INTRODUCTION TO COMPUTER GRAPHICS AND ANIMATION

ISIBOR O.O.
COMPUTER SCIENCE DEPARTMENT
LAGOS CITY POLYTECHNIC
IKEJA-NIGERIA.
osesisibor@gmail.com, 08063546421.
FEBRUARY 2017
REVISED FEBRUARY 2018.

1

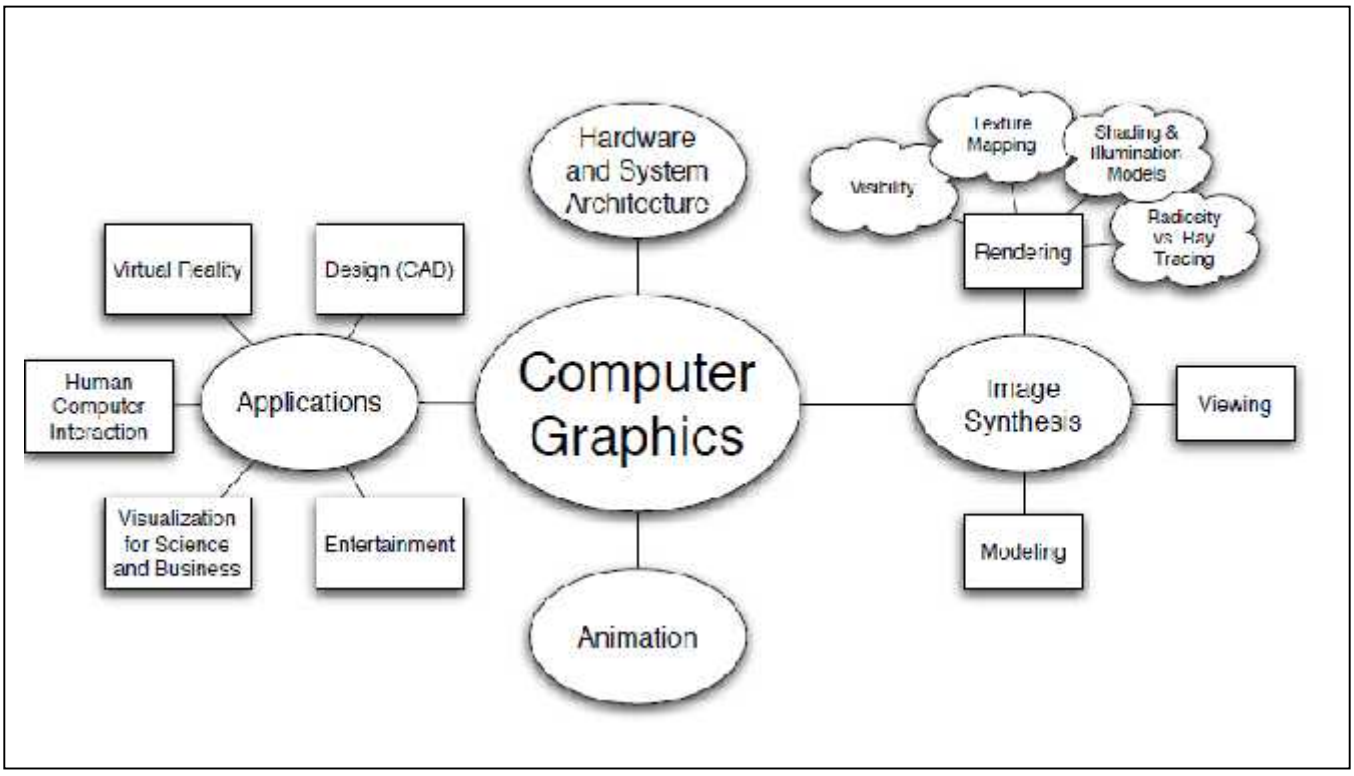
DEFINITION AND CONCEPTS OF COMPUTER GRAPHICS

Definition of Computer Graphics.

Computer graphics generally means creation, storage and manipulation of models and images. Such models come from diverse and expanding set of fields including physical, mathematical, artistic, biological, and even conceptual (abstract) structures.

"Perhaps the best way to define computer graphics is to find out **what it is not.** It is not a machine. It is not a computer, nor a group of computer programs. It is not the know-how of a graphic designer, a programmer, a writer, a motion picture specialist, or a reproduction specialist.

Computer graphics is all these –a consciously managed and documented technology directed toward communicating information accurately and descriptively.



History of Computer Graphics.

1. The Age of Sutherland

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

In 1963 Ivan Sutherland presented his paper Sketchpad at the Summer Joint Computer Conference.

Sketchpad allowed interactive design on a vector graphics display monitor with a light pen input device. Most people mark this event as the origins of computer graphics.

2. The Middle to Late '60's

Software and Algorithms

Jack Bresenham taught us how to draw lines on a raster device. He later extended this to circles. Anti-aliased lines and curve drawing is a major topic in computer graphics. **Larry Roberts** pointed out the usefulness of homogeneous coordinates, matrices and hidden line detection algorithms. **Steve Coons** introduced parametric surfaces and developed early computer aided geometric design concepts. The earlier work of **Pierre Bézier** on parametric curves and surfaces also became public. **Author Appel** at IBM developed hidden surface and shadow algorithms that were pre-cursors to ray tracing. The fast Fourier transform was discovered by **Cooley and Tukey.** This algorithm allow us to better understand signals and is fundamental for developing anti-aliasing techniques. It is also a precursor to wavelets.

Hardware and Technology

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

3. The Early '70's

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

Software and Algorithms

Rendering (shading) were discovered by **Gouraud and Phong** at the University of Utah. Phong also introduced a reflection model that included specular highlights. Keyframe based animation for 3-D graphics was demonstrated. XeroxY7U PARC developed a ``paint" program. Ed Catmull introduced parametric patch rendering, the z-buffer algorithm, and texture mapping. BASIC, C, and Unix were developed at Dartmouth and Bell Labs.

Hardware and Technology

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

4. The Middle to Late '70's

Software and Algorithms

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

Hardware and Technology

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

5. The Early '80's

Hardware and Technology

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

6. The Middle to Late '80's

Software and Algorithms

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

Hardware and Technology

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

7. The Early '90's

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

Software and Algorithms

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

Hardware and Technology

Hand-held computers were invented at Hewlett-Packard about 1991. Zip drives were invented at

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

8. The Middle to Late '90's

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

Software and Algorithms

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

Hardware and Technology

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

9. The '00's

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

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

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

ETHICAL ISSUES

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

APPLICATIONS OF COMPUTER GRAPHICS

1. Medical Imaging

There are few endeavors more noble than the preservation of life. Today, it can honestly be said that computer graphics plays an significant role in saving lives. The range of application spans from tools for teaching and diagnosis, all the way to treatment. Computer graphics is tool in medical applications rather than an a mere artifact. No cheating or tricks allowed.

2. Scientific Visualization

Computer graphics makes vast quantities of data accessible. Numerical simulations frequently produce millions of data values. Similarly, satellite-based sensors amass data at rates beyond our abilities to interpret them by any other means than visually. Mathematicians use computer graphics to explore abstract and high-dimensional functions and spaces. Physicists can use computer graphics to transcend the limits of scale. With it they can explore both microscopic and macroscopic world

3. Computer Aided Design

Computer graphics has had a dramatic impact on the design process. Today, most mechanical and electronic designs are executed entirely on computer. Increasingly, architectural and product designs are also migrating to the computer. Automated tools are also available that verify tolerances and design constraints directly from CAD designs. CAD designs also play a key role in a wide range of processes from the design of tooling fixtures to manufacturing.

4. Graphical User Interfaces (GUIs)

Computer graphics is an integral part of everyday computing. Nowhere is this fact more evident than the modern computer interface design. Graphical elements such as windows, cursors, menus, and icons are so common place it is difficult to imagine computing without them. Once graphics programming was considered a speciality. Today, nearly all professional programmers must have an understanding of graphics in order to accept input and present output to users.

5. Games

Games are an important driving force in computer graphics. In this class we are going to talk about games. We'll discuss on how they work. We'll also question how they get so much done with so little to work with.

6. Entertainment

If you can imagine it, it can be done with computer graphics. Obviously, Hollywood has caught on to this. Each summer, we are amazed by state- of-the-art special effects. Computer graphics is now as much a part of the entertainment industry as stunt men and makeup. The entertainment industry plays many other important roles in the field of computer graphics.

WHAT IS INTERACTIVE COMPUTER GRAPHICS?

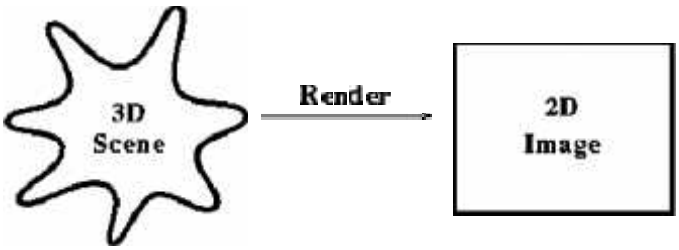
User controls contents, structure, and appearance of objects and their displayed images via rapid visual feedback.

Basic components of an interactive graphics system

1. Input (e.g., mouse, tablet and stylus, force feedback device, scanner, live video streams…),
2. Processing (and storage),
3. Display/Output (e.g., screen, paper-based printer, video recorder, non-linear editor.

2

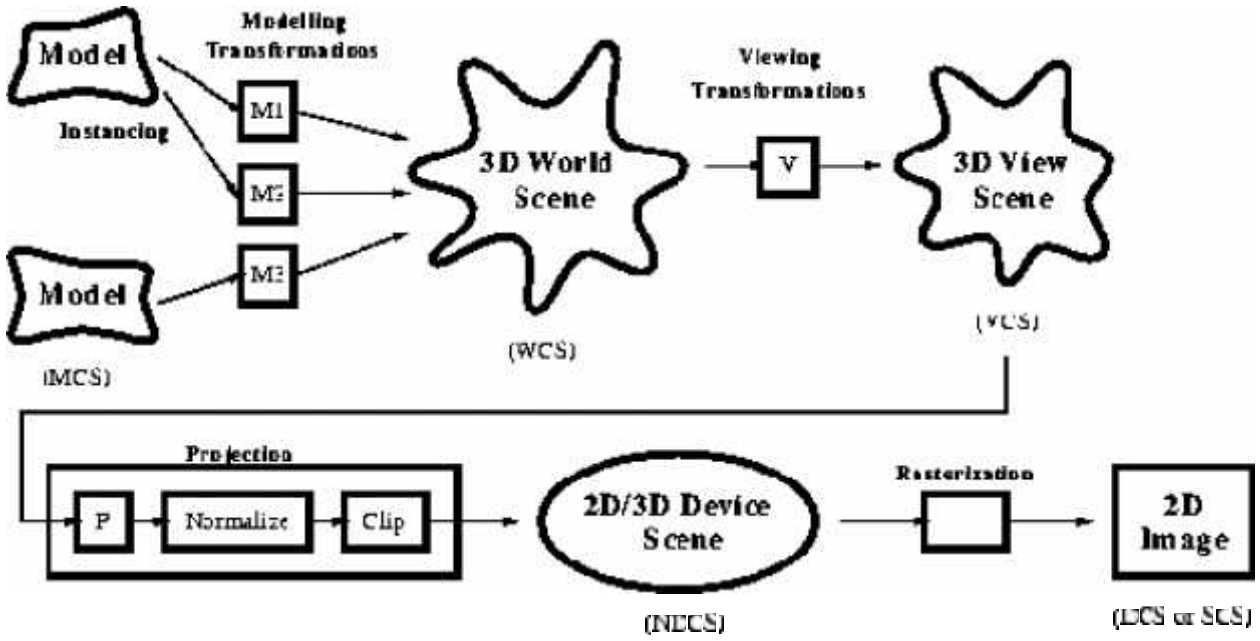
THE GRAPHICS RENDERING PIPELINE

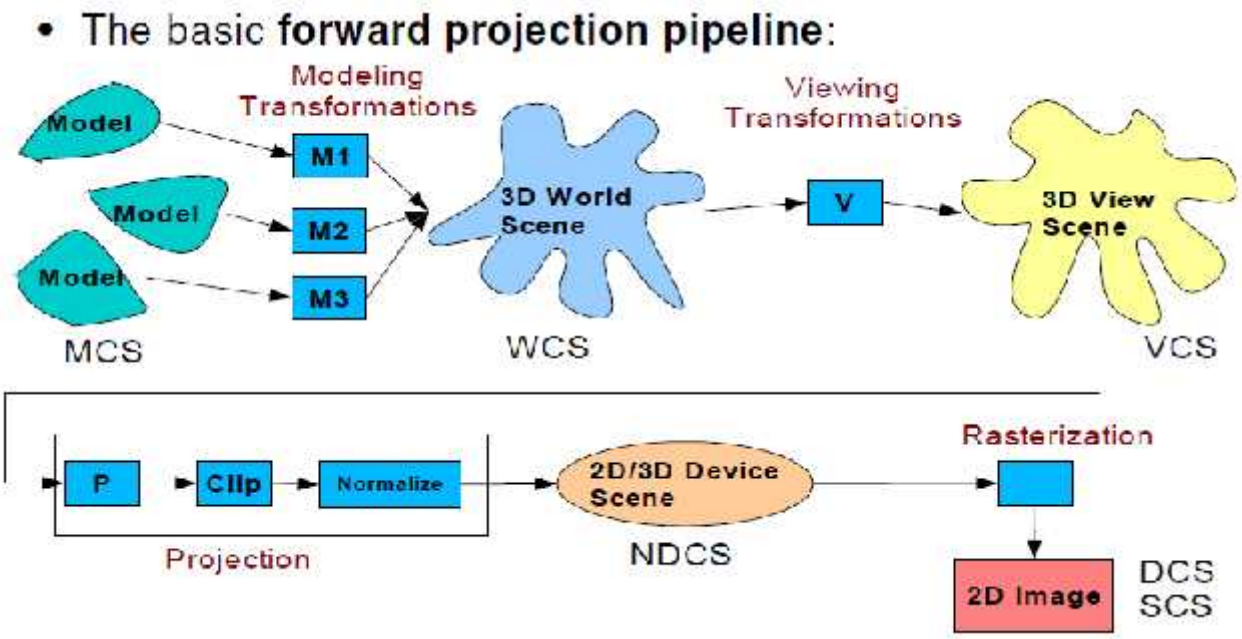


Rendering is the conversion of a scene into an image:

A **Scene** is composed of models in 3D space.
Models, composed of primitives, supported by the rendering system. Models are either entered by hand or created by a program.

