RV College of Engineering®

*Go, change the world*

# COMPUTER GRAPHICS & VIRTUAL REALITY (18CS72)

## UNIT-I

*Introduction to Computer Graphics and Virtual Reality*

# Introduction to Computer Graphics and Virtual Reality:

***Introduction to Computer Graphics and Virtual Reality:***

*Application areas of Computer Graphics, Introduction to* Graphics Programming with OpenGL, The openGL API: Graphics Functions, The Graphics Pipeline and state Machines, The openGL Interface, Primitives and Attributes, Polygon Basics: polygon types in openGL, Attributes, Color, RGB Color, Indexed Color, Control Functions, The Three- Dimensional Sierpinski Gasket. Display Lists Definition and execution of display Lists, Programming.

***Introduction to Virtual Reality:*** The three I's of virtual reality, commercial VR technology and the five classic components of a VR system.

# Introduction to Computer Graphics and Virtual Reality

*Introduction to Computer Graphics and Virtual Reality: (8 Hrs)*

- ✓ ***Application areas*** *of Computer Graphics,*
- ✓ *Introduction to* Graphics Programming with OpenGL, The openGL API: Graphics Functions, The Graphics Pipeline and state Machines,
- ✓ The openGL Interface, Primitives and Attributes,
- ✓ Polygon Basics: polygon types in openGL, Attributes, Color, RGB Color, Indexed Color, Control Functions,
- ✓ The Three- Dimensional Sierpinski Gasket.
- ✓ Display Lists Definition and execution of display Lists, Programming.

*Introduction to Virtual Reality:*

- ✓ The three I's of virtual reality,
- ✓ commercial VR technology and the five classic components of a VR system.

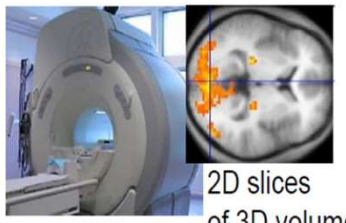
# Introduction

**How to Get Digital Images?**

- 2D real →2D digital
- 3D real →2D digital
- 3Dreal →3D digital


Stereoscopic Camera

3D Scanner



MRI

2D slices of 3D volume

- computer generated

# Introduction

## What to Do with Digital Images?

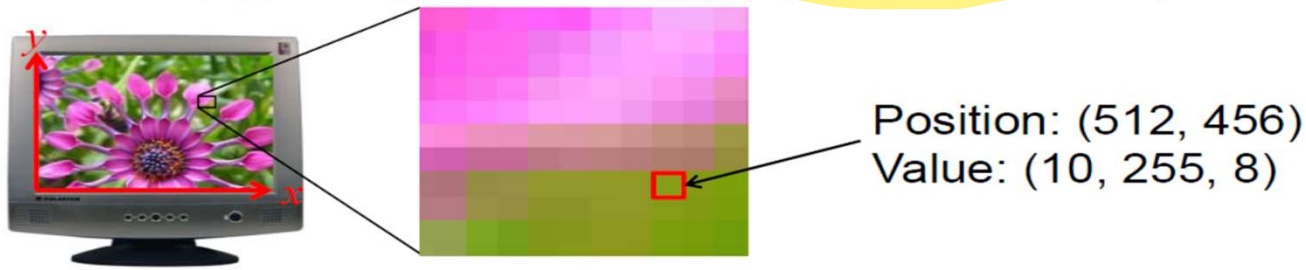
- **Image Processing**: process them to get new images
- **Computer Vision**: analyze them to get information about what is in the image
- **Computer Graphics**: generate them
- Computer Vision often seen as part of Image Processing or Image Analysis

| | | OUTPUT | |
|---|---|---|---|
| | | **Descriptions** | **Images** |
| **INPUT** | **Descriptions** | | Computer Graphics |
| | **Images** | Computer Vision | Image Processing |

# Introduction

## Pixels and Resolution

- **Pixel** or **pel** (picture element)
  Position $(x,y)$ + signal value $v$ (greyscale or colour)




Origin (0,0) of pixel coordinates sometimes in top left corner

- **Resolution:** how many pixels? width × height
- **Spatial resolution**: image pixels per cm or inch (in x and y)
  Can be used to convert pixel coordinates into physical coordinates

what are physical co ordinates

# Introduction

## Encoding of Colors

- **Bit-depth:** number of bits used to represent each pixel's value (typically 1, 8, 24 or 32)
- **Binary image:** bit depth is 1; only code values 0 (black) and 1 (white)
- **Scalar/monochrome/greyscale image:**
  - scalar code values (e.g. just a single number per color)
  - only grey values (from black to white) and no colour
- **Vector-valued image**
  - vector code values (e.g. several numbers per color)
  - All the colors can be represented

12

what happens if but depth is 1, 8, 24, 32
- higher the color depth, the better your images will generally look on the screen.

Binary images is one that consists of pixels that can have one of exactly two colors usually black and white

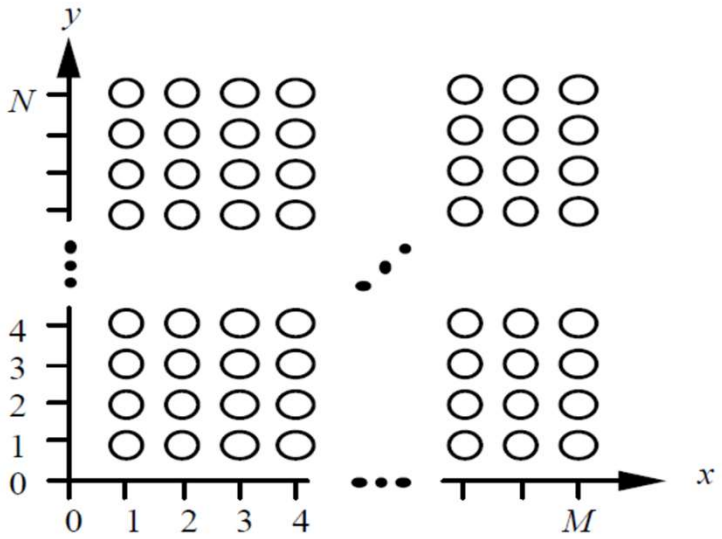
# Introduction

## Defining Images Mathematically

Images can be defined on an $M \times N$ arithmetic grid (or lattice)

$$\mathbf{R}_{M,N} = \{(x, y) : 1 \le x \le M \wedge 1 \le y \le N\}$$

- Pixel coordinates $x$ and $y$ with $x = 1,\ldots,M$ ; $y = 1,\ldots,N$
- Image as a function $f$: $\mathbf{R} \rightarrow \mathbf{V}$
- $\mathbf{V}$ is a set of signal values, e.g. grey levels or colors
- **Example**: pixel at position (100, 100) has color 255, i.e. $f$ (100, 100) = 255



# Introduction

**Computer Graphics**

- is the field of visual computing, where one utilizes computers both to **generate visual images synthetically** and to **integrate** or **alter visual information** sampled from the **real** world.
- **Generation of** (possibly realistic) **images of virtual scenes** using computer hardware.
  - **refers** to the **algorithms, conceptual constructions** and **mathematical background** needed to **render virtual scenes either 2D** or **3D**
- Distinct from "**Image Processing**"- in which images are scanned into the computer.

# Introduction

## Computer Graphics vs Image Processing

- **used to modify (improve picture quality), analyze** and **interpret (recognize visual pattern)** existing **pictures**, such as photographs, and TV scans
- **CG is used to create pictures**
- Typically, a photo/picture is **digitized** into an **image file** before image-processing methods are applied

link - one note

# Introduction

- image-processing methods are **used to**
  - enhance the **quality of the picture**
  - **rearrange** picture parts
  - **enhance color separations**
  - **improve** the quality of **shading** or
  - **retouching**

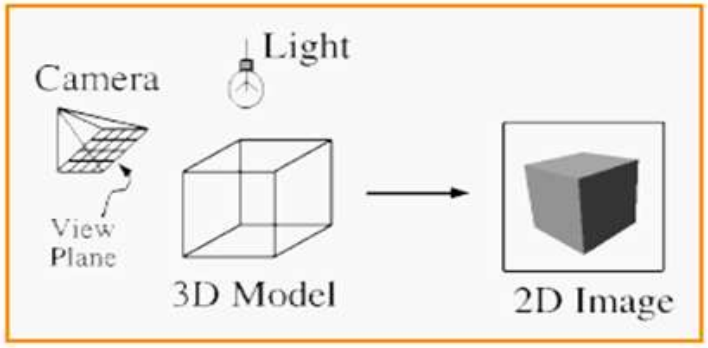
Example:

- **medical applications**:
  - **tomography**
    - is a technique of **X-ray photography** that allows cross-sectional views of physiological systems
  - **ultrasonics**
    - high-frequency **sound waves** to generate digital data
  - **nuclear medicine scanners**
    - collect digital data from **radiation** that is emitted by **ingested radionuclides**, and the **data** is then **plotted** as color-coded images
  - **computer-aided surgery**
    - **simulate actual surgical procedure** and try out different surgical cuts

# Introduction

## Computer Graphics

- What is computer graphics?
  - Imaging = *representing 2D images*
  - Modeling = *representing 3D objects*
  - Rendering = *constructing 2D images from 3D models*
  - Animation = *simulating changes over time*



# Introduction

Why Study Computer Graphics?

- Some people want a better set of tools for plotting curves and presenting the data they encounter in their other studies or works.
- Some want to write computer-animated games, while others are looking for a new medium for artistic expression.
- Most people want to be more productive, and to communicate ideas better, and computer graphics can be a great help.

