene and reach the viewer. We will assume that all objects are opaque. The simple model that we will use for describing the reflectance properties of objects is called the *Phong model*. The model is over 20 years old, and is based on modeling surface reflection as a combination of the following components:

Emission: This is used to model objects that glow (even when all the lights are off). This is unaffected by the presence of any light sources. However, because our illumination model is local, it does not behave like a light source, in the sense that it does not cause any other objects to be illuminated.

Ambient reflection: This is a simple way to model indirect reflection. All surfaces in all positions and orientations are illuminated equally by this light energy.

Diffuse reflection: The illumination produced by matter (i.e, dull or nonshiny) smooth objects, such as foam/ rubber.

Specular reflection: The bright spots appearing on smooth shiny (e.g. metallic or polished) surfaces. Although specular reflection is related to pure reflection (as with mirrors), for the purposes of our simple model these two are different. In particular, specular reflection only reflects light, not the surrounding objects in the scene.

Texture Mapping

Surface Detail: This is fine for smooth surfaces of uniform color (plaster walls, plastic cups, metallic objects), but many of the objects that we want to render have some complex surface finish that we would like to model. In theory, it is possible to try to model objects with complex surface finishes through extremely detailed models (e.g. modeling the cover of a book on a character by character basis) or to define some sort of regular mathematical texture function (e.g. a checkerboard or modeling bricks in a wall).

Textures and Texture Space: Although originally designed for textured surfaces, the process of *texture mapping* can be used to map (or .wrap.) any digitized image onto a surface. For example, suppose that we want to render a picture of the Mona Lisa, or wrap an image of the earth around a sphere, or draw a grassy texture on a soccer field. We could download a digitized photograph of the texture, and then map this image onto surface as part of the rendering process.

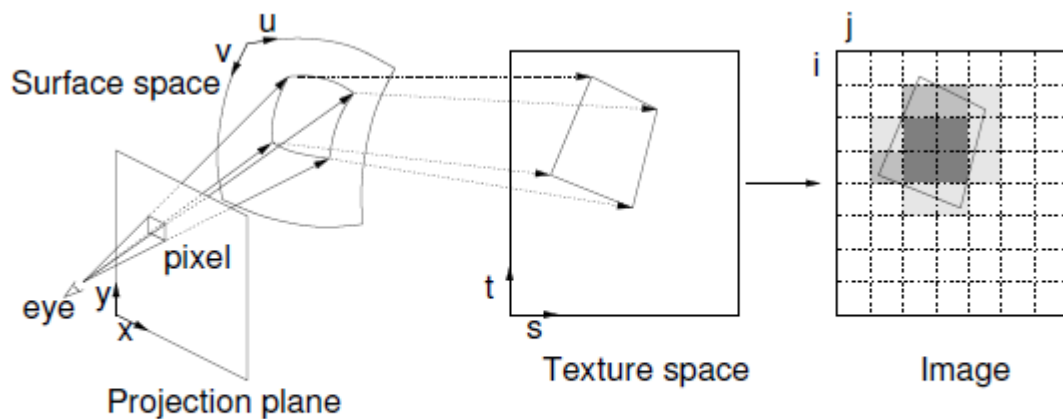
Parameterizations: We wish to .wrap. this 2-dimensional texture image onto a 2-dimensional surface. We need to define a wrapping function that achieves this. The surface resides in 3-dimensional space, so the wrapping function would need to map a point $(s; t)$ in texture space to the corresponding point $(x; y; z)$ in 3-space. This is typically done by first computing a 2-dimensional *parameterization* of the surface. This means that we associate each point on the object surface with two coordinates $(u; v)$ in *surface space*. Then we have three functions, $x(u; v)$, $y(u; v)$ and $z(u; v)$, which map the parameter pair $(u; v)$ to the $x; y; z$ -coordinates of the corresponding surface point. We then map a point $(u; v)$ in the parameterization to a point $(s; t)$ in texture space.

The Texture Mapping Process: Suppose that the unwrapping function IW , and a parameterization of the surface are given. Here is an overview of the texture mapping process. (See Figure below) We will discuss some of the details below.

Project pixel to surface: First we consider a pixel that we wish to draw. We determine the *fragment* of the object's surface that projects onto this pixel, by determining which points of the object project through the corners of the pixel. Let us assume for simplicity that a single surface covers the entire fragment. Otherwise we should average the contributions of the various surfaces fragments to this pixel.