Classically, "model" to "scene"' to "image" conversion broken into finer steps is called the **graphics pipeline** which is commonly implemented in graphics hardware to get interactive speeds. At a high level, the graphics pipeline usually looks like the diagram below:





Each stage refines the scene, converting primitives in modeling space to primitives in device space, where they are converted to pixels (rasterized). A number of coordinate systems are used:

MCS: Modeling Coordinate System.
WCS: World Coordinate System.
VCS: Viewer Coordinate System.
NDCS: Normalized Device Coordinate System.
DCS or **SCS**: Device Coordinate System or equivalently the Screen Coordinate System.

Keeping these straight is the key to understanding a rendering system. Transformation between two coordinate systems is represented with matrix. Derived information may be added (lighting and shading) and primitives may be removed (hidden surface removal) or modified (clipping).

3

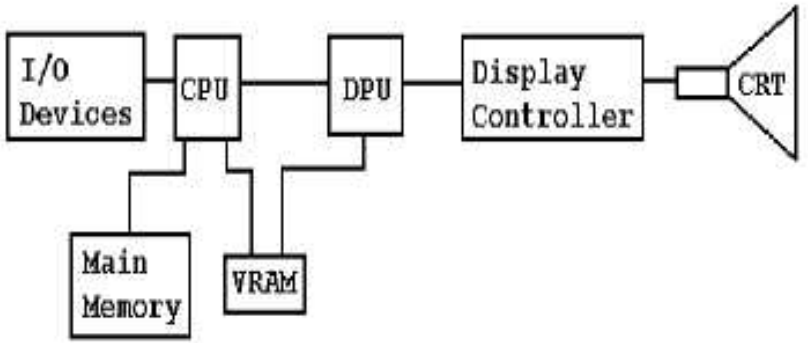
GRAPHICS HARDWARE, SOFTWARE AND DISPLAY DEVICES

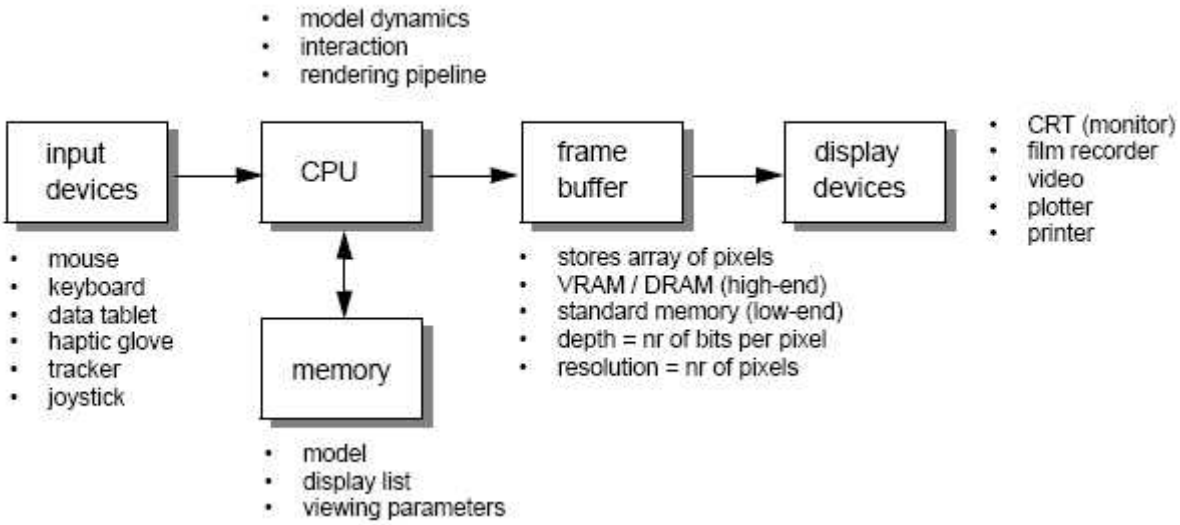
GRAPHICS SOFTWARE

Graphics software (that is, the software tool needed to create graphics applications) has taken the form of subprogram libraries. The libraries contain functions to do things like: draw points, lines, polygons apply transformations fill areas with color handle user interactions. An important goal has been the development of standard hardware- independent libraries such as:
CORE GKS (Graphical Kernel Standard)
PHIGS (Programmer's Hierarchical Interactive Graphics System)
X Windows OpenGL

(Study OpenGL)

GRAPHICS HARDWARE





Graphics Hardware Systems consist of the following:

A. **CPU**--Runs program in main memory and specifies what is to be drawn.
B. **CRT**--does the actual display.
C. **Display Controller**--Provides analog voltages needed to move beam and vary its intensity.
D. **DPU**—generates digital signals that drive display controller and offloads task of video control to separate processor.
E. **RAM**--Stores data needed to draw the picture and Dual-ported (written to by CPU, read from by DPU). It is Fast (e.g., 1000X1000, 50 Hz ==> 20 nsec access time!), and also called Refresh Buffer or Frame Buffer
F. **I/O devices**--interface CPU with user
    a. **Input devices can be described either by**
        i. **Physical properties:** Mouse, Keyboard, Trackball
        ii. **Logical Properties:** What is returned to program via API A position.
    b. **Input Devices are also categorized as follows**
        i. **String:** produces string of characters. (e.g keyboard)
        ii. **Valuator:** generates real number between 0 and 1.0 (e.g.knob)
        iii. **Locator:** User points to position on display (e.g. mouse)
        iv. **Pick:** User selects location on screen (e.g. touch screen in restaurant, ATM)

DISPLAY HARDWARE

An important component is the "refresh buffer" or "frame buffer" which is a random-access memory containing one or more values per pixel, used to drive the display.

The video controller translates the contents of the frame buffer into signals used by the **CRT** to illuminate the screen. It works as follows:

1. The display screen is coated with "phospors" which emit light when excited by an electron beam. (There are three types of phospor, emitting red, green, and blue light.) They are arranged in rows, with three phospor dots (R, G, and B) for each pixel.
2. The energy exciting the phosphors dissipates quickly, so the entire screen must be refreshed 60 times per second.
3. An electron gun scans the screen, line by line, mapping out a scan
4. Pattern. On each scan of the screen, each pixel is passed over once. Using the contents of the frame buffer, the controller controls the intensity of the beam hitting each pixel, producing a certain color.

FLAT-PANEL DISPLAYS:

This is the Technology used to replace CRT monitors.

- Characterized by Reduced volume, weight, power needs
- Thinner: can hang on a wall
- Higher resolution (High Definition).
- And comes in Two categories: **Emissive and Non-Emissive**

Emissive Flat Panel Display Devices (RASTER DEVICES)

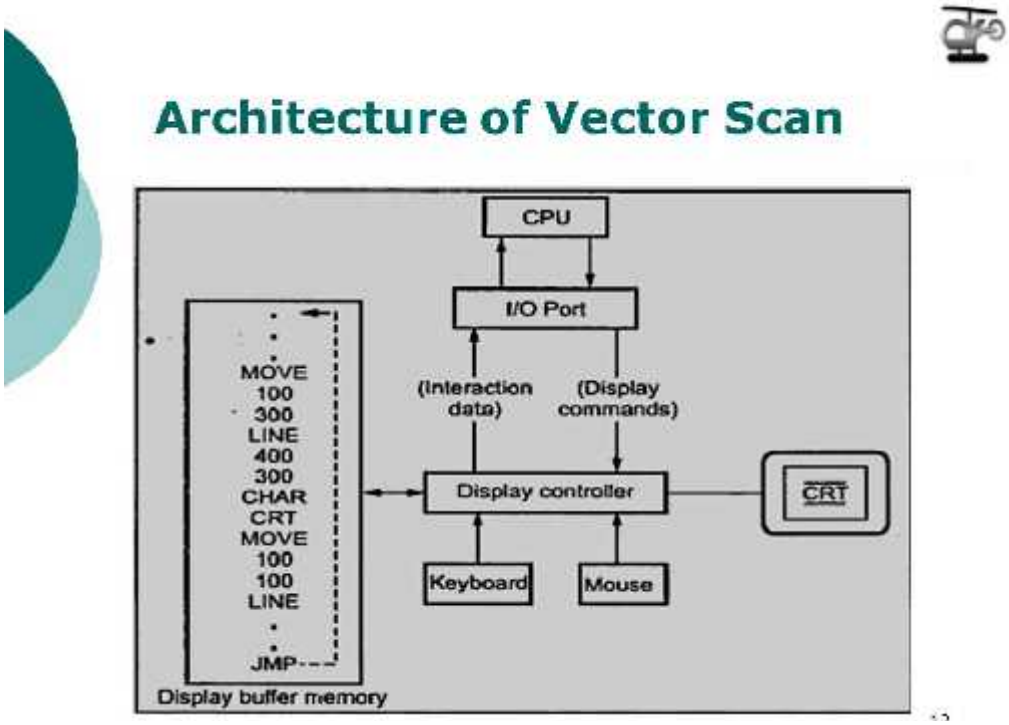
- Convert electrical energy to light.
- Uses Plasma panels (gas-discharge displays): that is –
    - Voltages fired to intersecting vertical/horizontal conductors cause gas to glow at that pixel
    - Resolution determined by density of conductors
    - Pixel selected by x-y coordinates of conductors
- These are "**raster**" **devices**

Other technologies require storage of x-y coordinates of pixels, e.g.: Thin-film electroluminescent displays, LEDs, Flat CRTs.

Non-Emissive Flat Panel Display Devices:

- Use optical effects to convert ambient light to pixel patterns,
- Example: LCDs
  - Pass polarized light from surroundings through liquid crystal material that can be aligned to block or transmit the light
  - Voltage applied to 2 intersecting conductors determines whether the liquid crystal blocks or transmits the light
- Like emissive devices, require storage of x-y coordinates of pixel to be illuminated

VECTOR SCAN SYSTEMS



Also called random, stroke, calligraphic displays, it possesses the following features:

i. Its images are drawn as line segments (vectors)
ii. Beam can be moved to any position on screen
iii. Refresh Buffer stores plotting commands so Frame Buffer often called "Display File" provides DPU with needed endpoint coordinates. Its pixel size is independent of frame buffer which gives a very high resolution

"**Vector graphics**" i.e. Early graphic devices were **line-oriented**. For example, a "pen plotter" from H-P. Image stored as line segments (vectors) that can be drawn anywhere on display device Primitive operation is line drawing.