# Application areas of Computer Graphics

- Display of information
- Design
- Simulation and animation
- User interfaces

# Application areas of Computer Graphics

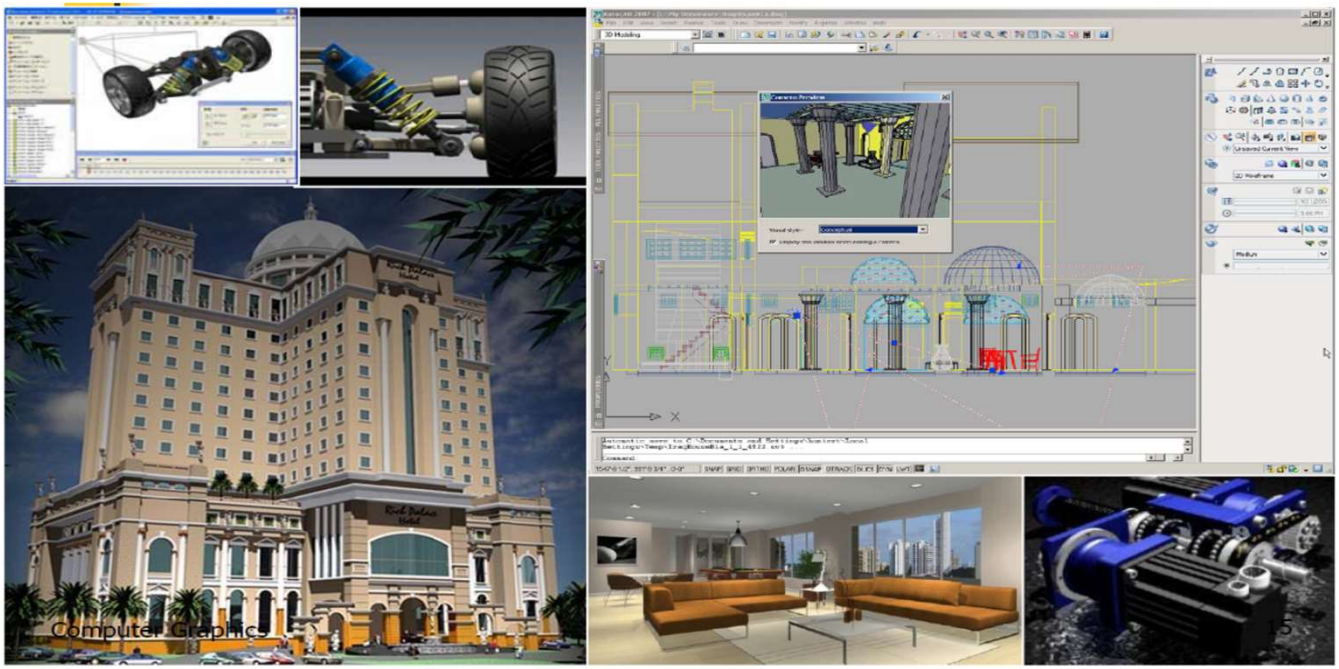
- Design
  - CAD/CAM
- Scientific/Volume Visualization
- Entertainment
  - Movies, Games
- Display Information
  - WWW, Books, Magazines
  - Paint systems
  - Process monitoring
- Simulations
  - VR/AR)

# Application areas of Computer Graphics

Design

- CAD/CAM

Computer aided design - design a building a , a motor, a whatever using software
computer aided manufacturing - use of software and computer-controlled machinery to automate manufacturing process



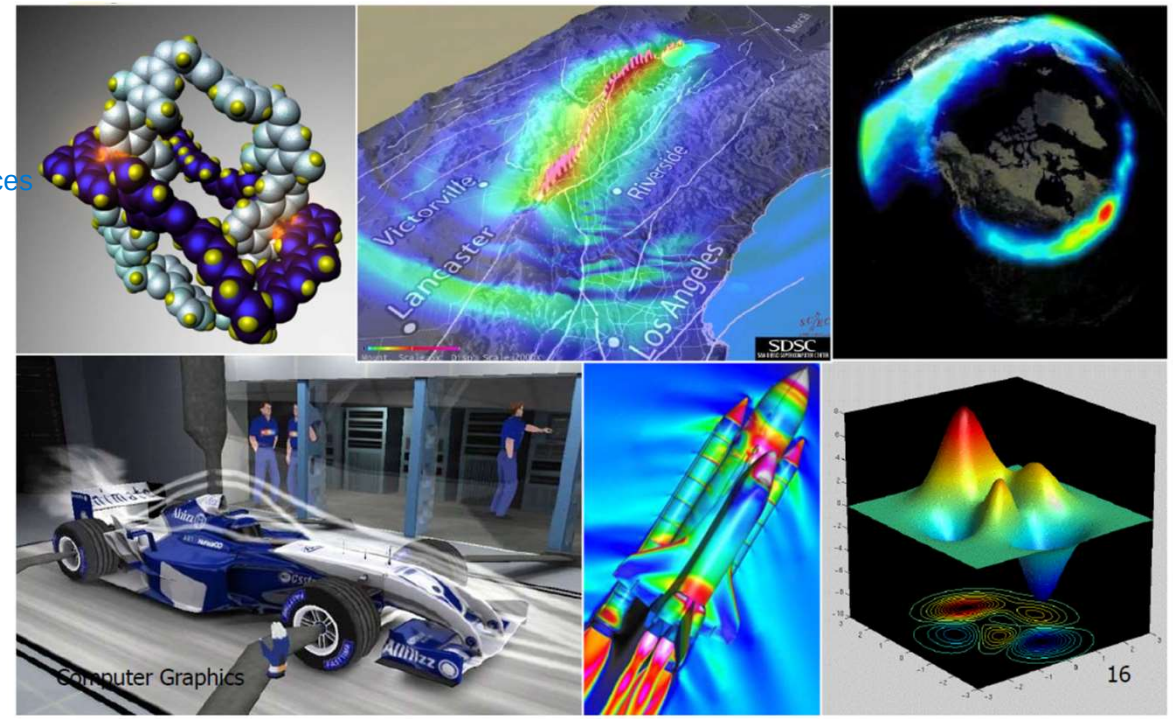
# Application areas of Computer Graphics

- Scientific/Volume Visualization

This comes under which application out of four ? I'll put it under design, simulation - animation allu baratte

Visualization of
atoms
rockets
4dimension, 5D, 6D spaces

DNA
Planets, sun, moon
stars etc



# Application areas of Computer Graphics

- Entertainment
  - Movies

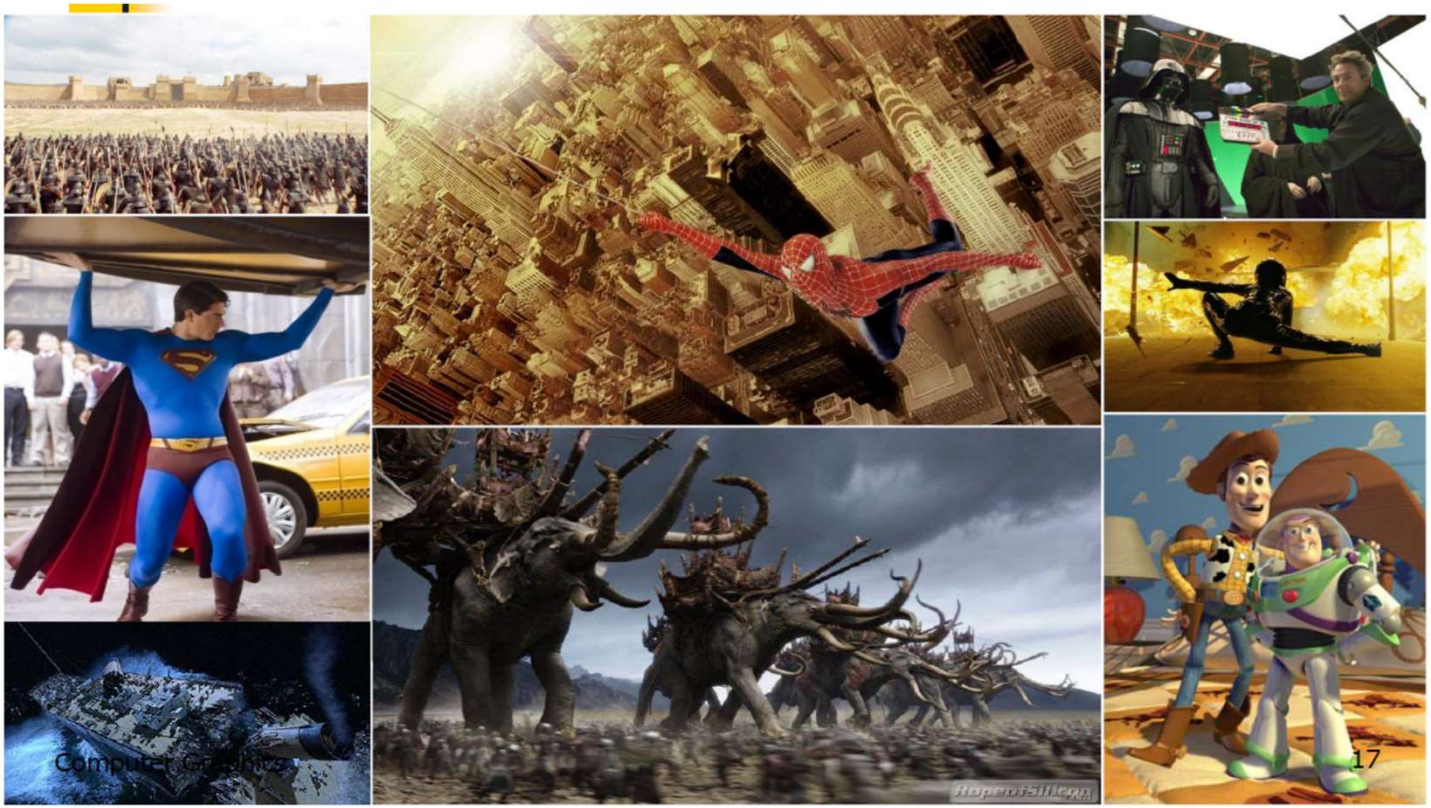
This comes under which one ? out of 4

simulation, animation? hmm

Movies use this at next level
Bahubali
Avatar
Superman,
spiderman
King Kong
Life of pie

such amazing graphics they can generate

graphics has took cinema industry to next level



# Application areas of Computer Graphics

- **Entertainment: Games**

Games - simulation / animation

US reported that two teens were playing video games even though their house wan on firs
- that level of immersion graphics has created

# Application areas of Computer Graphics

- Display Information
  - WWW, Books, Magazines
  - Paint systems
  - Process monitoring
- Browsing on the World Wide Web
  - The browser must rapidly interpret the data on a page and draw it on the screen as high quality text and graphics.
- Slide, Book, and Magazine Design
  - Computer graphics are used in page layout programs to design the final look of each page of a book or magazine.
  - The user can interactively move text and graphics around to find the most pleasing arrangement.

# Application areas of Computer Graphics

**Display Information**

- A paint system generates images. A common example of a paint system and photo manipulation system is Adobe Photoshop®

# Application areas of Computer Graphics

## Process Monitoring

Highly complex systems such as air traffic control systems must be monitored by a human to watch for impending trouble.

- An air traffic control system consists of monitors that display where nearby planes are situated.
  - The user sees a schematic representation for the process, giving the whole picture at a glance.
  - Various icons can flash or change color to alert the user to changes that need attention.

# Application areas of Computer Graphics

Process Monitoring

Planes

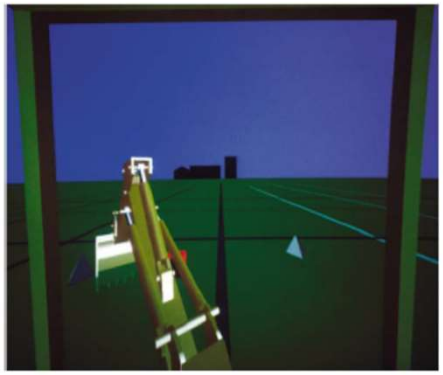

Computer Graphics

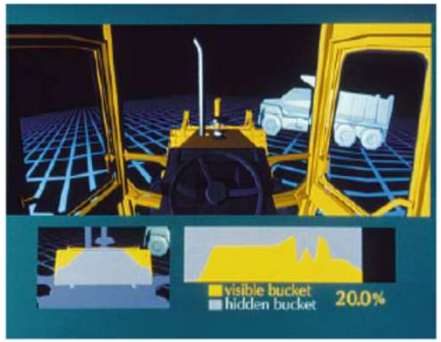
22

# Application areas of Computer Graphics

**Displaying Simulations**

- Machine simulator
- Flight simulator
  - The system is a plane with a shape and flying characteristics, along with a world consisting of a landing field, mountains, other planes, and air, all modeled appropriately.