Parameterize: We compute the surface space parameters $(u; v)$ for each of the four corners of the fragment. This generally requires a function for converting from the $(x; y; z)$ coordinates of a surface point to its $(u; v)$ parameterization.

Unwrap and average: Then we apply the inverse wrapping function to determine the corresponding region of texture space. Note that this region may generally have curved sides, if the inverse wrapping function is nonlinear. We compute the average intensity of the texels in this region of texture space, by computing a weighted sum of the texels that overlap this region, and then assign the corresponding average color to the pixel.



Texture mapping overview.

Bump and Environment Mapping

Bump mapping: Texture mapping is good for changing the surface color of an object, but we often want to do more. For example, if we take a picture of an orange, and map it onto a sphere, we find that the resulting object does not look realistic. The reason is that there is an interplay between the bumpiness of the orange's peel and the light source. As we move our viewpoint from side to side, the specular reflections from the bumps should move as well. However, texture mapping alone cannot model this sort of effect. Rather than just mapping colors, we should consider mapping whatever properties affect local illumination. One such example is that of mapping surface normals, and this is what *bump mapping* is all about



A bump-mapped object



A bump-map image

Bump mapping.

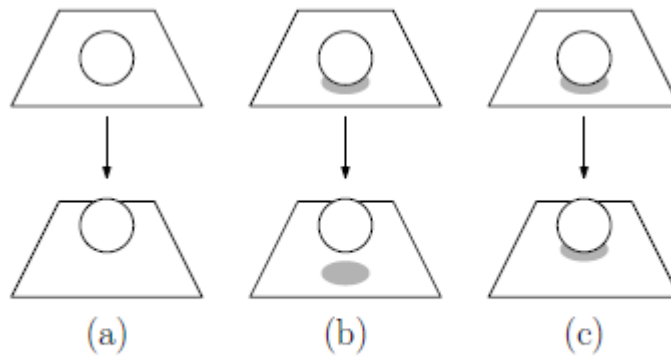
What is the underlying reason for this effect? The bumps are too small to be noticed through perspective depth. It is the subtle variations in *surface normals* that causes this effect. At first it seems that just displacing the surface normals would produce a rather artificial effect. But in fact, bump mapping produces remarkably realistic bumpiness effects. (For example, in above, the object appears to have a bumpy exterior, but an

inspection of its shadow shows that it is in fact modeled as a perfect geometric sphere. It just .looks. bumpy.)

How it's done: As with texture mapping we are presented with an image that encodes the bumpiness. Think of this as a monochrome (gray-scale) image, where a large (white) value is the top of a bump and a small (black) value is a valley between bumps. (An alternative, and more direct way of representing bumps would be to give a *normal map* in which each pixel stores the (x; y; z) coordinates of a normal vector. One reason for using gray-valued bump maps is that they are often easier to compute and involve less storage space.) As with texture mapping, it will be more elegant to think of this discrete image as an encoding of a continuous 2-dimensional *bump space*, with coordinates s and t . The gray-scale values encode a function called the *bump displacement function* $b(s; t)$, which maps a point $(s; t)$ in bump space to its (scalar-valued) height. As with texture mapping, there is an *inverse wrapping function* IW , which maps a point $(u; v)$ in the object's surface parameter space to $(s; t)$ in bump space.

Shadows

Shadows: Shadows give an image a much greater sense of realism. The manner in which objects cast shadows onto the ground and other surrounding surfaces provides us with important visual cues on the spatial relationships between these objects. As an example of this, imagine that you are looking down (say, at a 45_ angle) at a ball sitting on a smooth table top. Suppose that (1) the ball is moved vertically straight up a short distance, or (2) the ball is moved horizontally directly away from you by a short distance. In either case, the impression in the visual frame is essentially the same. That is, the ball moves upwards in the picture's frame (see Figure below).



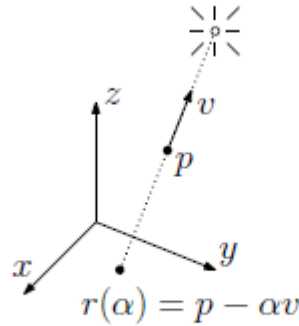
The role of shadows in ascertaining spatial relations.