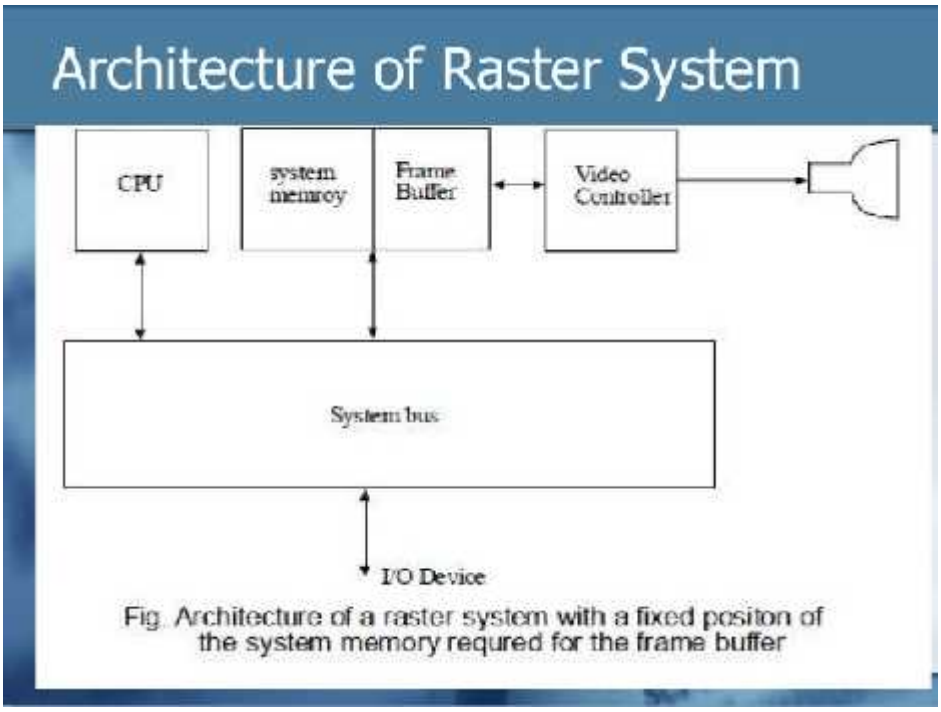
Advantages of Vector Scan

- High resolution (good for detailed line drawings)
- Crisp lines (no "jaggies")
- High contrast (beam can dwell on a single point for some time ==> very bright)
- Selective erase (remove commands from display file)
- Animation (change line endpoints slightly after each refresh)

Disadvantages of Vector Scan

- Complex drawings can have flicker Many lines so if time to draw > refresh time ==> flicker
- High cost--very fast deflection system needed
- Hard to get colors
- No area fill: so it's difficult to use for realistic (shaded) images
- 1960s Technology, only used for special purpose stuff today

RASTER SCAN SYSTEMS (TV Technology)



Beam continually traces a raster pattern. Its Intensity is adjusted as raster scan takes place

- In synchronization with beam
- Beam focuses on each pixel
- Each pixel's intensity is stored in frame buffer
- So resolution determined by size of frame buffer

Each pixel on screen visited during each scan, and Scan rate must be >= 30 Hz to avoid flicker

"**Raster graphics**" is today's standard. **A raster** is a 2-dimensional grid of **pixels (picture elements**). Image stored as a 2D array of color values in a memory area called the frame buffer. Each value stored determines the color/intensity of an accessible point on display device

Each pixel may be addressed and illuminated independently. So, the primitive operation is to draw a point; that is, assign a color to a pixel. Everything else is built upon that. There are a variety of raster devices, both hardcopy and display. ***Hardcopy:*** Laser printer, Ink-jet printer, Film recorder, Electrostatic printer, Pen plotter.

Scan Conversion here referes to the Process of determining which pixels need to be turned on in the frame buffer to draw a given graphics primitive. It need algorithms to efficiently scan convert graphics primitives like lines, circles, etc.

Advantages of Raster Scan Systems

- Low cost (TV technology)
- Area fill (entire screen painted on each scan)
- Colors
- Selective erase (just change contents of frame buffer)
- Bright display, good contrast but not as good as vector scan can be: this means it can't make beam dwell on a pixel

Disadvantages of Raster Scan Systems

- Large memory requirement for high resolution (but cost of VRAM has decreased a lot!)
- Aliasing (due to finite size of frame buffer)
  - – Finite pixel size
  - – Jagged lines (staircase effect)
  - – Moire patterns, scintillation, "creep" in animations
- Raster scan is the principal “now” technology for graphics displays!

DIFFERENCES BETWEEN A VECTOR SCAN DISPLAY AND A RASTER SCAN DISPLAY.

Vector Scan Display	Raster Scan Display
1. In vector scan display the beam is moved between the end points of the graphics primitives.	1. In raster scan display the beam is moved all over the screen one scan line at a time, from top bottom and then back to top,
2. Vector display flickers when the number of primitives in the buffer becomes too large.	2. In raster display, the refresh process is independent of the complexity of the image.
3. Scan conversion is not required.	3. Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixels in the frame buffer.
4. Scan conversion hardware is not required.	4. Because each primitive must be scan-converted,real time dynamics is for more computational and requires separate scan conversion hardware.
5. Vector display draws a continuous and smooth lines.	5. Raster display can display mathematically smooth lines, polygons, and boundaries of curved primitives only by approximating them with pixels on the raster grid.
6. Cost is more.	6. Cost is low.
7. Vector display only draws lines and characters.	7. Raster display has ability to display areas filled with solid colours or patterns.

CATHODE RAY TUBE (CRT)

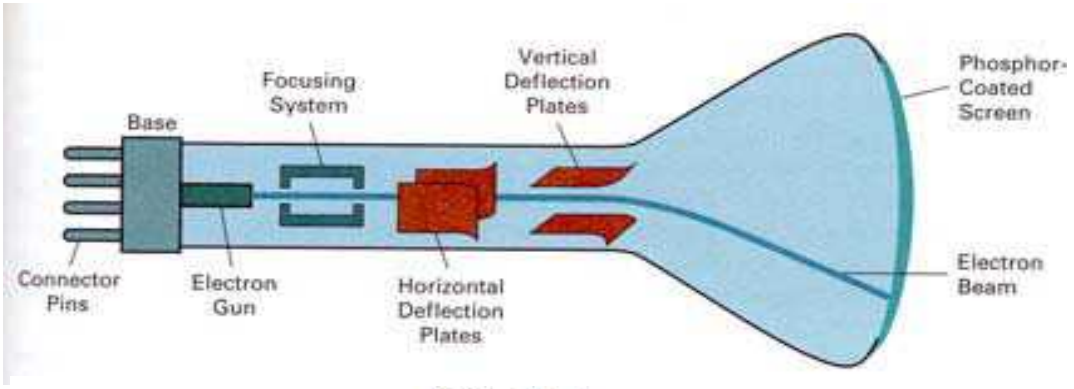
It Consists of:

i. Electron gun
ii. Electron focusing lens
iii. Deflection plates/coils
iv. Electron beam
v. Anode with phosphor coating

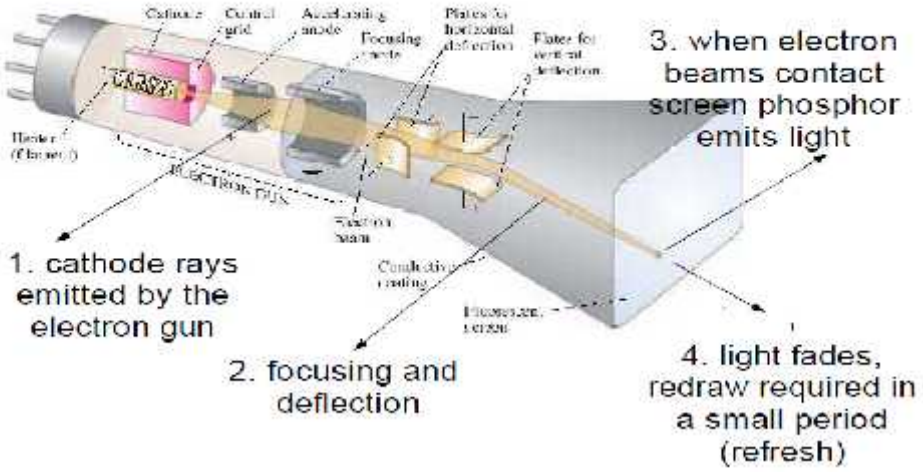
TYPES OF CRT

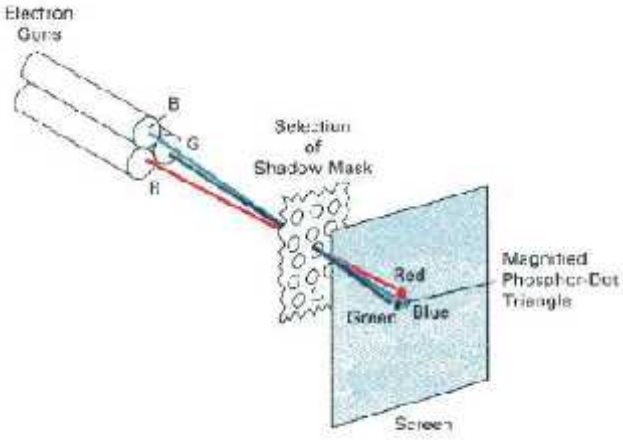
1. Direct View Storage Tubes (not CRT, no need for refresh, pictures stored as a permanent charge on phosphor screen)
2. Calligraphic refresh CRT (line drawing or vector random scan, need refreshing)
3. Rasterscan (point by point refreshing)

Refresh rate: # of complete images (frames) drawn on the screen in 1 second. Frames/sec.
Frame time: reciprocal of the refresh rate, time between each complete scan. sec/frame



CRT





Electron gun sends beam aimed (deflected) at a particular point on the screen, traces out a path on the screen, hitting each pixel once per cycle. “Scan lines”. Phosphors emit light (phosphoresence); output decays rapidly (exponentially - 10 to 60 microseconds) · As a result of this decay, the entire screen must be redrawn (refreshed) at least 60 times per second. This is called the refresh rate. If the refresh rate is too slow, we will see a noticeable flicker on the screen. CFF (Critical Fusion Frequency) is the minimum refresh rate needed to avoid flicker. This depends to some degree on the human observer. Also depends on the persistence of the phosphors; that is, how long it takes for their output to decay. The horizontal scan rate is defined as the number of scan lines traced out per second.

The most common form of CRT is the shadow-mask CRT. Each pixel consists of a group of three phosphor dots (one each for red, green, and blue), arranged in a triangular form called a triad. The

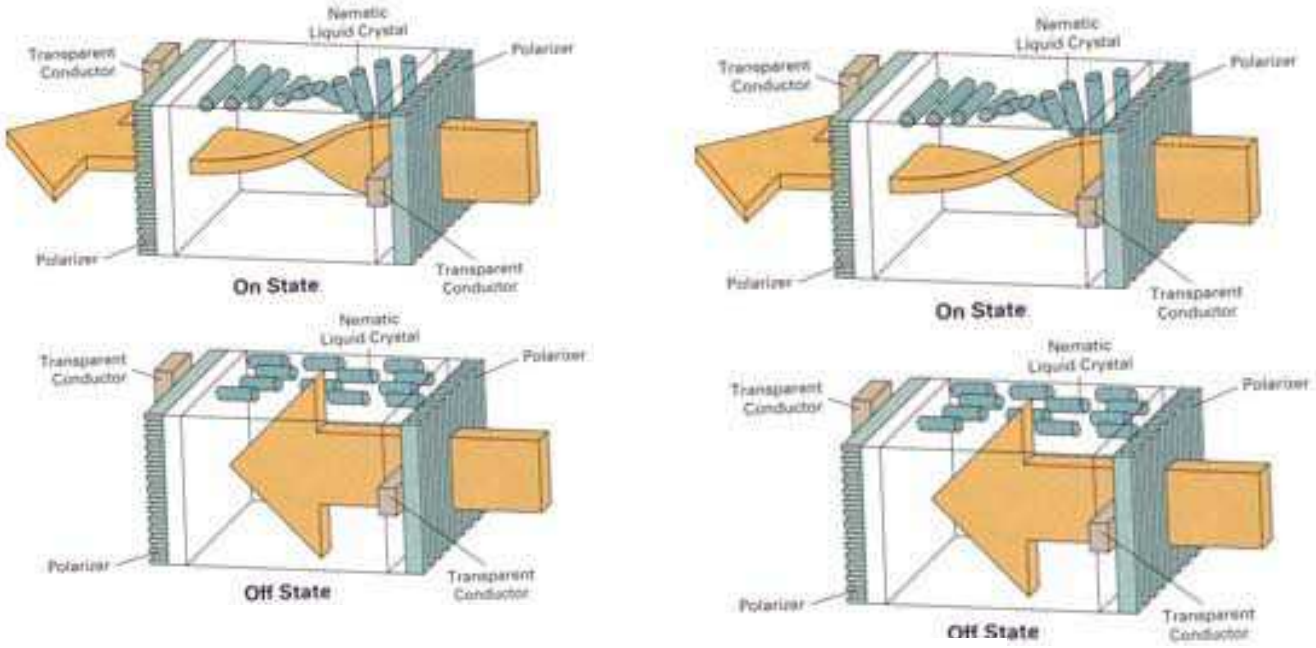
shadow mask is a layer with one hole per pixel. To excite one pixel, the electron gun (actually three guns, one for each of red, green, and blue) fires its electron stream through the hole in the mask to hit that pixel. The dot pitch is the distance between the centers of two triads. It is used to measure the resolution of the screen.

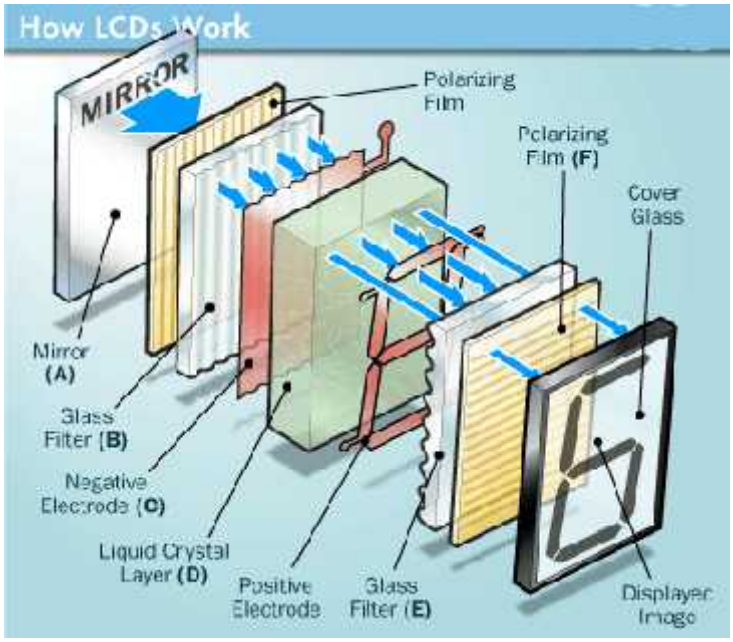
(Note: On a vector display, a scan is in the form of a list of lines to be drawn, so the time to refresh is dependent on the length of the display list.)