# Application areas of Computer Graphics

**Simulation:**

**Virtual/Augmented Reality**

# Application areas of Computer Graphics

## Graphs and Charts

**for**

- printed reports
- slides
- transparencies
- animated video

**used** to summarize

- financial
- statistical
- mathematical
- scientific
- engineering
- economic data for
  - research paper
  - managerial summaries
  - consumer info bulletins etc.

Interactive dashboards instead of excel sheet data

# Application areas of Computer Graphics

Computer-Aided Design CAD

- **for** engineering and architectural systems
- **used in** the design of
  - buildings
  - automobiles
  - aircraft
  - watercraft
  - spacecraft
  - computers
  - textiles
  - home appliances etc.

Nearly everything

# Application areas of Computer Graphics

Redundant slide

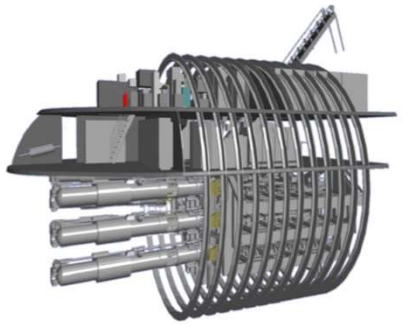
**Computer-Aided Design CAD or**

**Computer-Aided Drafting and Design CADD**

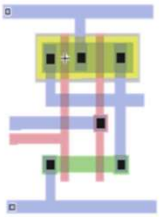
- **for** engineering and architectural systems
- **used in** the design of
  - buildings
  - automobiles
  - aircraft
  - water**craft**
  - space**craft**
  - computers
  - textiles
  - home appliances etc.

# Application areas of Computer Graphics

## Computer-Aided Design CAD or Computer-Aided Drafting and Design CADD



- the shapes used in a design **represent** the different network or circuit **components**
- standard shapes and links between them for mechanical, electrical, electronic, and logic circuit are **supplied** by the **design package**
  - designer can create personalized symbols



# Application areas of Computer Graphics

## Computer-Aided Design CAD

- **Real time, computer animation** is **used to test** a vehicle **performance**
  - **wire-frame displays** allow the designer to **see into** the **interior** of the vehicles and to watch the **behavior** of inner **components** during motion
- When object designs are **complete**,
  - **realistic lighting conditions** and **surface rendering** are applied to produce **displays** of the **final** product or **ad** of the product

# Application areas of Computer Graphics

## Computer-Aided Design CAD

Home design

- If the **fabrication** of a product can be **automated** is referred to as **CAM** Computer-Aided Manufacturing .
- **Architects** use **interactive computer graphics** (CG) methods in CAD **packages** to **lay out** rooms, doors, windows, stairs etc. in a building design in a way to **optimize space utilization** or **arrange** electrical **elements** on the circuit board
  - **CAD packages** provide **facilities for experimenting** with 3D interior layouts and lighting

# Application areas of Computer Graphics

Data Visualizations

display of info

Dashboards

- are produced as **graphical representations** for
  - scientific
  - engineering
  - medical **data sets,** referred to as **scientific visualization**

  **or**
  - commerce
  - industry
  - Nonscientific **data sets** referred to as **business visualization**

# Application areas of Computer Graphics

display of info

Data Visualizations

Medical - all the scannings

**used** to:

- **scan** the large **data sets to** determine trends and relationships amount data values
  - **data sets** (a collection of data) can contain scalar values, vectors, higher-order tensors, or any combination of these

  by converting to a color-coded display **(modeling data** or **color coded visualization)**
- **aid** in the understanding and analysis of complex **processes** and mathematical **functions** .
- **simulate** of **effects**

# Application areas of Computer Graphics

simulation

## Education and Training

**used**

- **models** of physical **processes**, physiological **functions**, population **trends**, or equipment to **help trainees** to **understand** the operation of a **system.**

- Examples of such specialized system are **simulators for**
  - training of ship captain
  - aircraft pilots
  - heavy equipment operator
  - air traffic personnel

yeah!, can't give plane to a person who is yet to learn how to fly it

- Most simulators provide **screens** **for visual displays** of the external environment

Driving Simulation
(Evans & Sutherland)

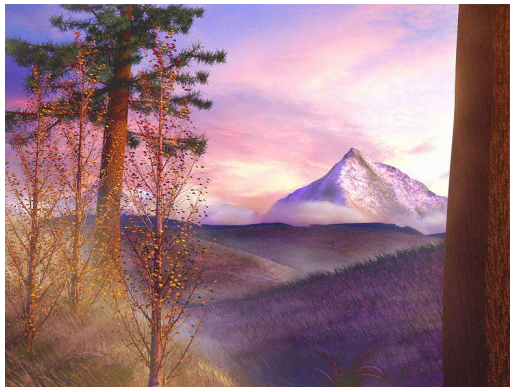
Flight Simulation
(NASA)

# Application areas of Computer Graphics

## Computer Art

**used**

- computer-graphics **methods** and **tools** for **fine** and **commercial arts**
- **Tools** include
  - **commercial** software packages
  - symbolic **mathematics** programs
  - CAD packages
  - desktop publishing software
  - animation software
  - paintbrush program etc.
  - Computer-generated **animations** are **used** in producing TV commercials.



# Application areas of Computer Graphics

Animation

## Entertainment

**used**

- computer-graphics **methods** such as computer modeling, computer-rendering and animation **for**
  - **TV productions**
  - **motion pictures**
  - **music videos**

**to get special effects, animations, characters, and scenes.**


Jurasic Park
*(Industrial, Light, & Magic)*

Quake
*(Id Software)*

**For example**

jackie chan, mr bean, tom and jerry, motu and patu, chota bheem

- **combine computer-generated figures** of people, animals, or cartoon characters with the **live actors**
- or **transform** an actor's **face into another shape** or object or **generate** buildings, terrain features, or other background scenes

Geri's Game
*(Pixar Animation Studios)*

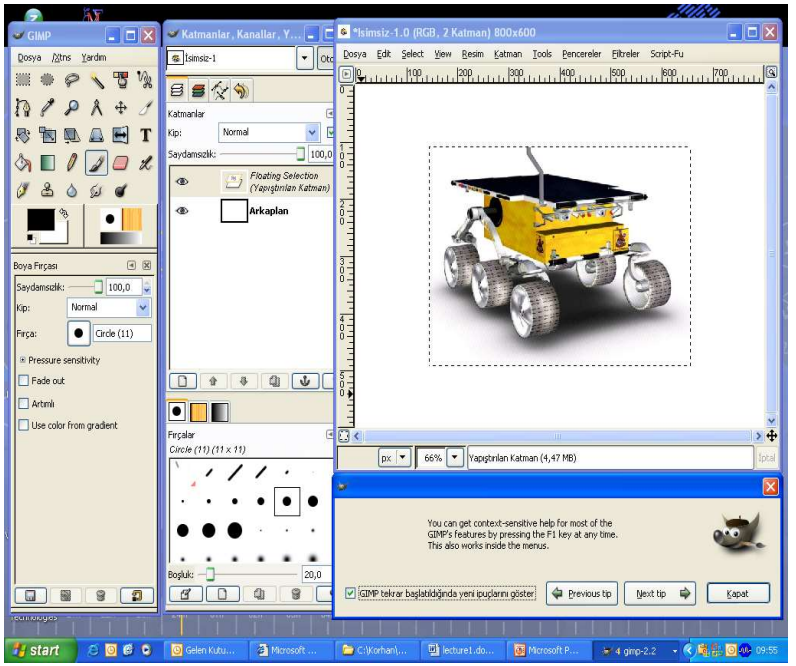
# Application areas of Computer Graphics

## Graphical User Interface GUI

- **used** for applications and operating system software e.g.
  - Internet Browsers
  - Innovative Interfaces
  - Desktop Interfaces

GUI over CLI - git for example

- a major component is a window manager that allows a user to display multiple rectangular screen areas, called **windows**
- GUI includes:
  - **menus**
    - selection of options, color values, and graphics parameters
  - **tool bars**
  - **icons**
  - **popup menus**





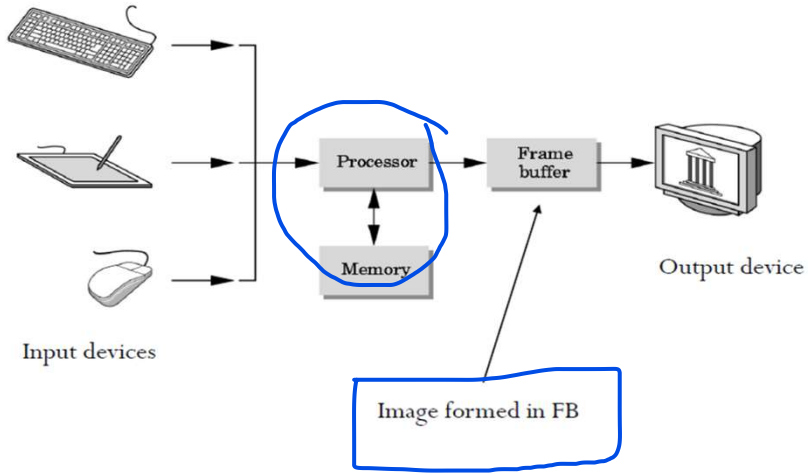

Completes application of computer graphics

from here till - enilla shut up and study

# Image Formation

- In computer graphics, we form images which are generally two dimensional using a process analogous to how images are formed by physical imaging systems
  - Cameras
  - Microscopes
  - Telescopes
  - Human visual system

# Basic Graphics System



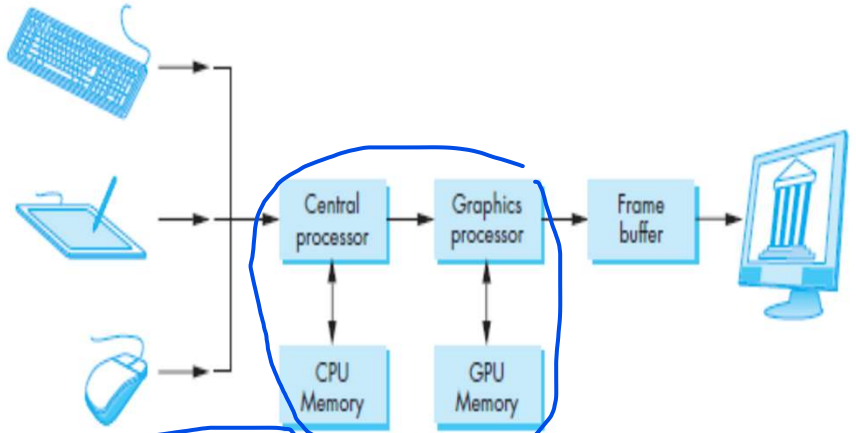
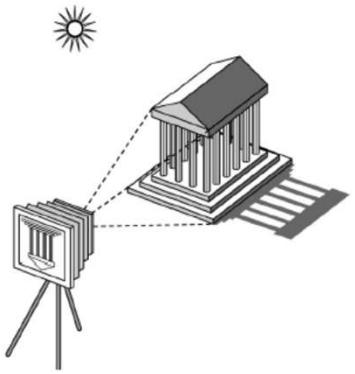




FIGURE 1.1 A graphics system.