If the ball's shadow were drawn, however, the difference would be quite noticeable. In case (1) (vertical motion), the shadow remains in a fixed position on the table as the ball moves away. In case (2) (horizontal motion), the ball and its shadow both move together.

Hard and soft shadows: In real life, with few exceptions, we experience shadows as .fuzzy. objects. The reason is that most light sources are engineered to be area sources, not point sources. One notable exception is the sun on a cloudless day. When a light source covers some area, the shadow varies from regions that completely outside the shadow, to a region, called the *penumbra*, where the light is partially visible, to a region called the *umbra*, where the light is totally hidden. The umbra region is completely

shadowed. The penumbra, in contrast, tends to vary smoothly from shadow to unshadowed as more and more of the light source is visible to the surface. Rendering penumbra effects is computational quite intensive, since methods are needed to estimate the fraction of the area of the light source that is visible to a given point on the surface. Static rendering methods, such as ray-tracing, can model these effects. In contrast, real-time systems almost always render hard shadows, or employ some image trickery (e.g., blurring) to create the illusion of soft shadows. In the examples below, we will assume that the light source is a point, and we will consider rendering of hard shadows only.

Shadow Painting: Perhaps the simplest and most sneaky way in which to render shadows is to simply .paint. them onto the surfaces where shadows are cast. For example, suppose that a shadow is being cast on flat table top by some occluding object P. First, we compute the shape of P's shadow P' on the table top, and then we render a polygon in the shape of P' directly onto the table. If P is a polygon and the shadow is being cast onto a flat surface, then the shadow shape P' will also be a polygon. Therefore, we need only determine the transformation that maps each vertex v of P to the corresponding shadow vertex v_0 on the table top. This process is illustrated in Figure below.



The shadow projection transformation.

Chapter 8: Viewing

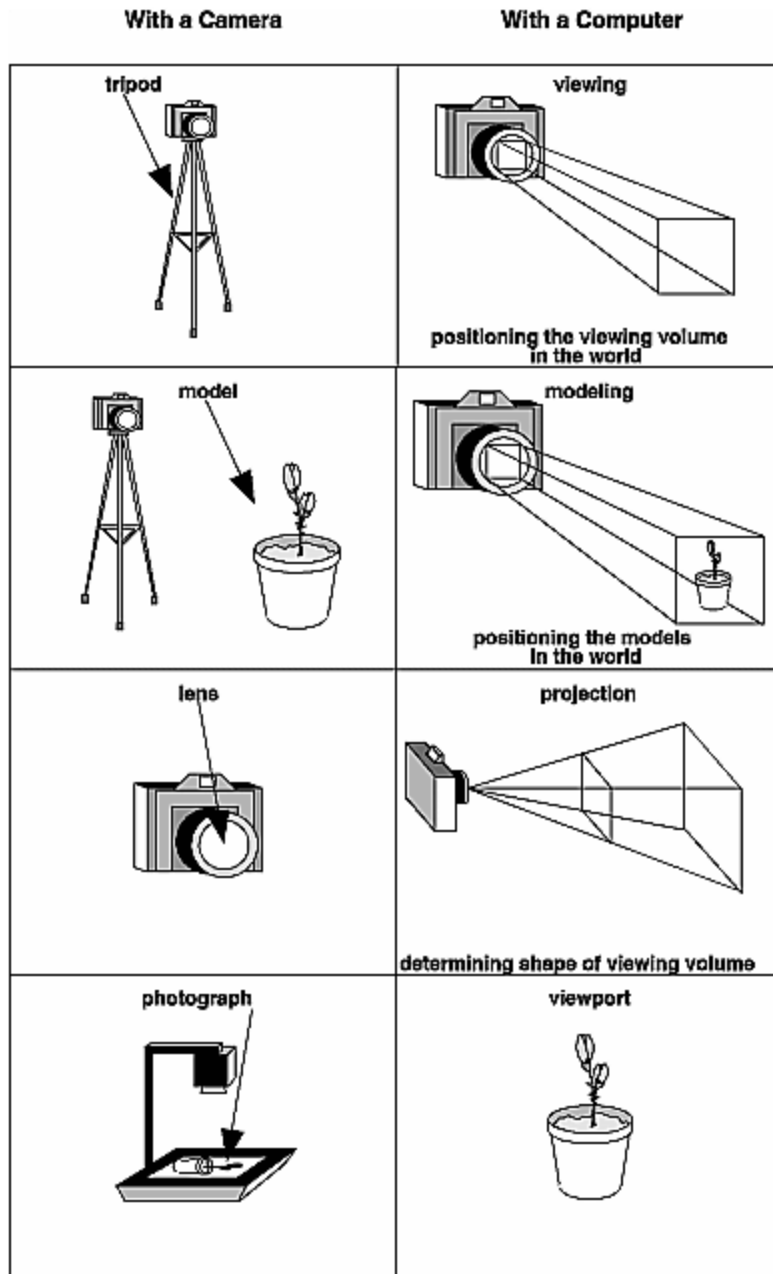
Is the mechanism under which views of a picture are displayed on an output device. A graphics package allows a user to specify which part of defined picture is to be displayed and where that part is to be placed on a display device.