The popular display types:

– Liquid Crystal Display
– Plasma display
– Field Emission Displays
– Digital Meromirror Devices
– Light Emitting Diodes
– 3D display devices (hologram or page scan methods)

Liquid Crystal Display (LCD)





A liquid crystal display consists of **6 layers**, arranged in the following order (back-to-front):

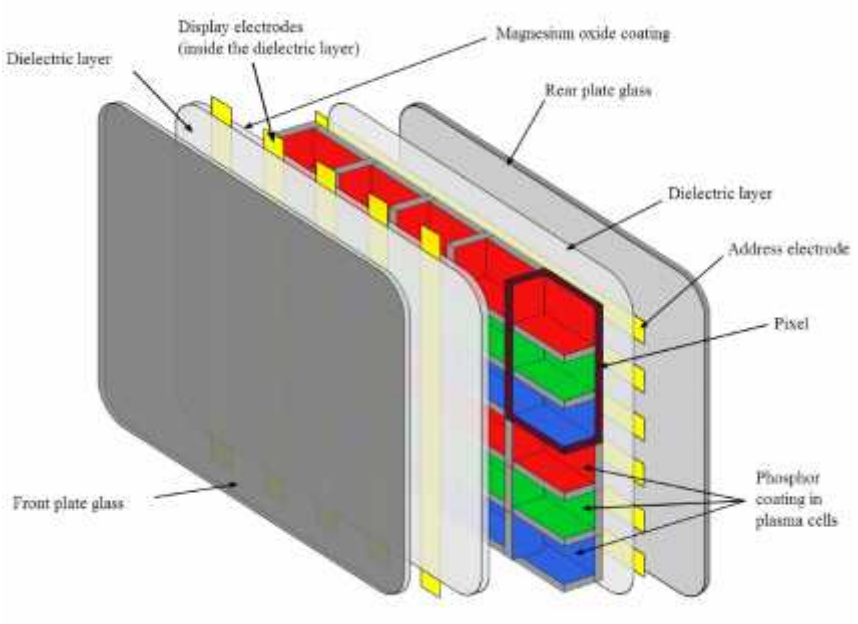
1. A reflective layer which acts as a mirror
2. A horizontal polarizer, which acts as a filter, allowing only the horizontal component of light to pass through
3. A layer of horizontal grid wires used to address individual pixels
4. The liquid crystal layer
5. A layer of vertical grid wires used to address individual pixels
6. A vertical polarizer, which acts as a filter, allowing only the vertical component of light to pass through

How it works:

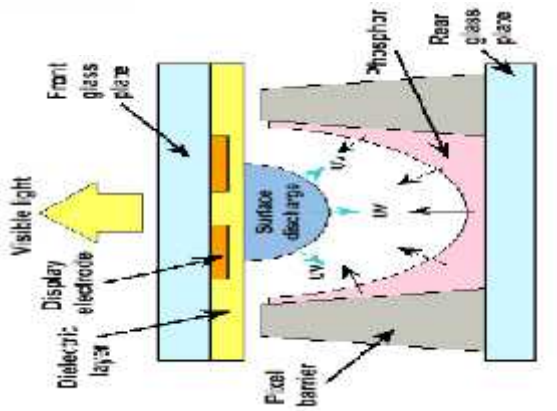
The liquid crystal rotates the polarity of incoming light by 90 degrees. Ambient light is captured, vertically polarized, rotated to horizontal polarity by the liquid crystal layer, passes through the horizontal filter, is reflected by the reflective layer, and passes back through all the layers, giving an appearance of lightness. However, if the liquid crystal molecules are charged, they become aligned and no longer change the polarity of light passing through them. If this occurs, no light can pass through the horizontal filter, so the screen appears dark.

The principle of the display is to apply this charge selectively to points in the liquid crystal layer, thus lighting or not lighting points on the screen. Crystals can be dyed to provide color. An LCD may be backlit, so as not to be dependent on ambient light. TFT (thin film transistor) is most popular LCD technology today.

Plasma Display Panels

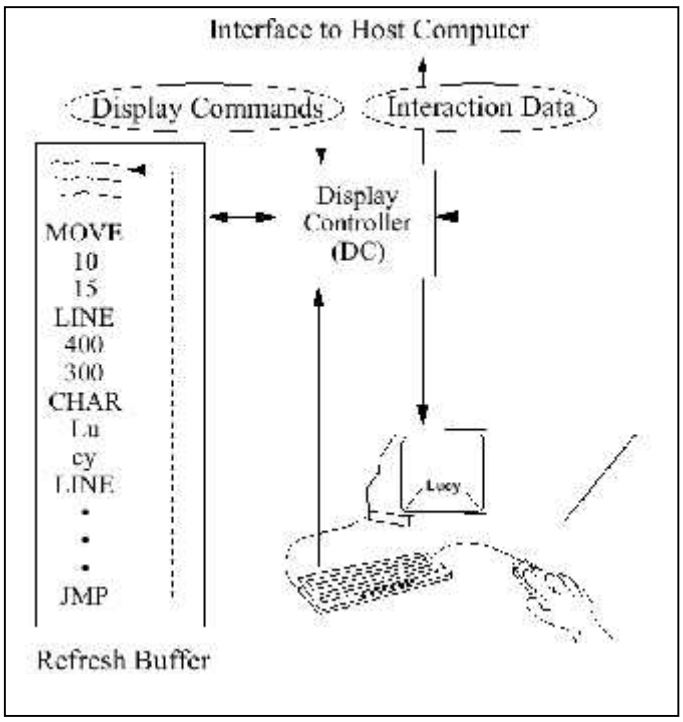


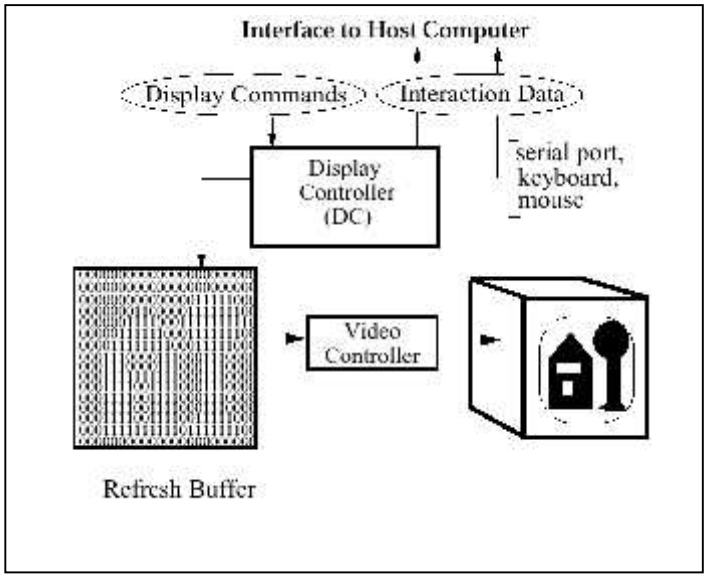
- Promising for large format displays
- Basically fluorescent tubes
- High-voltage discharge excites gas mixture (He, Xe) upon relaxation
- UV light is emitted
- UV light excites phosphors
- Large viewing angle



Vector Displays

Oscilloscopes were some of the 1st computer displays used by both analog and digital computers. Computation results used to drive the vertical and horizontal axis (X-Y). Intensity could also be controlled (Z-axis). Used mostly for line drawings Called **vector**, calligraphic or affectionately stroker displays. Display list had to be constantly updated (except for storage tubes).





Vector Architecture **Raster Architecture**

Raster display stores bitmap/pixmap in refresh buffer, also known as bitmap, frame buffer; be in separate hardware (VRAM) or in CPU's main memory (DRAM) Video controller draws all scan-lines at Consistent >60 Hz; separates update rate of the frame buffer and refresh rate of the CRT

Field Emission Devices (FEDs)

- Works like a CRT with multiple electron guns at each pixel uses modest voltages applied to sharp points to produce strong E fields
- Reliable electrodes proven difficult to produce
- Limited in size
- Thin, and requires a vacuum

Interfacing between the CPU and the Display

A typical video interface card contains a display processor, a frame buffer, and a video controller. The frame buffer is a random access memory containing some memory (at least one bit) for each pixel, indicating how the pixel is supposed to be illuminated. The depth of the frame buffer measures the number of bits per pixel. A video controller then reads from the frame buffer and sends control signals to the monitor, driving the scan and refresh process. The display processor processes software instructions to load the frame buffer with data.

(Note: In early PCs, there was no display processor. The frame buffer was part of the physical address space addressable by the CPU. The CPU was responsible for all display functions.)

Some Typical Examples of Frame Buffer Structures:

1. For a simple monochrome monitor, just use one bit per pixel.
2. A gray-scale monitor displays only one color, but allows for a range of intensity levels at each pixel. A typical example would be to use 6-8 bits per pixel, giving 64-256 intensity levels. For a color monitor, we need a range of intensity levels for each of red, green, and blue. There are two ways to arrange this.
3. A color monitor may use a color lookup table (LUT). For example, we could have a LUT with 256 entries. Each entry contains a color represented by red, green, and blue values. We then could use a frame buffer with depth of 8. For each pixel, the frame buffer contains an
4. Index into the LUT, thus choosing one of the 256 possible colors. This approach saves memory, but limits the number of colors visible at any one time.
5. A frame buffer with a depth of 24 has 8 bits for each color, thus 256 intensity levels for each color. 224 colors may be displayed. Any pixel can have any color at any time. For a 1024x1024 monitor we would need 3 megabytes of memory for this type of frame buffer. The display processor can handle some medium-level functions like scan conversion (drawing lines, filling polygons), not just turn pixels on and off. Other functions: bit block transfer, display list storage. Use of the display processor reduces CPU involvement and bus traffic resulting in a faster processor. Graphics processors have been increasing in power faster than CPUs, a new generation every 6-9 months, examples: 10 3E. NVIDIA GeForce FX
    - 125 million transistors (GeForce4: 63 million)
    - 128MB RAM
    - 128-bit floating point pipeline

One of the advantages of a hardware-independent API like OpenGL is that it can be used with a wide range of CPU-display combinations, from software-only to hardware-only. It also means that a fast video card may run slowly if it does not have a good implementation of OpenGL.

4

IMAGE REPRESENTATION

Introduction:

Computer Graphics is principally concerned with the generation of images, with wide ranging applications from entertainment to scientific visualization. In this unit, we begin our exploration of Computer Graphics by introducing the fundamental data structures used to represent images on modern computers. We describe the various formats for storing and working with image data, and for representing colour on modern machines.

The Digital Image

Virtually all computing devices in use today are digital; data is represented in a discrete form using patterns of **binary digits (bits)** that can encode numbers within finite ranges and with limited precision. By contrast, the images we perceive in our environment are analogue. They are formed by complex interactions between light and physical objects, resulting in continuous variations in light wavelength and intensity. Some of this light is reflected in to the retina of the eye, where cells convert light into nerve impulses that we interpret as a visual stimulus. Suppose we wish to 'capture' an image and represent it on a computer e.g. with a scanner or camera (the machine equivalent of an eye). Since we do not have infinite storage (bits), it follows that we must convert that analogue signal into a more limited digital form. We call this conversion process **sampling**. Sampling theory is an important part of Computer Graphics, underpinning the theory behind both image capture and manipulation.

Raster Image Representation

The Computer Graphics solution to the problem of image representation is to break the image (picture) up into a regular grid that we call a '**raster'**. Each grid cell is a 'picture cell', a term often contracted to **pixel**.

***Rasters** are used to represent digital images. Modern displays use a rectangular raster, comprised of W × H pixels. The raster illustrated here contains a greyscale image; its contents are represented in memory by a greyscale frame buffer.*

The values stored in the frame buffer record the intensities of the pixels on a discrete scale (0=black, 255=white).

The pixel is the atomic unit of the image; it is coloured uniformly, its single colour represents a discrete sample of light e.g. from a captured image.

In most implementations, rasters take the form of a rectilinear grid often containing many thousands of pixels.

The raster provides an orthogonal two-dimensional basis with which to specify pixel

coordinates.
By convention, pixels coordinates are zero-indexed and so the origin is located at the top-left of the image. Therefore pixel $(W-1, H-1)$ is located at the bottom-right corner of a raster of width W pixels and height H pixels. As a note, some Graphics applications make use of hexagonal pixels instead 1, however we will not consider these on the course.
The number of pixels in an image is referred to as the image's resolution.
Modern desktop displays are capable of visualizing images with resolutions around 1024×768 pixels (i.e. a million pixels or one mega-pixel). Even inexpensive modern cameras and scanners are now capable of capturing images at resolutions of several mega-pixels. In general, the greater the resolution, the greater the level of spatial detail an image can represent.