# Basic Graphics System

## Elements of Image Formation

- Object ✓
- Viewer ✓
- Light source(s) ✓

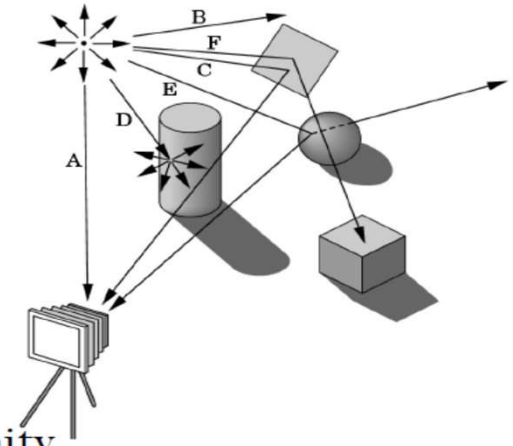


- Attributes that govern how light interacts with the materials in the scene
- Note the independence of the objects, the viewer, and the light source(s)

# Basic Graphics System

## Ray Tracing and Geometric Optics

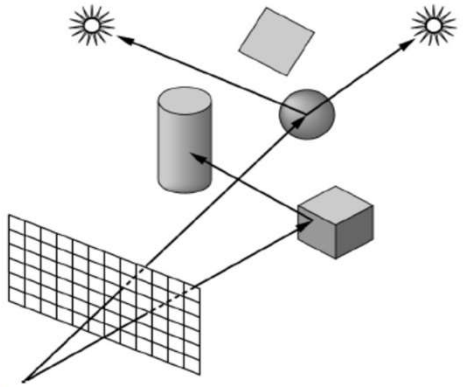
One way to form an image is to follow rays of light from a point source finding which rays enter the lens of the camera. However, each ray of light may have multiple interactions with objects before being absorbed or going to infinity.



# Basic Graphics System

## Physical Approaches

- **Ray tracing**: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity
  - Can handle global effects
    - Multiple reflections
    - Translucent objects
  - Slow
  - Must have whole data base available at all times



- **Radiosity**: Energy based approach
  - Very slow

# Basic Graphics System

Rasterization

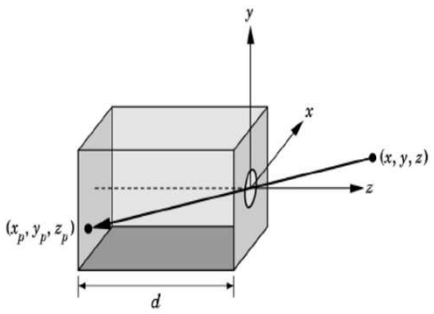
- The conversion of geometric entities to pixel colors and locations in the frame buffer is known as rasterization or scan conversion process.
- General purpose CPU / GPU can do this work.

# Basic Graphics System

- Field /angle of view: the angle made by largest object that
the camera can image on its film plane.
- Depth of field: focusing points within its field of view.
- Drawbacks of pinhole cameras:
  - Single ray of light is admitted.
  - Cannot adjust angle of view.
  - Solution is Lens-based cameras.

i'll leave this for now

## Pinhole Camera



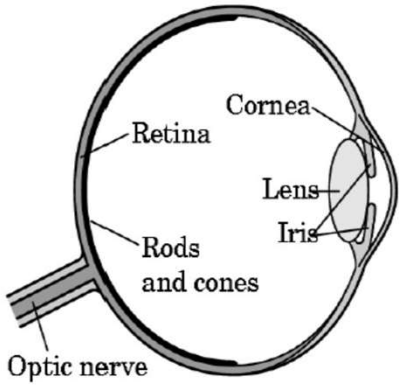

**Use trigonometry to find projection of point at (x,y,z)**

$x_p = -x/z/d \qquad y_p = -y/z/d \qquad z_p = d$

# Basic Graphics System

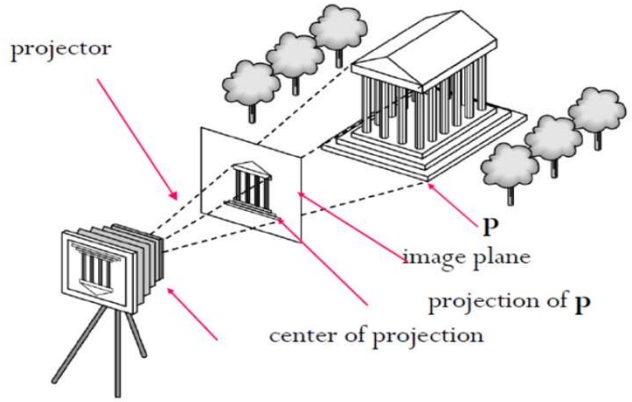
## Three-Color Theory

- Human visual system has two types of sensors
  - Rods: monochromatic, night vision
  - Cones
    - Color sensitive
    - Three types of cones
    - Only three values (the *tristimulus* values) are sent to the brain
- Need only match these three values
  - Need only three *primary* colors



# Basic Graphics System

## Synthetic Camera Model



# Basic Graphics System

A computer graphics system is a computer system; as such, it must have all the components of a general-purpose computer system.

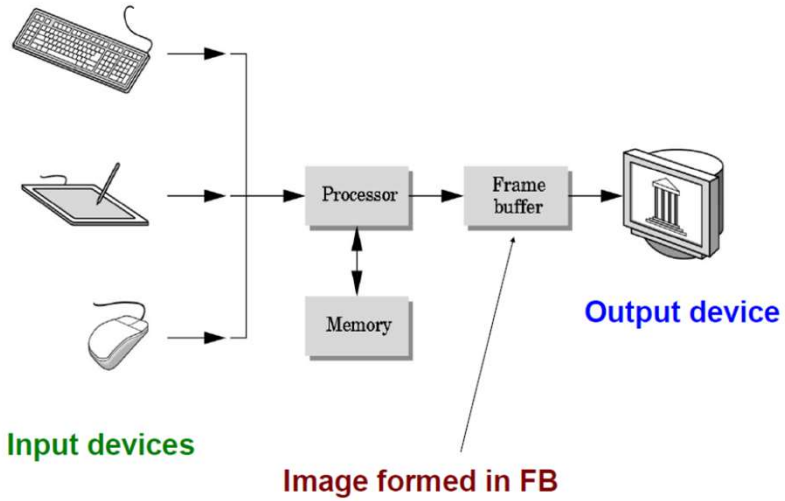
- A ***graphics system*** is specialized for performing graphic based operations and image generations.
- Has all components of a general-purpose computer.

A ***Graphics system has 5 main elements:***

- Input Devices
- Processor
- Memory
- Frame Buffer
- Output Devices

# Basic Graphics System

## Redundant slide



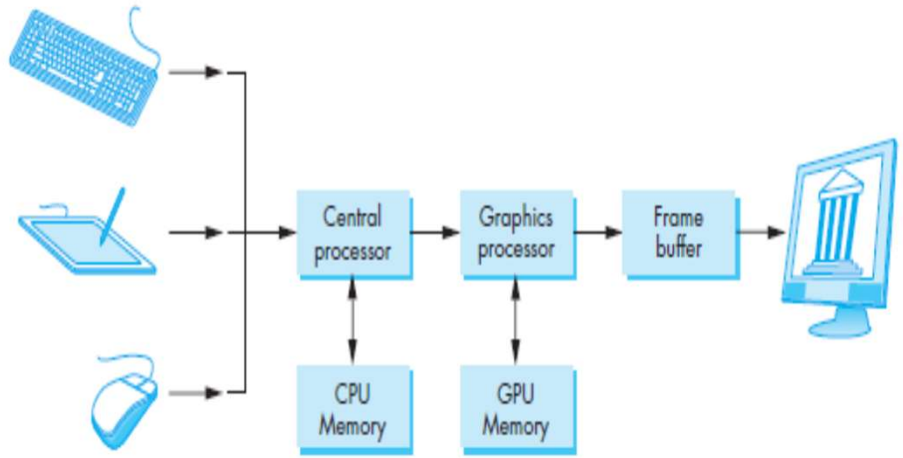
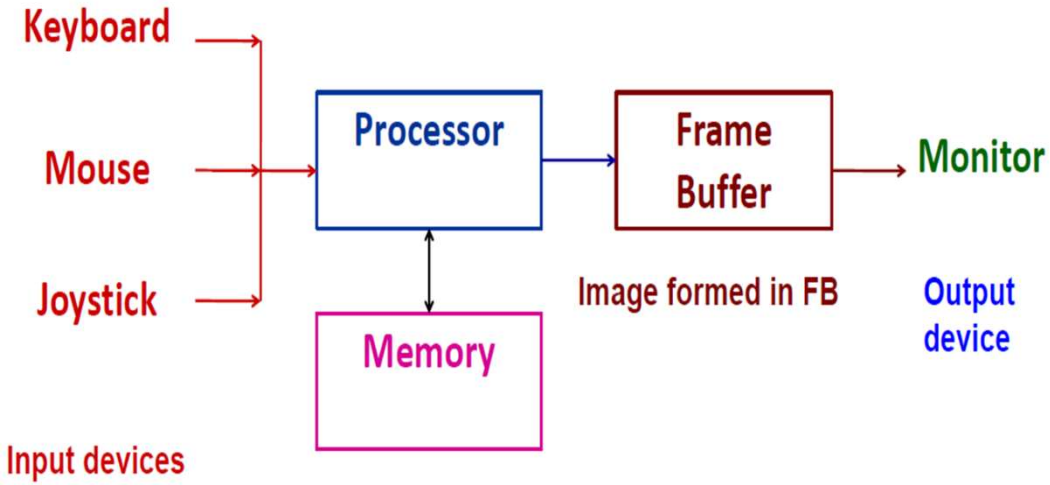




FIGURE 1.1 A graphics system.