The Camera Analogy

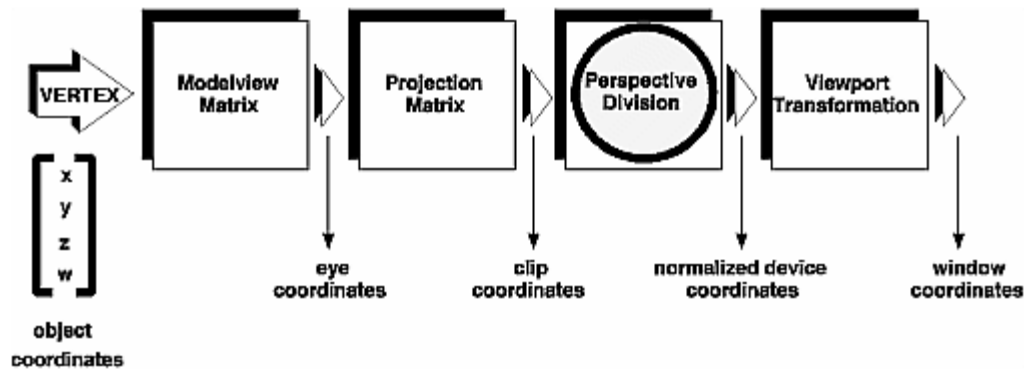
The transformation process to produce the desired scene for viewing is analogous to taking a photograph with a camera. As shown in Figure below, the steps with a camera (or a computer) might be the following:

- Set up your tripod and pointing the camera at the scene (viewing transformation).
- Arrange the scene to be photographed into the desired composition (modeling transformation).
- Choose a camera lens or adjust the zoom (projection transformation).
- Determine how large you want the final photograph to be - for example, you might want it enlarged (viewport transformation).

After these steps are performed, the picture can be snapped or the scene can be drawn.



Note that these steps correspond to the order in which you specify the desired transformations in your program, not necessarily the order in which the relevant mathematical operations are performed on an object's vertices. The viewing transformations must precede the modeling transformations in your code, but you can specify the projection and viewport transformations at any point before drawing occurs. Figure below shows the order in which these operations occur on your computer.



Stages of Vertex Transformation

To specify viewing, modeling, and projection transformations, you construct a 4×4 matrix M , which is then multiplied by the coordinates of each vertex v in the scene to accomplish the transformation

$$v' = Mv$$

(Remember that vertices always have four coordinates (x, y, z, w) , though in most cases w is 1 and for two-dimensional data z is 0.) Note that viewing and modeling transformations are automatically applied to surface normal vectors, in addition to vertices. (Normal vectors are used only in eye coordinates.) This ensures that the normal vector's relationship to the vertex data is properly preserved.

The viewing and modeling transformations you specify are combined to form the modelview matrix, which is applied to the incoming object coordinates to yield eye coordinates. Next, if you've specified additional clipping planes to remove certain objects from the scene or to provide cutaway views of objects, these clipping planes are applied. After that, OpenGL applies the projection matrix to yield clip coordinates. This transformation defines a viewing volume; objects outside this volume are clipped so that they're not drawn in the final scene. After this point, the perspective division is performed by dividing coordinate values by w , to produce normalized device coordinates. Finally, the transformed coordinates are converted to window coordinates by applying the viewport transformation. You can manipulate the dimensions of the viewport to cause the final image to be enlarged, shrunk, or stretched.