Resolution

A display's **"resolution"** is determined by:

i. number of scan lines (Each left-to-right trace)
ii. number of pixels (Each spot on the screen) per scan line
iii. number of bits per pixel

Resolution is used here to mean total number of bits in a display. It should really refer to the resolvable dots per unit length.

Examples:

Bitmapped display:	960 x 1152 x 1b	1/8 M B
NTSC TV:	640 x 480 x 1 6b	1/2 M B
Color workstation:	1280 x 1024 x 24b	4 MB
Laser-printed page:		
300 dpi:	8.5 x 11 x 300 x 1b	1 MB
1200 dpi:	8.5 x 11 x 1200 x 1b	17 MB
Film:	4500 x 3000 x 30b	50 MB

Frame aspect ratio = horizontal / vertical size

TV	4 : 3
HDTV	16 : 9
Letter-size paper	8.5 : 11 (about 3 : 4)
35mm film	3 : 2
Panavision	2.35 : 1

Pixel aspect ratio = pixel width / pixel height (nowadays, this is almost always 1.)

Hardware Frame Buffers

We represent an image by storing values for the colour of each pixel in a structured way. Since the earliest computer Visual Display Units (VDUs) of the 1960s, it has become common practice to reserve a large, contiguous block of memory specifically to manipulate the image currently shown on the computer's display. This piece of memory is referred to as a **frame buffer**. By reading or writing to this region of memory, we can read or write the colour values of pixels at particular positions on the display.
Note that the term 'frame buffer' as originally defined, strictly refers to the area of memory reserved for direct manipulation of the currently displayed image. In the early days of Graphics, special

hardware was needed to store enough data to represent just that single image. However, we may now manipulate hundreds of images in memory simultaneously and the term 'frame buffer' has fallen into informal use to describe any piece of storage that represents an image.

There are a number of popular formats (i.e. ways of encoding pixels) within a frame buffer. This is partly because each format has its own advantages, and partly for reasons of backward compatibility with older systems (especially on the PC). Often video hardware can be switched between different video modes, each of which encodes the frame buffer in a different way.

We will describe three common frame buffer formats in the subsequent sections; **the greyscale, pseudo-colour, and true-colour formats**. If you do Graphics, Vision or mainstream Windows GUI programming then you will likely encounter all three in your work at some stage.

A. Greyscale Frame Buffer

Arguably the simplest form of frame buffer is the greyscale frame buffer; often mistakenly called 'black and white' or 'monochrome' frame buffers. **Greyscale buffers encodes pixels using various shades of grey**. In common implementations, pixels are encoded as an unsigned integer using 8 bits (1 byte) and so can represent 28 = 256 different shades of grey. Usually black is represented by value 0, and white by value 255. A mid-intensity grey pixel has value 128. Consequently, an image of width W pixels and height H pixels requires W ×H bytes of memory for its frame buffer.

The frame buffer is arranged so that the first byte of memory corresponds to the pixel at coordinates (0, 0). Recall that this is the top- left corner of the image. Addressing then proceeds in a left-right, then top-down manner (see Figure). So, the value (grey level) of pixel (1, 0) is stored in the second byte of memory, pixel (0, 1) is stored in the (W + 1) th byte, and so on. Pixel (x, y) would be stored at buffer offset A where:

$A = x + Wy$ (2.1) i.e. A byte from the start of the frame buffer. Sometimes we use the term scan line to refer to a full row of pixels. A scan-line is therefore W pixels wide.

Old machines, such as the ZX Spectrum, required more CPU time to iterate through each location in the frame buffer than it took for the video hardware to refresh the screen. In an animation, this would cause undesirable flicker due to partially drawn frames. To compensate, byte range [0, (W − 1)] in the buffer wrote to the first scan-line, as usual. However, bytes [2W, (3W −1)] wrote to a scan-line one third of the way down the display, and [3W, (4W −1)] to a scan-line two thirds down. This interleaving did complicate Graphics programming but prevented visual artifacts that would otherwise occur due to slow memory access speeds.

B. Pseudo-colour Frame Buffer

The pseudo-colour frame buffer allows representation of colour images. The storage scheme is identical to the greyscale frame buffer. However, the pixel values do not represent shades of grey. Instead each possible value (0 − 255) represents a particular colour; more specifically, an index into a list of 256 different colours maintained by the video hardware.

The colours themselves are stored in a "**Colour Lookup Table" (CLUT)** which is essentially a map < colourindex, colour > i.e. a table indexed with an integer key (0−255) storing a value that represents colour. In alternative terminology the CLUT is sometimes called a **palette**. As we will discuss in greater detail shortly, many common colours can be produced by adding together

(mixing) varying quantities of Red, Green and Blue light.

For example, Red and Green light mix to produce Yellow light. Therefore the value stored in the CLUT for each colour is a triple (R,G,B) denoting the quantity (intensity) of Red, Green and Blue light in the mix. Each element of the triple is 8 bit i.e. has range (0 − 255) in common implementations.

The earliest colour displays employed pseudo-colour frame buffers. This is because memory was expensive and colour images could be represented at identical cost to grayscale images (plus a small storage overhead for the CLUT). **The obvious disadvantage of a pseudocolour frame buffer** is that only a limited number of colours may be displayed at any one time (i.e. **256 colours**). However the colour range (we say gamut) of the display is 28 × 28 × 28 = 224 = 16, 777, 216 colours. Pseudo-colour frame buffers can still be found in many common platforms e.g. both MS and X Windows (for convenience, backward compatibility etc.) and in resource constrained computing domains (e.g. low-spec games consoles, mobiles). Some low-budget (in terms of CPU cycles) animation effects can be produced using pseudo-colour frame buffers. Consider an image filled with an expanses of colour index 1 (we might set CLUT < 1,Blue >, to create a blue ocean). We could sprinkle consecutive runs of pixels with index '2,3,4,5' sporadically throughout the image. The CLUT could be set to increasing, lighter shades of Blue at those indices. This might give the appearance of waves. The colour values in the CLUT at indices 2,3,4,5 could be rotated successively, so changing the displayed colours and causing the waves to animate/ripple (but without the CPU overhead of writing to multiple locations in the frame buffer). Effects like this were regularly used in many '80s and early '90s computer games, where computational expense prohibited updating the frame buffer directly for incidental animations.

C. True-Colour Frame Buffer

The true-colour frame-buffer also represents colour images, but does not use a CLUT. The RGB colour value for each pixel is stored directly within the frame buffer. So, if we use 8 bits to represent each Red, Green and Blue component, we will require 24 bits (3 bytes) of storage per pixel.

As with the other types of frame buffer, pixels are stored in left-right, then top-bottom order. So in our 24 bit colour example, pixel (0, 0) would be stored at buffer locations 0, 1 and 2.Pixel (1, 0) at 3, 4, and 5; and so on. Pixel (x, y) would be stored at offset A

where: S = 3W

A = 3x + Sy where S is sometimes referred to as the stride of the display.

The advantages of the true-colour buffer complement the disadvantages of the pseudo-colour buffer we can represent all 16 million colours at once in an image (given a large enough image!), but our image takes 3 times as much storage as the pseudo-colour buffer. The image would also take longer to update (3 times as many memory writes) which should be taken under consideration on resource constrained platforms (e.g. if writing a video codec on a mobile phone).

Alternative forms of true-colour buffer

The true colour buffer, as described, uses 24 bits to represent RGB colour. The usual convention is to write the R, G, and B values in order for each pixel. Sometime image formats (e.g. Windows Bitmap) write colours in order B, G, R. This is primarily due to the little-endian hardware architecture of PCs, which run Windows. These formats are sometimes referred to as RGB888 or BGR888 respectively.

5

GEOMETRIC MODELING.

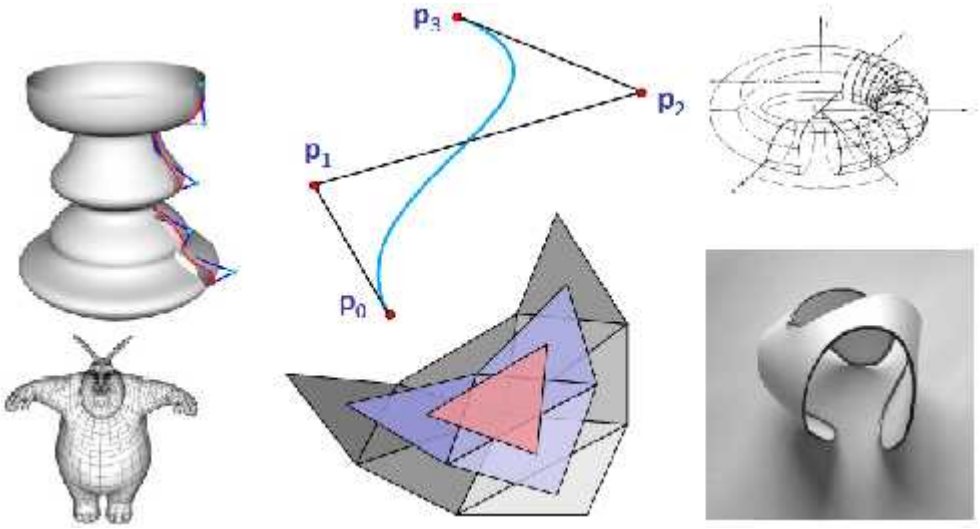
In computer graphics we work with points and vectors defined in terms of some coordinate frame (a positioned coordinate system). We also need to change coordinate representation of points and vectors, hence to transform between different coordinate frames.

There are many ways for creating graphical data. The Classic way is **Geometric Modeling.**
Other approaches are:

- **3D scanners**
- Photography for measuring optical properties
- Simulations, e.g., for flow data

Geometric Modeling is the computer-aided design and manipulation of geometric objects. (CAD). It is the basis for:

- Computation of geometric properties
- Rendering of geometric objects
- Physics computations (if some physical attributes are given)



Geometric objects convey a part of the real or theoretical world; often, something tangible. They are described by their **geometric** and **topological properties**:

- **Geometry** describes the form and the position/orientation in a coordinate system.
- **Topology** defines the fundamental structure that is invariant against continuous transformations.

Different geometry
Same topology

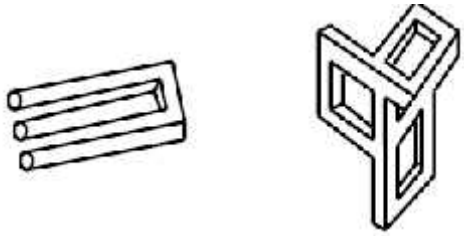
Different geometry
Different topology

3D models are geometric representations of 3D objects with a certain level of abstraction. Let's distinguish between **three types of models**:

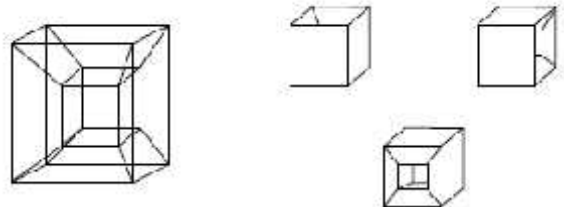
1) **Wire Frame Models**: describe an object using boundary lines. No relationship exist between these curves and surfaces between them are not defined

Properties of wire frame models:

i. Simple, traditional
ii. Non-sense objects possible
iii. Visibility of curves cannot be decided
iv. Solid object intersection cannot be computed
v. Surfaces between the curves cannot be computed automatically
vi. Not useable for CAD/CAM



Non-sense objects

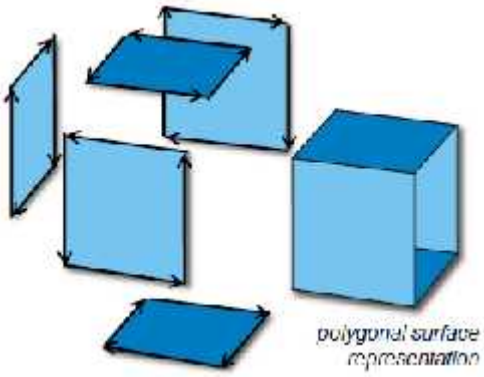


Ambiguity of wire frame models

2) **Surface Models**: describe an object using boundary surfaces

Properties of Surface models:

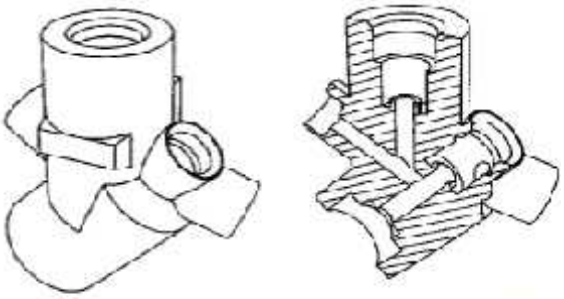
i. Describes the hull, but not the interior of an object
ii. Often implemented using polygons, hull of a sphere or ellipsoid, free-form surfaces, …
iii. No relationship between the surfaces
iv. The interior between them is not defined
v. Visibility computations is present. Solid intersection comp. not present.
vi. Most often used type of model