# Basic Graphics System

redundant slide



# Basic Graphics System

## Input Devices

Go, ch

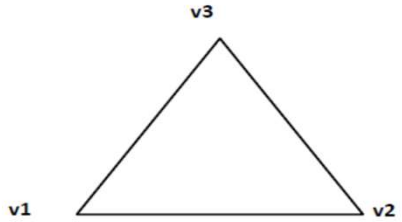
- Keyboard + another input device
  - like **mouse, joystick or data tablet** - called as pointing devices.
  - Allow user to indicate a particular location on the display.
- Advanced Graphics systems allow 3D input devices
  - like **Laser range finders** *(uses a laser beam to determine the distance to an object)* and **acoustic sensors** *(to measure sound levels).*


# Basic Graphics System

## Processor

- **Graphical function of the processor**
  - to take specifications of graphical primitives (lines, polygons etc) generated by application programs.
  - to assign values to the pixels in the frame buffer.
- **Example :**

A triangle is specified by its 3 vertices, but to display its outline by 3 line segments connecting vertices, graphics system must generate a set of pixels that appear as line segments to the viewer.



# Basic Graphics System

## Processor

- All graphic systems have a **specialized graphics processor.**
- It is designed to carryout special graphics functions such as **rasterization.**

# Basic Graphics System

previous definition - Process of converting geometric entities to pixel colors and locations in FB

## Rasterization (Scan Conversion)

- The process of converting primitives into pixel representation(2D image).
- Each point of this image contains information such as color and depth.

- Now a days, special purpose graphics processing units(GPUs) are used.
- GPU can be on mother board or on a graphics card.

# Basic Graphics System

## Memory

Graphics system has two types of memory.

- General Memory
- Frame Buffer

**General Memory** – used to store ***non-graphics*** based data.

**Frame Buffer** – used to store ***graphics*** data.

# Basic Graphics System

## Frame Buffer

- It is a special type of memory present on graphics system.
- In CG, a picture or an image is produced as an ***array of pixels(Raster).*** These pixels are stored in the frame buffer.
- ***Resolution of frame buffer***

  It is the no. of pixels in the frame buffer.

→ see next slide

# Basic Graphics System

➢ **Pixels and the Frame Buffer**

- ❑ A picture is produced as an array (raster) of picture elements (pixels).
- ❑ These pixels are collectively stored in the Frame Buffer.

➢ **Properties of frame buffer:**

- ❑ ***Resolution*** – number of pixels in the frame buffer
- ❑ ***Depth or Precision*** – number of bits used for each pixel

E.g.: 1 bit deep frame buffer allows 2 colors

8 bit deep frame buffer allows 256 colors.

➢ In **full-color** systems, there are 24 (or more) bits per pixel. Such systems can display sufficient colors to represent most images realistically.

- ❑ They are also called **true-color** systems, or **RGB-color** systems because individual groups of bits in each pixel are assigned to each of the three primary colors—red, green, and blue—used in most displays.

# Basic Graphics System

- A **Frame buffer** is implemented either with special types of memory chips or it can be a part of system memory.
- In simple systems **the CPU** does both normal and graphical processing.
- Graphics processing - Take specifications of graphical primitives from application program and assign values to the pixels in the frame buffer
  - is also known as ***Rasterization or scan conversion***.

Another definition aa ?

- Today, virtually all graphics systems are characterized by special-purpose **graphics processing units (GPUs),** custom-tailored to carry out specific graphics functions.
  - The GPU can be either on the mother board of the system or on a graphics card.

GPU position ,FB

- The ***frame buffer*** is ***accessed through*** the graphics processing unit and usually is on the same circuit board as the GPU.

# Basic Graphics System

**Pixels**

skip

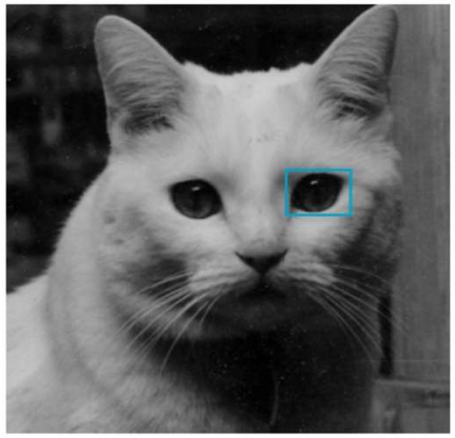
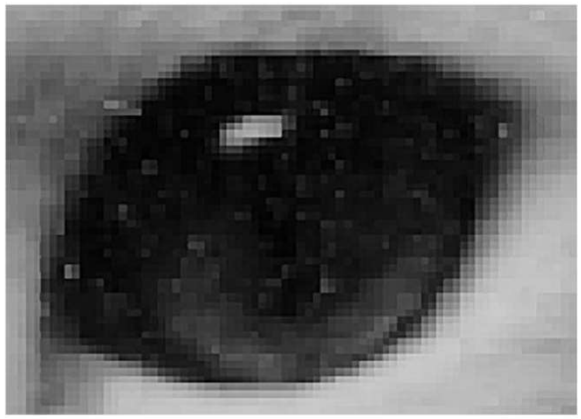


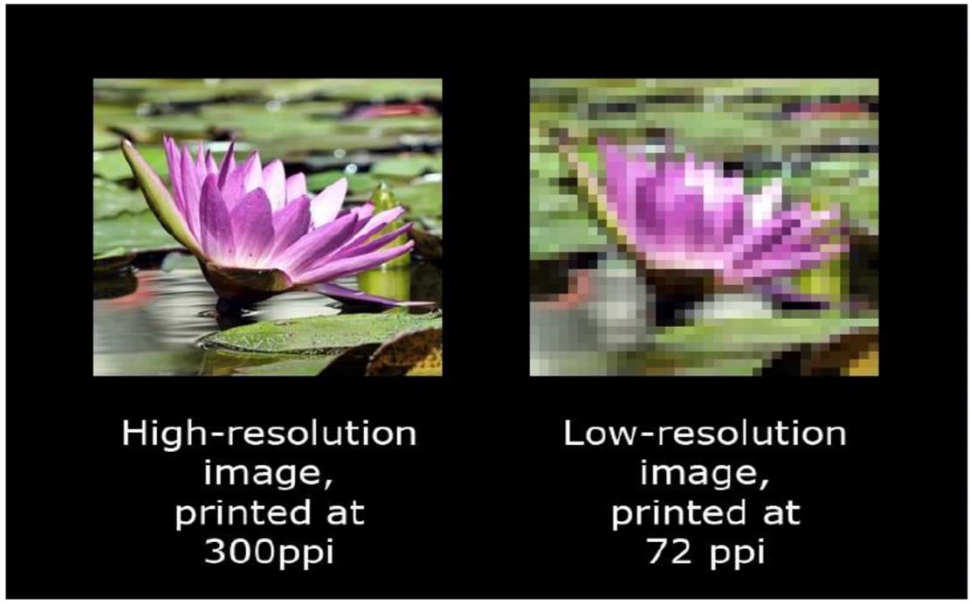
Image of a Cat using pixels



Detail of area around one eye showing individual pixels

# Basic Graphics System

**Pixels**

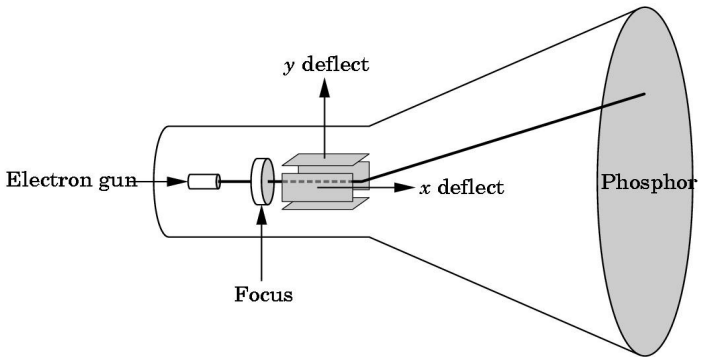



ppi - pixels per inch
- is the measure of resolution in a digital image or video display

# Basic Graphics System

**Output Devices**

- The most predominant type of display has been the Cathode Ray Tube (CRT).
- They are rapidly being replaced by flatscreen technologies such as LEDs, LCDs ,plasma etc.



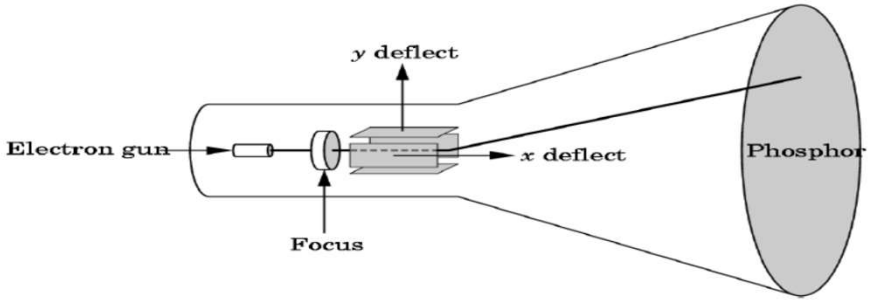
# Basic Graphics System

do we need this ?

skipping now

## Working of Cathode Ray tube (CRT)

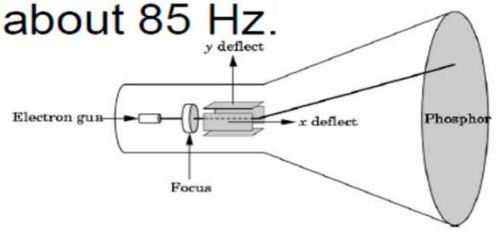
- The digital output produced by a processor is given to ***digital-to-analog converter.***
- The analog voltages produced are applied across the x and y deflection plates.
- The direction of the beam can be controlled by these two deflection plates.
- When electrons strike the phosphor coating on the tube, light is emitted.



# Basic Graphics System

## Working of Cathode Ray tube (CRT)

- Colored CRTs have 3 different colored phosphors (**RED**, **GREEN**, **BLUE**). They have 3 electron beams corresponding to 3 types of phosphors.
- A typical CRT will emit light only for a short duration when hit by an electron.
- This would produce flicker in the image.
- To produce a flicker-free image, the electron beam must be refreshed at a high rate called ***refresh rate.***
- Modern displays have refresh rate of about 85 Hz.



# Basic Graphics System

**CRT :**

**Refresh Rate** – In order to view a flicker free image, the image on the screen has to be retraced by the beam at a high rate (modern systems operate at 85Hz)

- **2 types of refresh:**

- **Noninterlaced display:** Pixels are displayed row by row at the refresh rate.
- **Interlaced display:** Odd rows and even rows are refreshed alternately.