You might correctly suppose that the x and y coordinates are sufficient to determine which pixels need to be drawn on the screen. However, all the transformations are performed on the z coordinates as well. This way, at the end of this transformation process, the z values correctly reflect the depth of a given vertex (measured in distance away from the screen). One use for this depth value is to eliminate unnecessary drawing. For example, suppose two vertices have the same x and y values but different z values. OpenGL can use this information to determine which surfaces are obscured by other surfaces and can then avoid drawing the hidden surfaces.

Parallel projections

Parallel projections have lines of projection that are parallel both in reality and in the projection plane. Parallel projection corresponds to a perspective projection with an infinite focal length (the distance from the image plane to the projection point), or "zoom". Within parallel projection there is an ancillary category known as "pictorials".

Pictorials show an image of an object as viewed from a skew direction in order to reveal all three directions (axes) of space in one picture. Because pictorial projections innately contain this distortion, in the role, drawing instrument for pictorials, some liberties may be taken for economy of effort and best effect.

There are two types of parallel projections: -

i) ***Orthographic projection***

Parallel projections have lines of projection that are parallel both in reality and in the projection plane. Parallel projection corresponds to a perspective projection with an infinite focal length (the distance from the image plane to the projection point), or "zoom". Within parallel projection there is an ancillary category known as "pictorials". Pictorials show an image of an object as viewed from a skew direction in order to reveal all three directions (axes) of space in one picture. Because pictorial projections innately contain this distortion, in the role, drawing instrument for pictorials, some liberties may be taken for economy of effort and best effect.

- It means representing a 3D object on a 2D plane
- It is a form of a parallel projection, where all the projection lines perpendicular to the projection plane. This results in every plane of the scene appearing in affine transformation on the surface.

ii) ***Oblique projection***

In oblique projections the parallel projection rays are not perpendicular to the viewing plane as with orthographic projection, but strike the projection plane at an angle other than ninety degrees. In both orthographic and oblique projection, parallel lines in space appear parallel on the projected image. Because of its simplicity, oblique projection is used exclusively for pictorial purposes rather than for formal, working drawings. In an oblique pictorial drawing, the displayed angles among the axes as well as the foreshortening factors (scale) are arbitrary. The distortion created thereby is usually attenuated by aligning one plane of the imaged object to be parallel with the plane of projection thereby creating a true shape, full-size image of the chosen plane. Special types of oblique projections are cavalier projection and cabinet projection.

- It is a type of parallel projection
- it projects an image by intersecting parallel rays (projectors)
- The projectors in oblique projection intersect the projection plane at an oblique angle to produce the projected image

Clipping

Any procedure which identifies that portion of a picture which is either inside or outside a picture is referred to as a clipping algorithm or clipping. The region against which an object is to be clipped is called clipping window. The following are the various primitive types we can clip

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

line clipping algorithms

- i. Cohen-Sutherland

The algorithm divides a 2D space into 9 regions, of which only the middle part (viewport) is visible. In 1967, flight simulation work by Danny Cohen led to the development of the Cohen–Sutherland computer graphics two and three dimensional line clipping algorithms, created with Ivan Sutherland.

The algorithm includes, excludes or partially includes the line based on where:

- Both endpoints are in the viewport region (bitwise OR of endpoints == 0): trivial accept.
- Both endpoints are on the same non-visible region (bitwise AND of endpoints != 0): trivial reject.
- Both endpoints are in different regions: In case of this non trivial situation the algorithm finds one of the two points that is outside the viewport region (there will be at least one point outside). The intersection of the outpoint and extended viewport border is then calculated (i.e. with the parametric equation for the line) and this new point replaces the outpoint. The algorithm repeats until a trivial accept or reject occurs.

The numbers in the figure below are called outcodes. An outcode is computed for each of the two points in the line. The first bit is set to 1 if the point is above the viewport. The bits in the outcode represent: Top, Bottom, Right, Left. For example the outcode 1010 represents a point that is top-right of the viewport. Note that the outcodes for endpoints must be recalculated on each iteration after the clipping occurs.

1001	1000	1010
0001	0000	0010
0101	0100	0110

ii. Nicholl–Lee–Nicholl

Using the Nicholl–Lee–Nicholl algorithm, the area around the clipping window is divided into a number of different areas, depending on the position of the initial point of the line to be clipped. This initial point should be in three predetermined areas; thus the line may have to be translated and/or rotated to bring it into the desired region. The line segment may then be re-translated and/or re-rotated to bring it to the original position. After that, straight line segments are drawn from the line end point, passing through the corners of the clipping window. These areas are then designated as L, LT, LB, or TR, depending on the location of the initial point. Then the other end point of the line is checked against these areas. If a line starts in the L area and finishes in the LT area then the algorithm concludes that the line should be clipped at x_w (max). Thus the number of clipping points is reduced to one, compared to other algorithms that may require two or more clipping points.