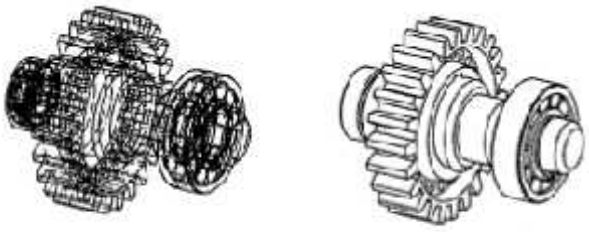
3) **Solid Models**: describe an object as a solid, that is, it describe the 3D object completely by covering the solid

Properties of Surface models:

i. For every point in 3D, we can decide whether it is inside or outside of the solid.
ii. Visibility and intersection computations are fully supported
iii. Basis for creating solid objects using computer-aided manufacturing



solid model and a cut through it (Werkbild Strässle, from Ockert, 1993)

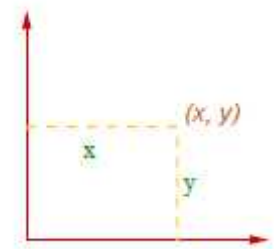


visibility computation for lines using a solid model

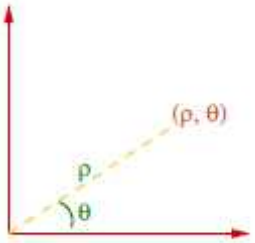
CARTESIAN COORDINATE SYSTEM:

Point Representation in two dimensions come in two forms:

1. Cartesian Coordinates: (x, y)
2. Polar Coordinates: (ρ, Θ)



Cartesian Coordinates

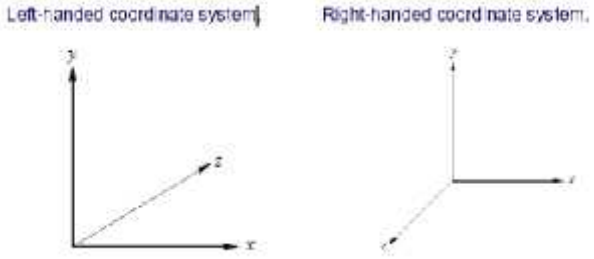


Polar Coordinates

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

A Cartesian coordinate system is an orthogonal coordinate system with lines as coordinate axes. **A Cartesian coordinate frame** is a Cartesian coordinate system positioned in space.

Right/Left-handed coordinate system



Vectors

A vector u; v;w is a directed line segment (no concept of position). Vectors are represented in a coordinate system by a n-tuple $v = (v_1; : : : ; v_n)$.

The dimension of a vector is dim(v) = n.

Length jvj and direction of a vector is invariant with respect to choice of Coordinate system.

Points, Vectors and Notation

Much of Computer Graphics involves discussion of points in 2D or 3D. Usually we write such points as Cartesian Coordinates e.g. $p = [x, y]^T$ or $q = [x, y, z]^T$ ·Point coordinates are therefore vector quantities, as opposed to a single number e.g. 3 which we call a scalar quantity. In these notes we write vectors in bold and underlined once. Matrices are written in bold, double-underlined.

The superscript $[...]^T$ denotes transposition of a vector, so points p and q are column vectors (coordinates stacked on top of one another vertically). This is the convention used by most researchers with a Computer Vision background, and is the convention used throughout this course. By contrast, many Computer Graphics researchers use row vectors to represent points. For this reason you will find row vectors in many Graphics textbooks including Foley et al, one of the course texts. Bear in mind that you can convert equations between the two forms using transposition. Suppose we have a 2 × 2 matrix M acting on the 2D point represented by column vector p. We would write this as Mp.

If p was transposed into a row vector $p' = p^T$, we could write the above transformation $p'M^T$. So to convert between the forms (e.g. from row to column form when reading the course-texts), remember that: $Mp = (p^TM^T)^T$

MANY COMPUTER GRAPHICS TOPICS USE LINEAR ALGEBRA

To name a few:

– Dot (aka scalar, inner) product of two vectors: can be used to measure angles

– cross (aka vector) product of two vectors: can be used to find the direction perpendicular to a plane (the one that the two vectors live on)

– matrix multiplication: matrices often represent some geometric transformation; multiplying two together yields a matrix that represents the composite of the two transformations (as if one transformation is applied and afterwards the other one is applied.) can be used to project a 3d object onto a 2d plane using perspective (or some other) projection

– matrix multiplied by a vector: allows some transform (represented by the matrix) to be applied to a set of points (represented by the vectors). The set of points can be the positions of the corners of a polygon (e.g. a triangle, an octagon) or a polyhedron (e.g. a cube).

Matrix Algebra

A matrix is a rectangular array of numbers. Both vectors and scalars are degenerate forms of matrices. By convention we say that an (n×m) matrix has n rows and m columns; i.e. we write (height × width). In this subsection we will use two 2 × 2 matrices for our examples:

$$\begin{aligned} \underline{\underline{A}} + \underline{\underline{B}} &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ &= \begin{bmatrix} (a_{11} + b_{11}) & (a_{12} + b_{12}) \\ (a_{21} + b_{21}) & (a_{22} + b_{22}) \end{bmatrix} \end{aligned}$$

Observe that the notation for addressing an individual element of a matrix is x row, column.

Matrix Addition

Matrices can be added, if they are of the same size. This is achieved by summing the elements in one matrix with corresponding elements in the other matrix:

This is identical to vector addition.

$$\underline{\underline{A}} + \underline{\underline{B}} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} (a_{11}+b_{11}) & (a_{12}+b_{12}) \\ (a_{21}+b_{21}) & (a_{22}+b_{22}) \end{bmatrix}$$

Matrix Scaling

Matrices can also be scaled by multiplying each element in the matrix by a scale factor.
Again, this is identical to vector scaling.

$$s\underline{\underline{A}} = \begin{bmatrix} sa_{11} & sa_{12} \\ sa_{21} & sa_{22} \end{bmatrix}$$

Matrix Multiplication

As we will see later, matrix multiplication is a cornerstone of many useful geometric transformations in Computer Graphics. You should ensure that you are familiar with this operation.

$$\underline{\underline{AB}} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} (a_{11}b_{11}+a_{12}b_{21}) & (a_{11}b_{12}+a_{12}b_{22}) \\ (a_{21}b_{11}+a_{22}b_{21}) & (a_{21}b_{21}+a_{22}b_{22}) \end{bmatrix}$$

In general each element c_{ij} of the matrix C = AB, where A is of size (n×P) and B is of size (P × m) has the form:

$$c_{ij} = \sum_{k=1}^{P} a_{ik} b_{kj}$$

Not all matrices are compatible for multiplication. In the above system, A must have as many columns as B has rows. Furthermore, matrix multiplication is non-commutative, which means that BA 6= AB, in general. Given equation 1.27 you might like to write out the multiplication for BA to satisfy yourself of this.

Finally, matrix multiplication is associative i.e.: ABC = (AB)C = A(BC)

If the matrices being multiplied are of different (but compatible) sizes, then the complexity of evaluating such an expression varies according to the order of multiplication1.

Matrix Inverse and the Identity

The identity matrix I is a special matrix that behaves like the number 1 when multiplying scalars (i.e. it has no numerical effect):

IA = A

The identity matrix has zeroes everywhere except the leading diagonal which is set to 1, e.g. the (2 × 2) identity ma

$$\underline{\underline{I}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

1Finding an optin of multiplication order is a (solved) interesting optimization problem, but falls outside the scope of this course.

The identity matrix leads us to a definition of the inverse of a matrix, which we write A^{-1}. The inverse of a matrix, when pre- or post- multiplied by its original matrix, gives the identity:

$$AA^{+}-1 = A^{-1}A = I$$

As we will see later, this gives rise to the notion of reversing a geometric transformation. Some geometric transformations (and matrices) cannot be inverted. Specifically, a matrix must be square and have a non-zero determinant in order to be inverted by conventional means.

1.7.5 Matrix Transposition

Matrix transposition, just like vector transposition, is simply a matter of swapping the rows and columns of a matrix. As such, every matrix has a transpose. The transpose of A is written A^T :

$$\underline{\underline{A}}^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}$$

For some matrices (the orthonormal matrices), the transpose actually gives us the inverse of the matrix. We decide if a matrix is orthonormal by inspecting the vectors that make up the matrix's columns, e.g. $[a_{11}, a_{21}]$T and $[a_{12}, a_{22}]^T$. These are sometimes called column vectors of the matrix. If the magnitudes of all these vectors are one, and if the vectors are orthogonal (perpendicular) to each other, then the matrix is orthonormal. Examples of orthonormal matrices are the identity matrix, and the rotation matrix that we will meet in subsequent classes.

6

GRAPHICS RENDERING: TRANSFORMATION

In Computer Graphics we most commonly model objects using points, i.e. locations in 2D or 3D space. For example, we can model a 2D shape as a polygon whose vertices are points. By manipulating the points, we can define the shape of an object, or move it around in space.
In 3D too, we can model a shape using points. Points might define the locations (perhaps the corners) of surfaces in space. In this unit, we will describe how to manipulate models of objects and display them on the screen.

Transformation

Transformations are often considered to be one of the hardest concepts in elementary computer graphics. But transformations are straightforward, as long as you

- •Have a clear representation of the geometry
- •Understand the underlying mathematics
- •Are systematic about concatenating transformations

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

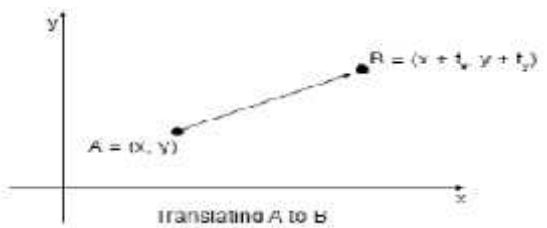
1. Change coordinate frames (world, window, viewport, device, etc).
2. Compose objects of simple parts with local scale/position/orientation of one part defined with regard to other parts. For example, for articulated objects.
3. Use deformation to create new shapes.

A. TRANSLATION (2D)

This is a transformation on an object that simply moves it to a different position somewhere else within the same coordinate system. To translate an object, we translate each of its vertices (points). It involves moving an object along a line from one location to another.
To translate the point (x_1 , y_1) by t_x in x and t_y in y
– result is (x_2, y_2) So, $(x_2, y_2) = (x_1 + t_x, y_1 + t_y)$
• Translations can be represented by adding vectors.

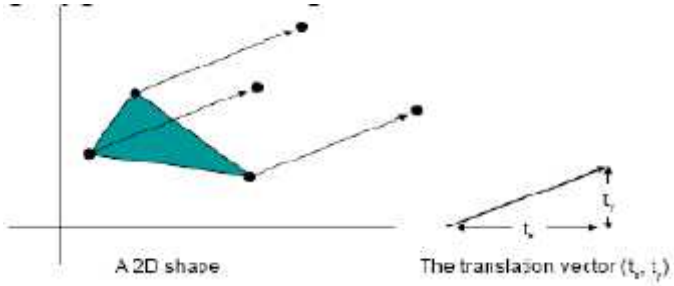
$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x_1 + t_x \\ y_1 + t_y \end{bmatrix}$$

Suppose we want to move a point from A to B e.g, the vertex of a polygon. This operation is called a translation
To translate point A by (t_x, t_y), we add (t_x, t_y) to A's coordinates



To translate a 2D shape by (t_x, t_y)

•Translate each point that defines the shape e.g., each vertex of a polygon, the center point of a circle, the control points of a curve

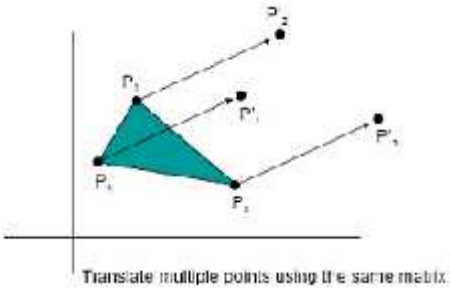


Translation by (t_x, t_y) moves each object point by (t_x, t_y) $(x, y) \rightarrow (x + t_x, y + t_y)$

Translation is a linear operation. The new coordinates are a linear combination of the previous coordinates, the new coordinates are determined from a linear system

$x' = x + t_x$

$y' = y + t_y$



Hence, translations can be expressed using matrix notation

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

for (x, y) = P_1, P_2, and P_3

Using matrices to represent transformations is convenient e.g., if the transformation is part of an animation

$$\begin{pmatrix} x(k) \\ y(k) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + (k/N) \begin{pmatrix} t_x \\ t_y \end{pmatrix} \text{ , for } k = 0, 1, 2, \ldots N$$

B. Rotation (2D)

Rotation is a transformation on an object that changes its position by rotating the object some angle about some axis. **Rotations** in the x-y plane are about an axis parallel to z. The point of intersection of the rotation axis with the x-y plane is the **pivot point**. We need to specify the **angle** and **pivot point** about which the object is to be rotated.

• To rotate an object, we rotate each of its vertices (points).