# Basic Graphics System

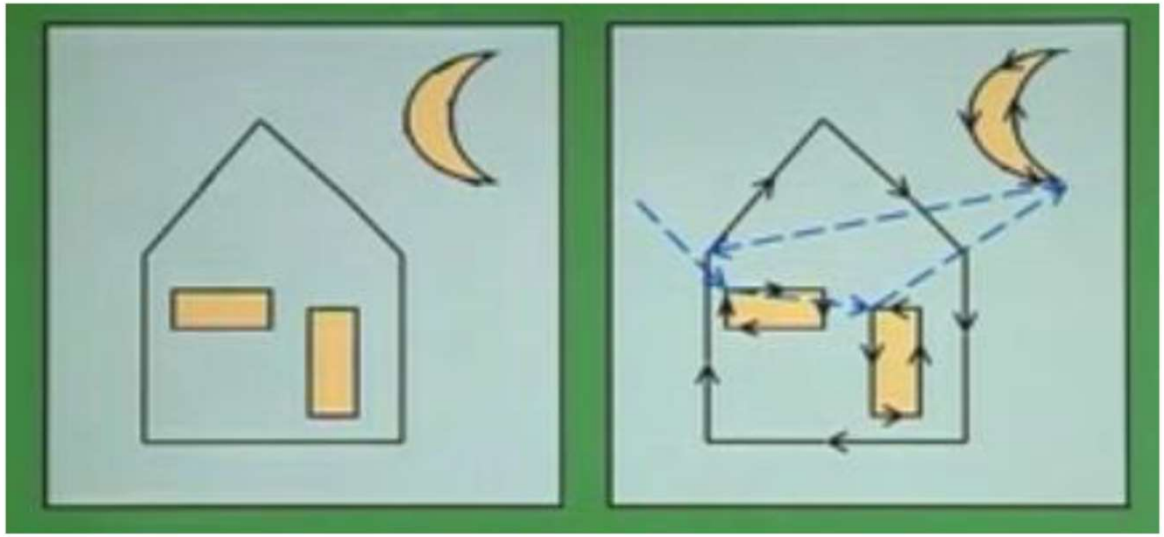
| Random Scan CRT | Raster system CRT |
|---|---|
| Also called as calligraphic OR vector CRT. **Characters are made of sequence of strokes.** | **Entire screen is a matrix of pixels.** It takes pixels from frame buffer and displays them as points on the surface of the CRT. (Noninterlaced & interlaced displays) |
| A beam can be moved from any position to any other position. | Line cannot be drawn directly from one point to another. |
| Limited use. | Widely used. |
| Less efficient. | More efficient and can generate any image. |

see some video

# Basic Graphics System

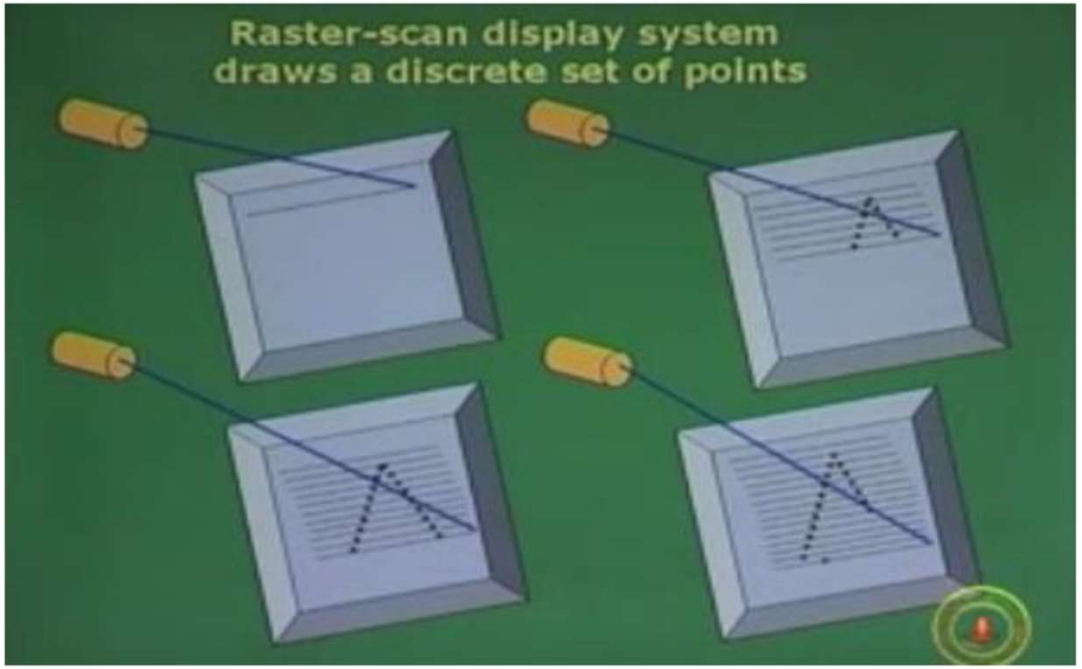
**Random scan CRT**

above differences bardmele, write these examples



# Basic Graphics System

## Raster System CRT

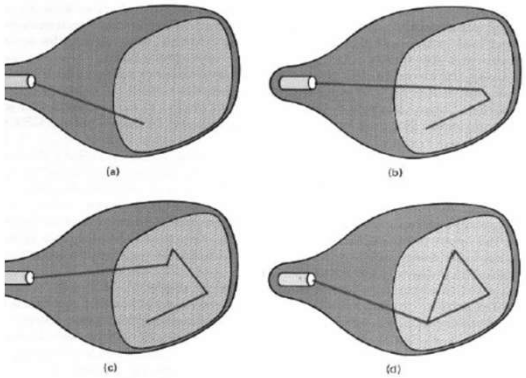


# Basic Graphics System

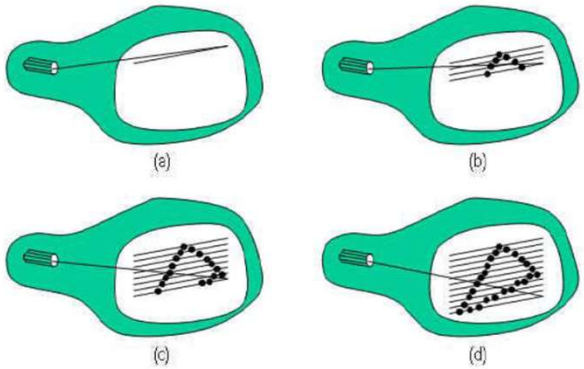
## Types of CRTs

nicely explains

**Random scan CRT**



**Raster system CRT**



# Basic Graphics System

Redundant slide

**CRT :**

**Refresh Rate** – In order to view a flicker free image, the image on the screen has to be retraced by the beam at a high rate (modern systems operate at 85Hz)

- **2 types of refresh:**

- **Noninterlaced display:** Pixels are displayed row by row at the refresh rate.
- **Interlaced display:** Odd rows and even rows are refreshed alternately.

# Basic Graphics System

- In a raster system- the graphics system takes pixels from the frame buffer and displays them as points on the surface of ***the display in one of two fundamental ways***.
- In a **noninterlaced** system, the pixels are displayed row by row, or scan line by scan line, at the refresh rate.
- In an **interlaced** display, odd rows and even rows are refreshed alternately.
  - Interlaced displays are used in commercial television.
  - In an interlaced display operating at 60 Hz, the screen is redrawn in its entirety only 30 times per second
  - Viewers located near the screen, however, can notice the difference between the interlaced and noninterlaced displays.
- **Noninterlaced displays** are becoming more widespread, even though these ***displays process pixels at twice the rate of the interlaced display.***

# Basic Graphics System

- Color CRTs have three different colored phosphors (red, green, and blue), arranged in small groups. One common style arranges the phosphors in triangular groups called **triads**, each triad consisting of three phosphors, one of each primary.
- Most color CRTs have three electron beams, corresponding to the three types of phosphors.
- In the ***shadow-mask CRT*** a metal screen with small holes—the **shadow mask**—ensures that an electron beam excites only phosphors of the proper color.

how

write this diagram as well

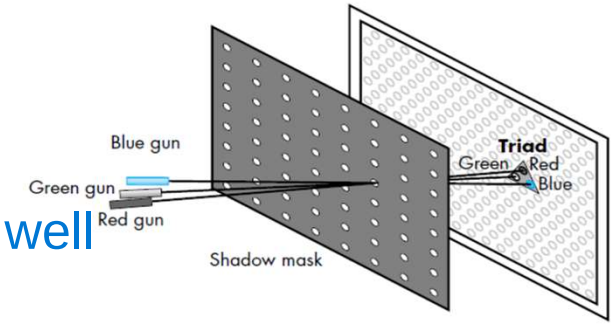
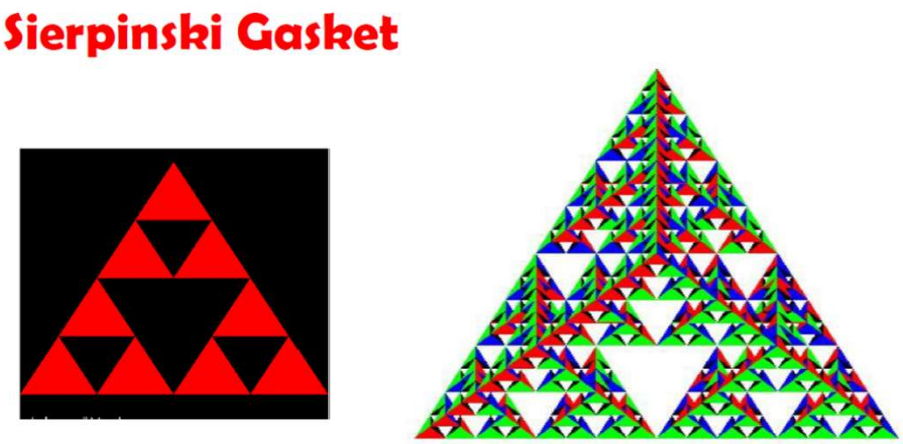



FIGURE 1.4 Shadow-mask CRT.

# Sierpinski Gasket



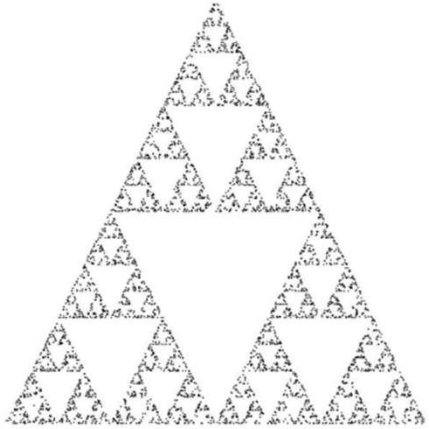

skip

# Sierpinski Gasket

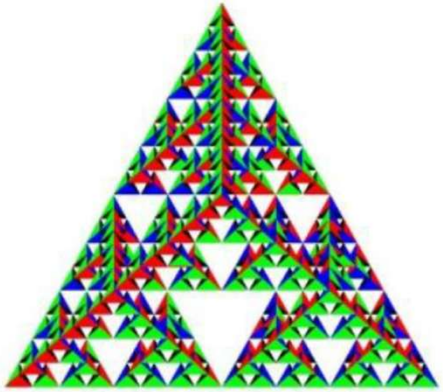
## Sierpinski Gasket Construction

skip

random number generation



recursion



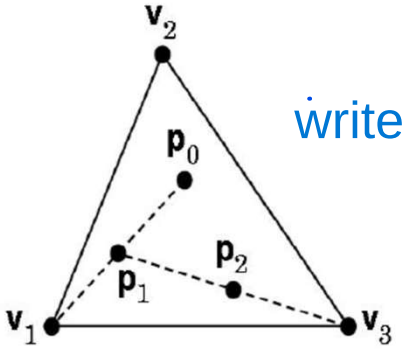
# Sierpinski Gasket

## Construction of Sierpinski Gasket - using random numbers

how can this algo generate s gasket

The construction of sierpinski gasket as follows:

1. Pick up a random point inside the triangle.
2. Select one of the three vertices at random.
3. Find the point halfway between initial point and the randomly selected vertex.
4. **Display this new point** by putting some sort of marker, such as a small circle, at its location.
5. Replace the initial point with this new point.
6. Return to step 2.




write this

**Thus, each time a point that is generated, it is displayed on the output device.**

**In the figure $p_0$ is the initial point, and $p_1$ and $p_2$ are the first two points generated by the algorithm.**

# Sierpinski Gasket

## Construction of Sierpinski Gasket - using random numbers

**Possible form of our program might be:**

```
main ( )
{
      initialize_the_system ();
      for (some_number_of_points )
      {
          pt = generate_a_point();
          display_the_point(pt);
      }
      cleanup()
}
```

# Sierpinski Gasket

## Construction of Sierpinski Gasket using OpenGL

```
#include <GL/glut.h>
void main (int argc, char **argv) /* the main function */
{
        glutInit(&argc,argv);
        glutDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize (500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("simple OpenGL example");
        glutDisplayFunc(display);
        myinit( );
        glutMainLoop( ); // Event Processing
}
```

all these functions are the docs - GLUT docs, one note link

# Sierpinski Gasket

## Construction of Sierpinski Gasket using OpenGL

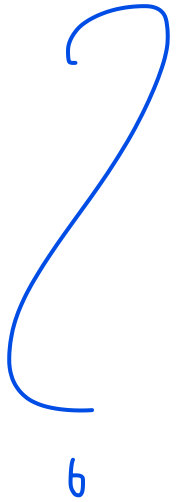
omg, I really need to brush up C

```
typedef GLfloat point2 [2];
/* defines a point as two dimensional array */

void display(void)
{       /* an arbitrary triangle */
        point2 vertices[3]= { {0.0,0.0}, {250.500}, {500.0,0.0}};
        static point2 p = {75.0, 50.0}; /* set to any initial point */
        int j,k;
        int rand( ); /* standard random number generator */
```

# Sierpinski Gasket

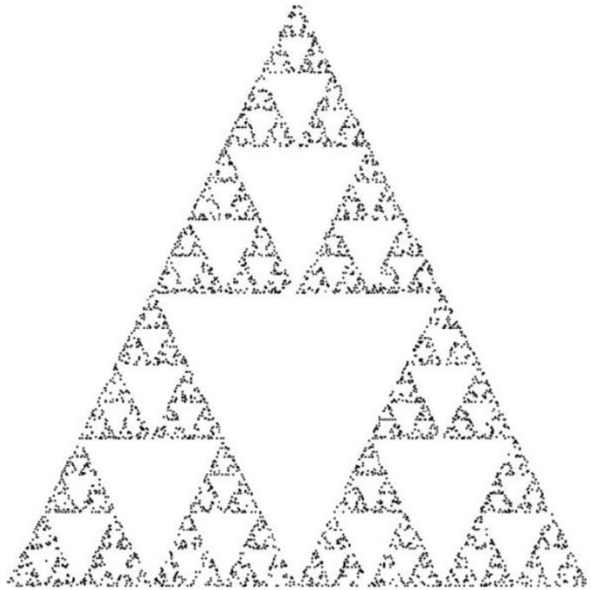
## Construction of Sierpinski Gasket using OpenGL

```
for (k=0;k<5000;k++)
{       j = rand ( )%3; /* pick a random vertex from 0, 1, 2 */
        p[0] = (p[0]+ vertices[ j ][0])/2;
        p[1] = (p[1]+ vertices[ j ][1])/2;
        /* display the new point */
        glBegin(GL_POINTS);
        glVertex2fv(p);
        glEnd();
}
glFlush(); /* points are to be displayed as soon as possible */
}
```



# Sierpinski Gasket

**Sierpinski Gasket - output**



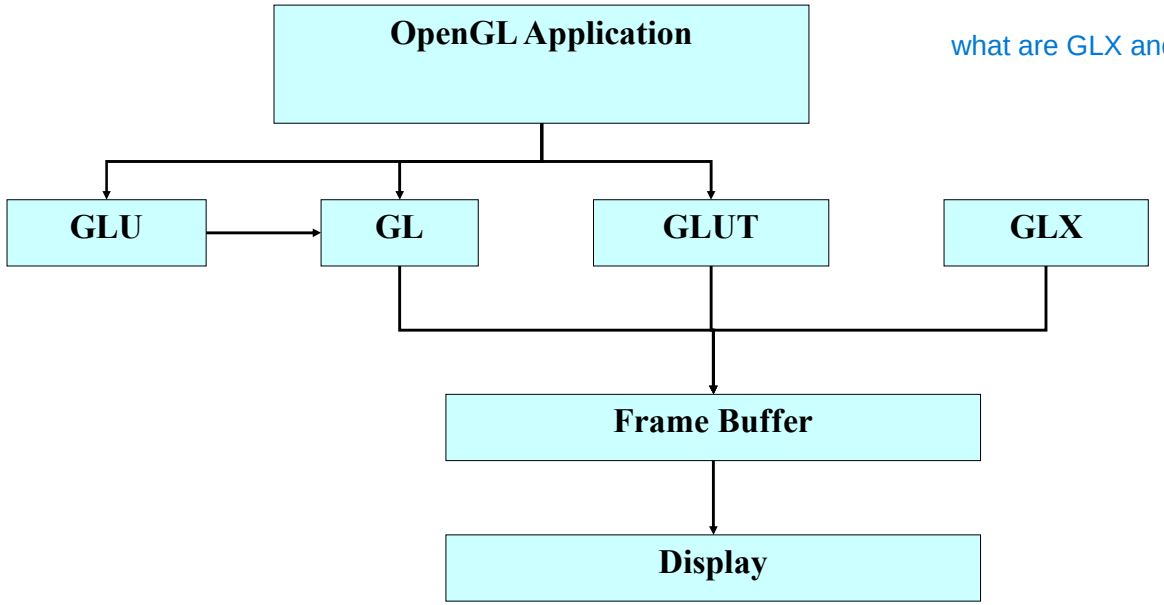
# Graphics Programming using OpenGL


- **The OpenGL API:**

  **GLU** (which stands for the OpenGL utility library) and **GLUT** (an OpenGL Utility Toolkit).

- OpenGL is designed to be a machine independent graphics library
- **The Main Program:** OpenGL was intentionally designed to be independent of any specific window system.
- Basic Window operations are provided by a separate library, called ***GLUT*** or ***OpenGL Utility Toolkit.***
- It is the GLUT toolkit which provides the necessary tools for requesting that windows to be created and providing interaction with I/O devices.

# OpenGL Library Layers



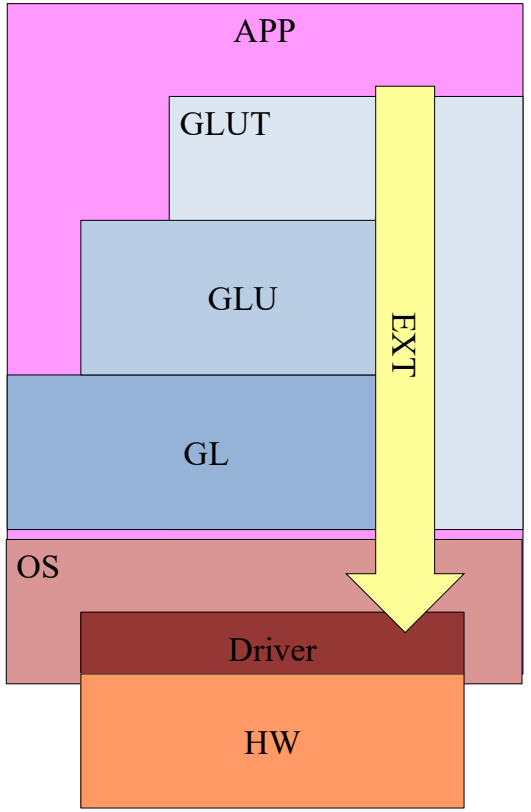

what are GLX and GL

# OpenGL Library Layers

- OpenGL = GL + GLU
  - Basic low-level GL routines implemented using OS graphics routines
  - Timesaving higher-level GLU routines implemented using GL routines
- GLUT opens and manages OpenGL windows and adds helper functions
- OpenGL Extensions provide direct device-dependent access to hardware

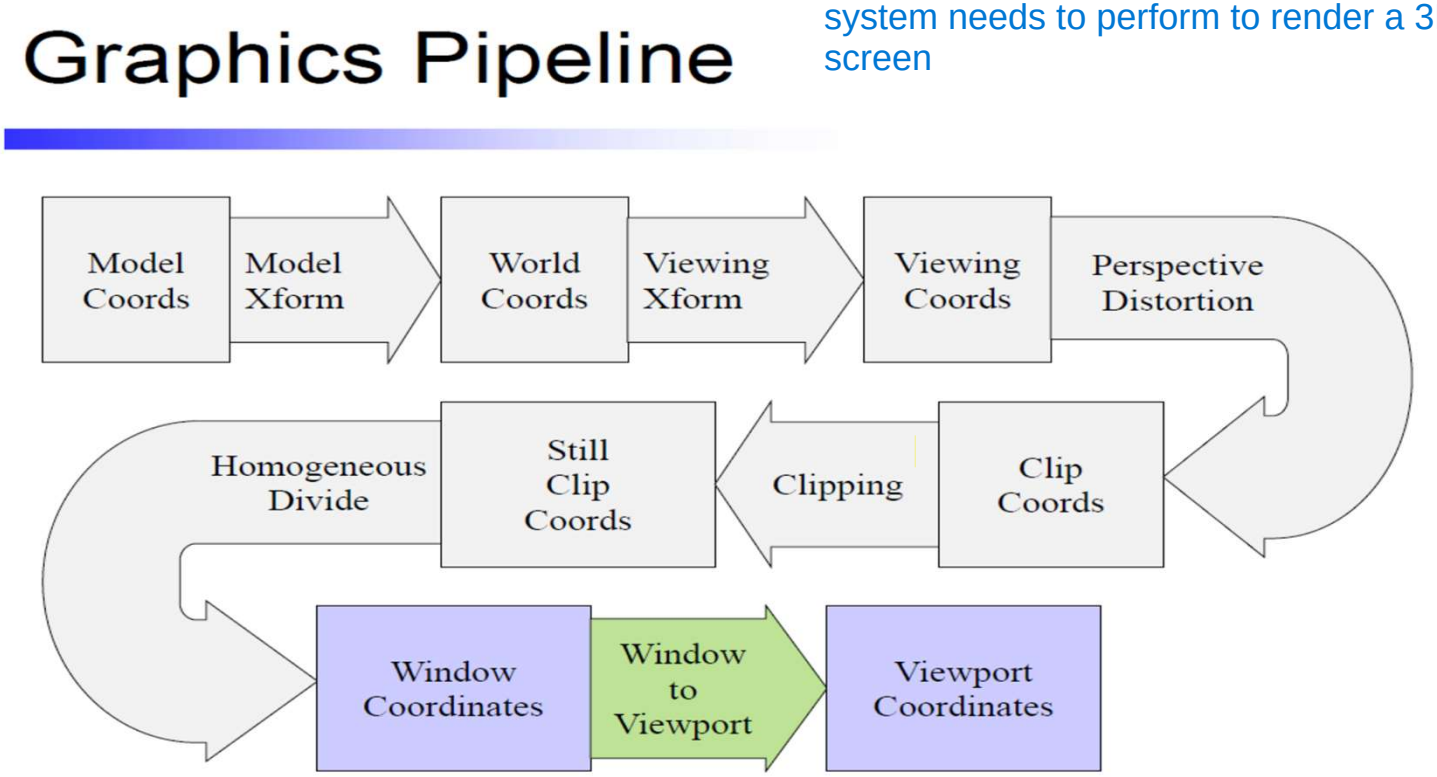
write this diagram

idella diagram nodi baribahudu



# Graphics Pipeline

is a conceptual model that describes what steps a graphics system needs to perform to render a 3D scene to a 2D screen




How can I remember this ?