LCD Advantages and Disadvantages

Copyright © 1990-2011 by DisplayMate Technologies Corporation. All Rights Reserved.

This article, or any part thereof, may not be copied, reproduced, mirrored, distributed or incorporated

into any other work without the prior written permission of DisplayMate Technologies Corporation.

Although not as versatile as CRTs, LCDs are being seen as the preferred display for an increasing number of applications. They can produce very bright and sharp images but can also be harder to properly set up than CRTs. This article outlines their major pros and cons arranged in order of importance. For a more detailed discussion see [What Makes a Great LCD](#) and [Testing and Evaluating LCDs](#).

Note: most of the discussion here regarding LCDs also applies to DLP, Plasma and LCoS displays.

Principal LCD Advantages

1. **Sharpness**
Image is perfectly sharp at the native resolution of the panel. LCDs using an analog input require careful adjustment of pixel tracking/phase (see Interference, below).
2. **Geometric Distortion**
Zero geometric distortion at the native resolution of the panel. Minor distortion for other resolutions because the images must be rescaled.
3. **Brightness**
High peak intensity produces very bright images. Best for brightly lit environments.
4. **Screen Shape**
Screens are perfectly flat.
5. **Physical**
Thin, with a small footprint. Consume little electricity and produce little heat.

Principal LCD Disadvantages

1. Resolution

Each panel has a fixed pixel resolution format determined at the time of manufacture that can not be changed. All other image resolutions require rescaling, which generally results in significant image degradation, particularly for fine text and graphics. For most applications should only be used at the native resolution of the panel. If you need fine text and graphics at more than one resolution do not get an LCD display.

2. Interference

LCDs using an analog input require careful adjustment of pixel tracking/phase in order to reduce or eliminate digital noise in the image. Automatic pixel tracking/phase controls seldom produce the optimum setting. Timing drift and jitter may require frequent readjustments during the day. For some displays and video boards you may not be able to entirely eliminate the digital noise.

3. Viewing Angle

Limited viewing angle. Brightness, contrast, gamma and color mixtures vary with the viewing angle. Can lead to contrast and color reversal at large angles. Need to be viewed as close to straight ahead as possible.

4. Black-Level, Contrast and Color Saturation

LCDs have difficulty producing black and very dark grays. As a result they generally have lower contrast than CRTs and the color saturation for low intensity colors is also reduced. Not suitable for use in dimly lit and dark environments.

5. White Saturation

The bright-end of the LCD intensity scale is easily overloaded, which leads to saturation and compression. When this happens the maximum brightness occurs before reaching the peak of the gray-scale or the brightness increases slowly near the maximum. Requires careful adjustment of the Contrast control.

6. Color and Gray-Scale Accuracy

The internal Gamma and gray-scale of an LCD is very irregular. Special circuitry attempts to fix it, often with only limited success. LCDs typically produce fewer than 256 discrete intensity levels. For some LCDs portions of the gray-scale may

be dithered. Images are pleasing but not accurate because of problems with black-level, gray-scale and Gamma, which affects the accuracy of the gray-scale and color mixtures. Generally not suitable for professional image color balancing.

7.BadPixelsandScreenUniformity

LCDs can have many weak or stuck pixels, which are permanently on or off. Some pixels may be improperly connected to adjoining pixels, rows or columns. Also, the panel may not be uniformly illuminated by the backlight resulting in uneven intensity and shading over the screen.

8.MotionArtifacts

Slow response times and scan rate conversion result in severe motion artifacts and image degradation for moving or rapidly changing images.

9.AspectRatio

LCDs have a fixed resolution and aspect ratio. For panels with a resolution of 1280x1024 the aspect ratio is $5:4=1.25$, which is noticeably smaller than the $4:3=1.33$ aspect ratio for almost all other standard display modes. For some applications may require switching to a letterboxed 1280x960, which has a 4:3 aspect ratio.

10.Cost

Considerably more expensive than comparable CRTs.