• Positive angles are in the counterclockwise direction.

Let's derive the rotation matrix.

• To rotate a 2D point (x_1, y_1) an arbitrary angle of B, about the origin as a pivot point do the following.

• From the diagram below, we have:

$\sin(A + B) = y_2 / r$ => $y_2 = r \sin(A + B)$

$\cos(A + B) = x_2 / r$ => $x_2 = r \cos(A + B)$

$\sin(A) = y_1 / r$ => $y_1 = r \sin(A)$

$\cos(A) = x_1 / r$ => $x_1 = r \cos(A)$

• Known equalities exist for sin(A+B) and cos(A+B)

$\sin(A + B) = \sin(A) \cos(B) + \cos(A) \sin(B)$

$\cos(A + B) = \cos(A) \cos(B) – \sin(A) \sin(B)$

• Solve for x_2 and y_2.

• $x_2 = r \cos(A + B) = r \cos(A) \cos(B) – r \sin(A) \sin(B) = x_1 \cos(B) – y_1 \sin(B)$

• $y_2 = r \sin(A + B) = r \sin(A) \cos(B) + r \cos(A) \sin(B) = y_1 \cos(B) + x1 \sin(B)$

• So, $(x_2, y_2) = (x_1 \cos(B) – y_1 \sin(B) , y_1 \cos(B) + x_1 \sin(B))$

• This will rotate a point (x_1, y_1) an angle of B about the pivot point of the origin.

*Rotate (x_1 , y_1) by some angle B counterclockwise, the result is (x_2, y_2)

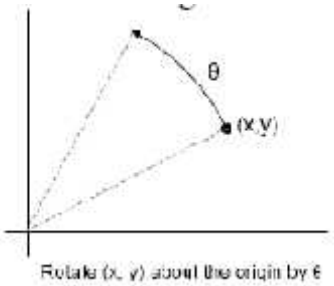
$(x_2, y_2) = (x_1 * \cos(B) – y_1 * \sin(B), y_1 * \cos(B) + x_1 * \sin(B))$

• Rotation can be represented by matrix multiplication:

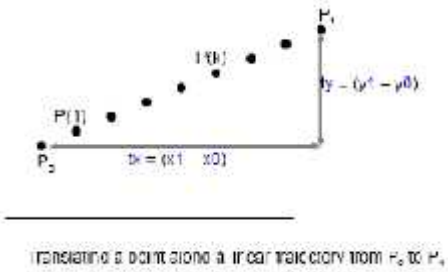
$\begin{pmatrix} x_2 \end{pmatrix} = \begin{pmatrix} \cos(B) & -\sin(B) \end{pmatrix} \begin{pmatrix} x_1 \end{pmatrix}$

$\begin{pmatrix} y_2 \end{pmatrix} = \begin{pmatrix} \sin(B) & \cos(B) \end{pmatrix} \begin{pmatrix} y_1 \end{pmatrix}$

Suppose we want to rotate a point about the origin

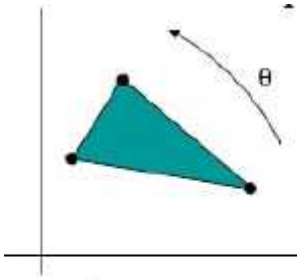


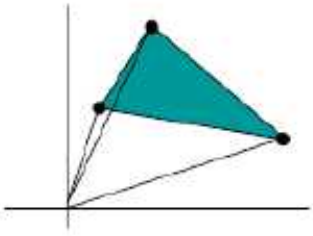
Rotate (x, y) about the origin by θ



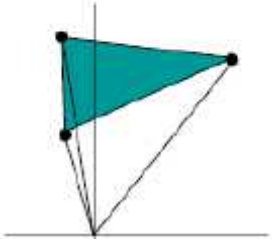
Translating a point along a linear trajectory from P_0 to P_1

To rotate a 2D shape defined by a set of points





Rotate the shape.



B. SCALING (2D)

Scaling is a transformation on an object that changes its size. Just as the translation could have been different amounts in x and y, you can scale x and y by different factors. **Scaling** is a transformation on an object that changes its size within the same coordinate system. To scale an object we scale each of its vertices (points).

To scale a 2D point (x_1, y) by s_x in the x direction and s_y in the y direction, we simply calculate the new coordinates to be: $(x_2, y_2) = (s_x x_1 , s_y y_1)$.

Scaling $(x_2, y_2) = (x_1 * s_x, y_1 * s_y)$

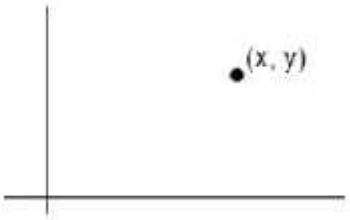
Scaling can be represented by matrix multiplication where the scale factors are along the diagonal.

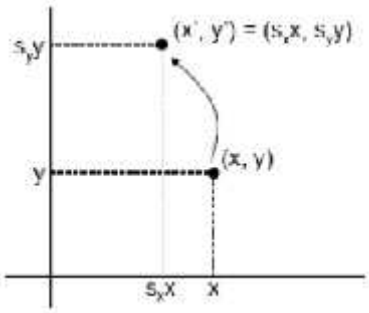
$[s_x \; 0] [x_1] = [s_x * x_1]$

$[0 \; s_y] [y_1] [s_y * y_1]$

Scale by (s_x, s_y) about the origin

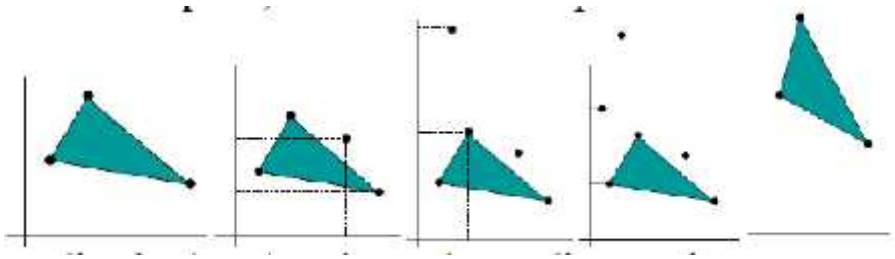
–Scale the coordinates (x,y) by (s_x, s_y)





Otherwise, the scale is non-uniform.

To scale an object about the origin by (s_x, s_y), Scale each point, and redraw the shape.



Scaling by (s_x, s_y) scales each coordinate point

$(x, y) \rightarrow (s_x x, s_y y)$

Scaling is a linear operation. The new coordinates are a linear combination of the previous coordinates. The new coordinates are determined from a linear system

$x' = s_x x$

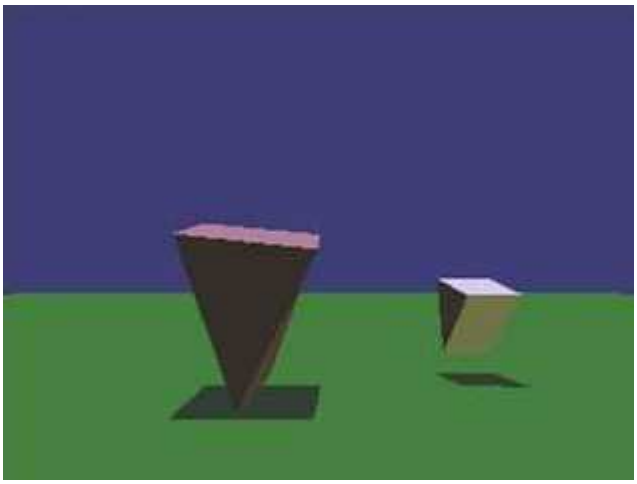
$y' = s_y y$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

7

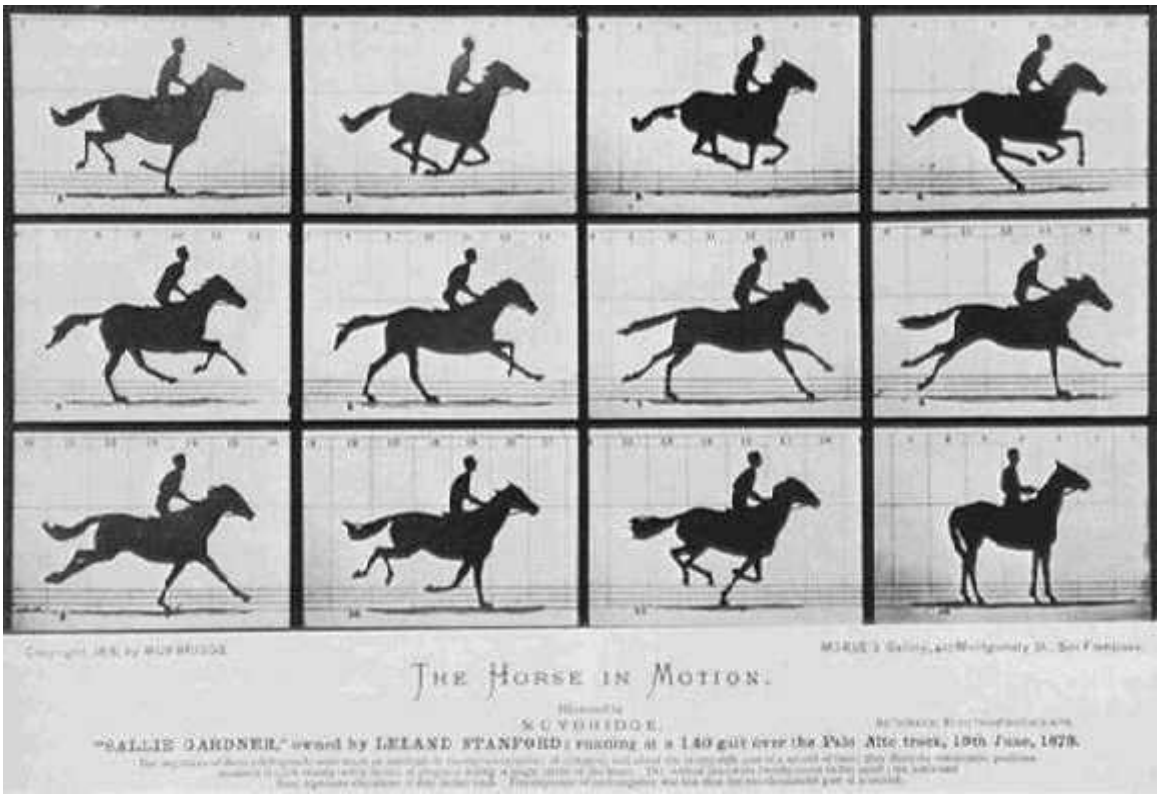
ANIMATION

Motion can bring the simplest of characters to life. Even simple polygonal shapes can convey a number of human qualities when animated: identity, character, gender, mood, intention, emotion, and so on.



Very simple characters (image by Ken Perlin)

A movie is a sequence of frames of still images. For video, the frame rate is typically 24 frames per second. For film, this is 30 frames per second.



In general, animation may be achieved by specifying a model with n parameters that identify degrees of freedom that an animator may be interested in such as

- polygon vertices,
- spline control,
- joint angles,
- muscle contraction,
- camera parameters, or
- color.

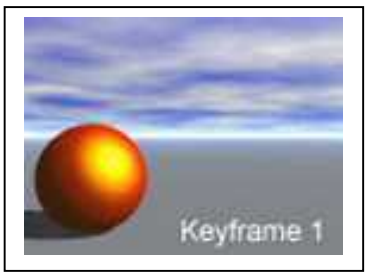
With n parameters, this results in a vector q in n-dimensional state space. Parameters may be varied to generate animation. A model's motion is a trajectory through its state space or a set of motion curves for each parameter over time, i.e. q(t), where t is the time of the current frame. Every animation technique reduces to specifying the state space trajectory.

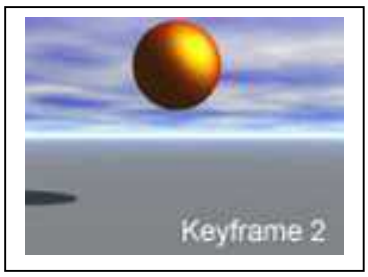
The basic animation algorithm is then: for $t=t_1$ to t_{end} : render(~q(t)).
Modeling and animation are loosely coupled. Modeling describes control values and their actions. Animation describes how to vary the control values. There are a number of animation techniques, including the following:

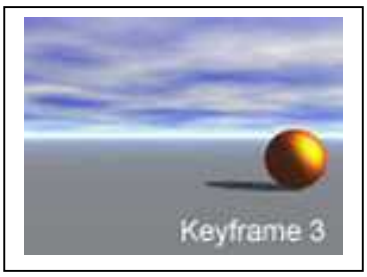
- User driven animation
  - Keyframing
  - Motion capture
- Procedural animation
  - Physical simulation
  - Particle systems
  - Crowd behaviors
- Data-driven animation

A. KEYFRAMING

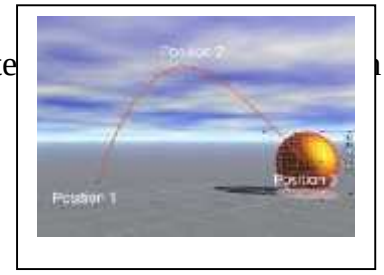
Keyframing is an animation technique where motion curves are interpolated through states at times, (q_1, ..., q_T), called keyframes, specified by a user.

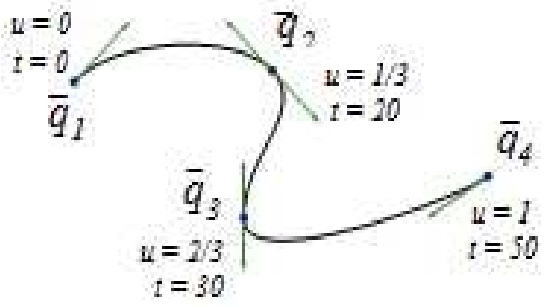






Catmull-Rom splines are well suite because they pass through their control points.





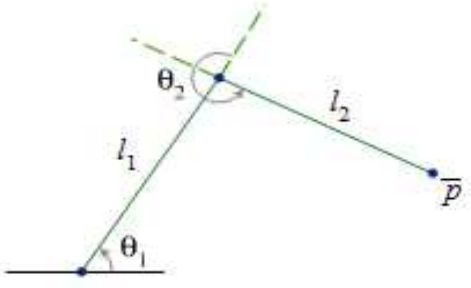
• Pros:
  – Very expressive
  – Animator has complete control over all motion parameters
• Cons:
  – Very labor intensive
  – Difficult to create convincing physical realism
• Uses:
  – Potentially everything except complex physical phenomena such as smoke, water, or fire

B. KINEMATICS

Kinematics describe the properties of shape and motion independent of physical forces that cause motion. Kinematic techniques are used often in keyframing, with an animator either setting joint parameters explicitly with **forward kinematics** or specifying a few key joint orientations and having the rest computed automatically with **inverse kinematics**.

i. Forward Kinematics

With forward kinematics, a point $\bar{p}$ is positioned by $\bar{p} = f(\Theta)$ where Θ is a state vector ($\theta 1, \theta 2, \ldots \theta n$) specifying the position, orientation, and rotation of all joints.



For the above example, $\bar{p} = (l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2), l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2))$.

ii. Inverse Kinematics

With inverse kinematics, a user specifies the position of the end effector, $\bar{p}$, and the algorithm has to evaluate the required Θ give $\bar{p}$. That is, $\Theta = f^{-1}(\bar{p})$.

Usually, numerical methods are used to solve this problem, as it is often nonlinear and either underdetermined or overdetermined. A system is underdetermined when there is not a unique solution, such as when there are more equations than unknowns. A system is overdetermined when it is inconsistent and has no solutions.

Extra constraints are necessary to obtain unique and stable solutions. For example, constraints may be placed on the range of joint motion and the solution may be required to minimize the kinetic energy of the system.

C. MOTION CAPTURE

In motion capture, an actor has a number of small, round markers attached to his or her body that reflect light in frequency ranges that motion capture cameras are specifically designed to pick up.

(image from movement.nyu.edu)

With enough cameras, it is possible to reconstruct the position of the markers accurately in 3D. In practice, this is a laborious process. Markers tend to be hidden from cameras and 3D reconstructions fail, requiring a user to manually fix such drop outs. The resulting motion curves are often noisy, requiring yet more effort to clean up the motion data to more accurately match what an animator wants.

Despite the labor involved, motion capture has become a popular technique in the movie and game industries, as it allows fairly accurate animations to be created from the motion of actors. However, this is limited by the density of markers that can be placed on a single actor. Faces, for example, are still very difficult to convincingly reconstruct.

• Pros:
  – Captures specific style of real actors
• Cons:
  – Often not expressive enough
  – Time consuming and expensive
  – Difficult to edit
• Uses:
  – Character animation
  – Medicine, such as kinesiology and biomechanics

D. PHYSICALLY-BASED ANIMATION

It is possible to simulate the physics of the natural world to generate realistic motions, interactions, and deformations. **Dynamics** rely on the time evolution of a physical system in response to forces.

Newton's second law of motion states $\mathbf{f} = ma$, where $\mathbf{f}$ is force, m is mass, and a is acceleration. If $x(t)$ is the path of an object or point mass, then

$v(t) = \frac{dx(t)}{dt}$ is velocity and $a(t) = \frac{dv(t)}{dt} = \frac{d^2 x(t)}{dt^2}$

is acceleration. Forces and mass combine to determine acceleration, i.e. any change in motion.

In **forward simulation** or **forward dynamics**, we specify the initial values for position and velocity, $x(0)$ and $v(0)$, and the forces. Then we compute $a(t)$, $v(t)$, $x(t)$

where $a(t) = \frac{f(t)}{m}$,

$v(t) = \int_0^t a(t)dt + v(0)$, and $x(t) = \int_0^t v(t)dt + x(0)$.

Forward simulation has the advantage of being reasonably easy to simulate. However, a simulation is often very sensitive to initial conditions, and it is often difficult to predict paths x(t) without running a simulation—in other words, control is hard.

With **inverse dynamics**, constraints on a path x(t) are specified. Then we attempt to solve for the forces required to produce the desired path. This technique can be very difficult computationally.

Physically-based animation has the advantages of:

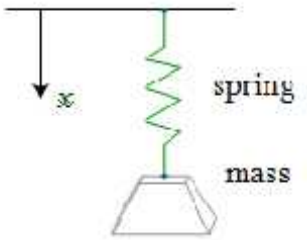
• Realism,
• Long simulations are easy to create,
• Natural secondary effects such as wiggles, bending, and so on—materials behave naturally,
• Interactions between objects are also natural.

The main disadvantage of physically-based animation is the lack of control, which can be critical, for example, when a complicated series of events needs to be modeled or when an artist needs precise control over elements in a scene.

• Pros:
  – Very realistic motion
• Cons:
  – Very slow
  – Very difficult to control
  – Not expressive
• Uses:
  – Complex physical phenomena

i. Single 1D Spring-Mass System

Spring-mass systems are widely used to model basic physical systems. In a 1D spring, x(t) represents the position of mass, increasing downwards.



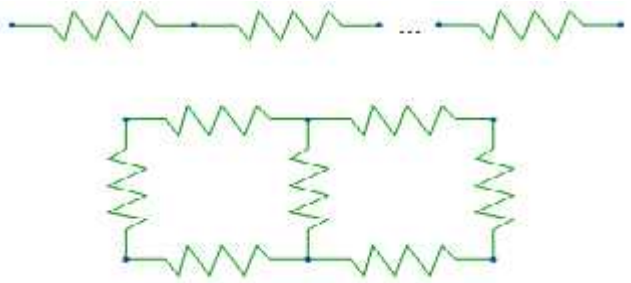
A spring has resting length l and stiffness k. Deformation force is linear in the difference from the resting length. Hence, a spring's internal force, according to Hooke's Law, is $\mathbf{f}^s(t) = k(l - x(t))$.

The external forces acting on a spring include gravity and the friction of the medium. That is, $\mathbf{f}^g = mg$ and $\mathbf{f}^d(t) = -\rho v(t) = -\rho \frac{dx(t)}{dt}$, where ρ is the damping constant.

Hence, the total force acting on a spring is $\mathbf{f}(t) = \mathbf{f}^s(t) + \mathbf{f}^g + \mathbf{f}^d(t)$. Then we may use $a(t) = \frac{f(t)}{m}$ with initial conditions $x(0) = x_0$ and $v(0) = v_0$ to find the position, velocity, and acceleration of a spring at a given time t.

ii. 3D Spring-Mass Systems

Mass-spring systems may be used to model approximations to more complicated physical systems. Rope or string may be modeled by placing a number of springs end-to-end, and cloth or rubber sheets may be modeled by placing masses on a grid and connecting adjacent masses by springs.



Let the ith mass, m_i, be at location $\bar{p}_i(t)$, with elements $x_i(t)$, $y_i(t)$, $z_i(t)$. Let l_{ij} denote the resting length and k_{ij} the stiffness of the spring between masses i and j.

The **internal force** for mass i is

$$f_{ij}^{s}(t) = -k_{ij}\, e_{ij} \frac{p_i - p_j}{\|p_i - p_j\|}.$$

where $e_{ij} = l_{ij} - \|p_i - p_j\|$

k.

> Note:
> It is the case that $f^{s}_{ij}(t) = -f^{s}_{ji}(t)$.

The net total internal force on a mass i is then

$$f^{s}_{i}(t) = \sum_{j \in N_i} f^{s}_{ij}(t),$$

where N_i is the set of indices of neighbors of mass i.

al

ordinary differential equation solver, such as the Runge-Kutta method, with finite difference approximations to derivatives.

To find an approximation to a(t), we choose a time increment Δt so the solution is computed at $t_i = i\Delta t$.

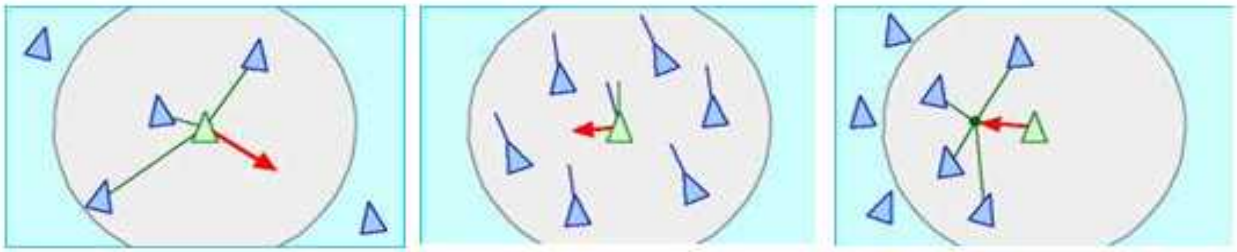
The simplest approach is the use Euler time integration with forward differences:

- Compute $a_i(t) = f_i(t)/m_i$.
- Update $v_i(t + \Delta t) = v_i(t) + \Delta t a_i(t)$
- Update $\bar{p}_i(t + \Delta t) = \bar{p}_i(t) + \Delta t v_i(t)$.

iv. Particle Systems

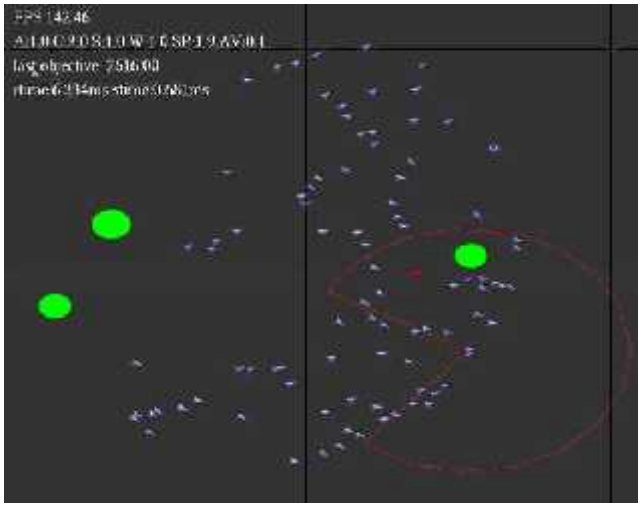
A particle system fakes passive dynamics to quickly render complex systems such as fire, flowing water, and sparks. A particle is a point in space with some associated parameters such as velocity, time to live, color, or whatever else might be appropriate for the given application. During a simulation loop, particles are created by emitters that determine their initial properties, and existing particles are removed if their time to live has been exceeded. The physical rules of the system are then applied to each of the remaining particles, and they are rendered to the display. Particles are usually rendered as flat textures, but they may be rendered procedurally or with a small mesh as well

E. BEHAVIORAL ANIMATION



Flocking behaviors

Particle systems don't have to model physics, since rules may be arbitrarily specified. Individual particles can be assigned rules that depend on their relationship to the world and other particles, effectively giving them behaviors that model group interactions. To create particles that seem to flock together, only three rules are necessary to simulate separation between particles, alignment of particle steering direction, and the cohesion of a group of particles.



Particles that flock and steer around obstacles

More complicated rules of behavior can be designed to control large crowds of detailed characters that would be nearly impossible to manually animate by hand. However, it is difficult to program

characters to handle all but simple tasks automatically. Such techniques are usually limited to animating background characters in large crowds and characters in games.

A crowd with rule-based behaviors

• Pros:
  – Automatic animation
  – Real-time generation
• Cons:
  – Human behavior is difficult to program
• Uses:
  – Crowds, flocks, game characters

F. DATA-DRIVEN ANIMATION

Data-driven animation uses information captured from the real world, such as video or captured motion data, to generate animation. The technique of video textures finds points in a video sequence that are similar enough that a transition may be made without appearing unnatural to a viewer, allowing for arbitrarily long and varied animation from video. A similar approach may be taken to allow for arbitrary paths of motion for a 3D character by automatically finding frames in motion capture data or keyframed sequences that are similar to other frames. An animator can then trace out a path on the ground for a character to follow, and the animation is automatically generated from a database of motion.

• Pros:
  – Captures specific style of real actors
  – Very flexible
  – Can generate new motion in real-time
• Cons:
  – Requires good data, and possibly lots of it
• Uses:
  – Character animation