# General Structure of the program

Section 1: Header files inclusion.

Section 2: Definition of call back function.

Section 3: Main function.

# General Structure of the program

- OpenGL programs often run in an event loop:
  - Start the program
  - Run some initialization code
  - Run an infinite loop and wait for events such as
    - Key press
    - Mouse move, click
    - Reshape window
    - Expose event

# General Structure of the program

- main:
  - find GL visual and create window
  - initialize GL states (e.g. viewing, color, lighting)
  - initialize display lists
  - loop
    - check for events (and process them)
    - if window event (window moved, exposed, etc.)
    - modify viewport, if needed
    - redraw
    - else if mouse or keyboard
    - do something, e.g., change states and redraw
- redraw:
  - clear screen (to background color)
  - change state(s), if needed
  - render some graphics
  - change more states
  - render some more graphics
  - .
  - . …
  - swap buffers

# OpenGL Command Syntax (1)

commands andre? - functions ?

- OpenGL commands start with “gl”
- OpenGL constants start with “GL_” underscore is there ,
- Some commands end in a number and one, two or three letters at the end (indicating number and type of arguments)
- A Number indicates number of arguments
- Characters indicate type of argument

# OpenGL Command Syntax (1)

- `f' float
- `d' double float
- `s' signed short integer
- `i' signed integer
- `b' character
- `ub' unsigned character
- `us' unsigned short integer
- `ui' unsigned integer

# OpenGL Command Syntax (1)

- "v" at the end of the name indicates a vector format. what is vector format?

Examples: glColor*()

see, so many unique functions - none of the openGL functions are overloaded

- **glColor3b**, **glColor3d**, **glColor3f**, **glColor3i**, **glColor3s**, **glColor3ub**, **glColor3ui**, **glColor3us**, **glColor4b**, **glColor4d**, **glColor4f**, **glColor4i**, **glColor4s**, **glColor4ub**, **glColor4ui**, **glColor4us**, **glColor3bv**, **glColor3dv**, **glColor3fv**, **glColor3iv**, **glColor3sv**, **glColor3ubv**, **glColor3uiv**, **glColor3usv**, **glColor4bv**, **glColor4dv**, **glColor4fv**, **glColor4iv**, **glColor4sv**, **glColor4ubv**, **glColor4uiv**, **glColor4usv**

what is viewport and what is window?



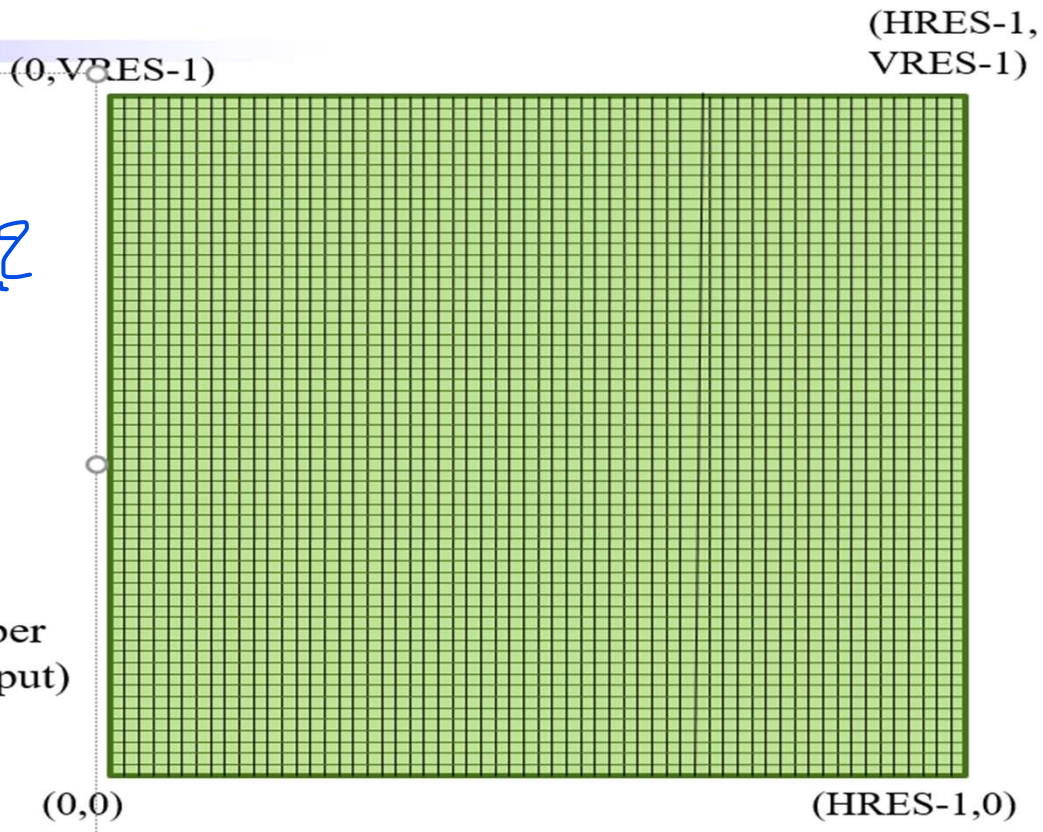
# Viewport Coordinates

- Physical per-pixel integer coordinates
- Also called screen or device coordinates

glViewport(x,y,w,h)

- x,y – lower left pixel
- w – width
- h – height

- Sometimes (0,0) is in the upper left corner (e.g. for mouse input)



# Window Coordinates

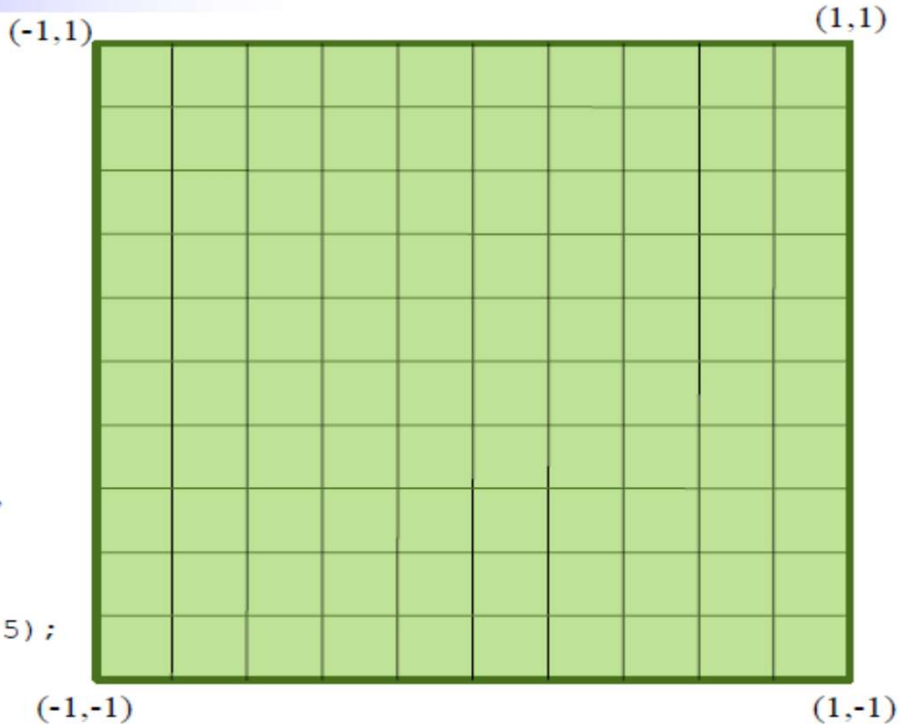
- Logical, mathematical floating-point coordinates

glOrtho(l,r,b,t,n,f)

- left, right, bottom, top
- near, far: limits depth
- gluOrtho2D(l,r,b,t) calls glOrtho(l,r,b,t,-1,1)

- To use per-pixel coordinates, call:

```
gluOrtho2D(-.5,-.5,w-.5,h-.5);
```

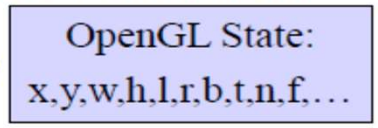


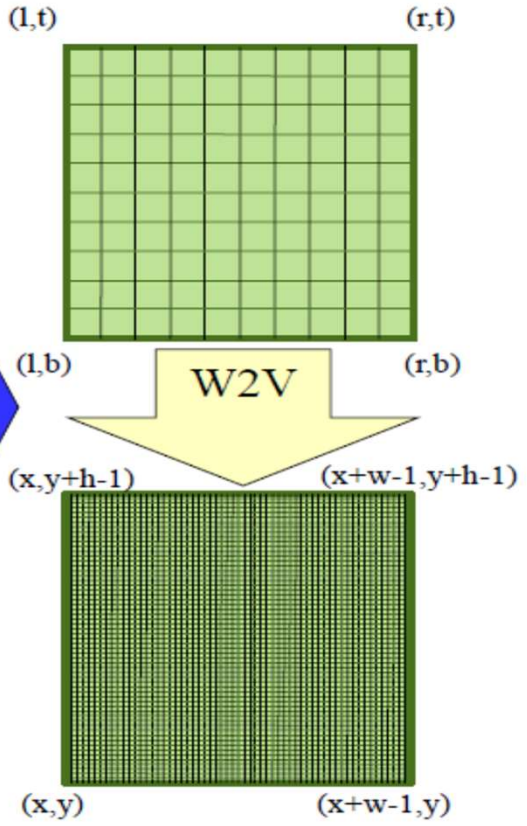
# OpenGL State

- OpenGL commands change its internal *state* variables which control how it turns geometric models into pictures

- E.g. the window-to-viewport transformation

glViewport(x,y,w,h)
glOrtho(l,r,b,t,n,f)



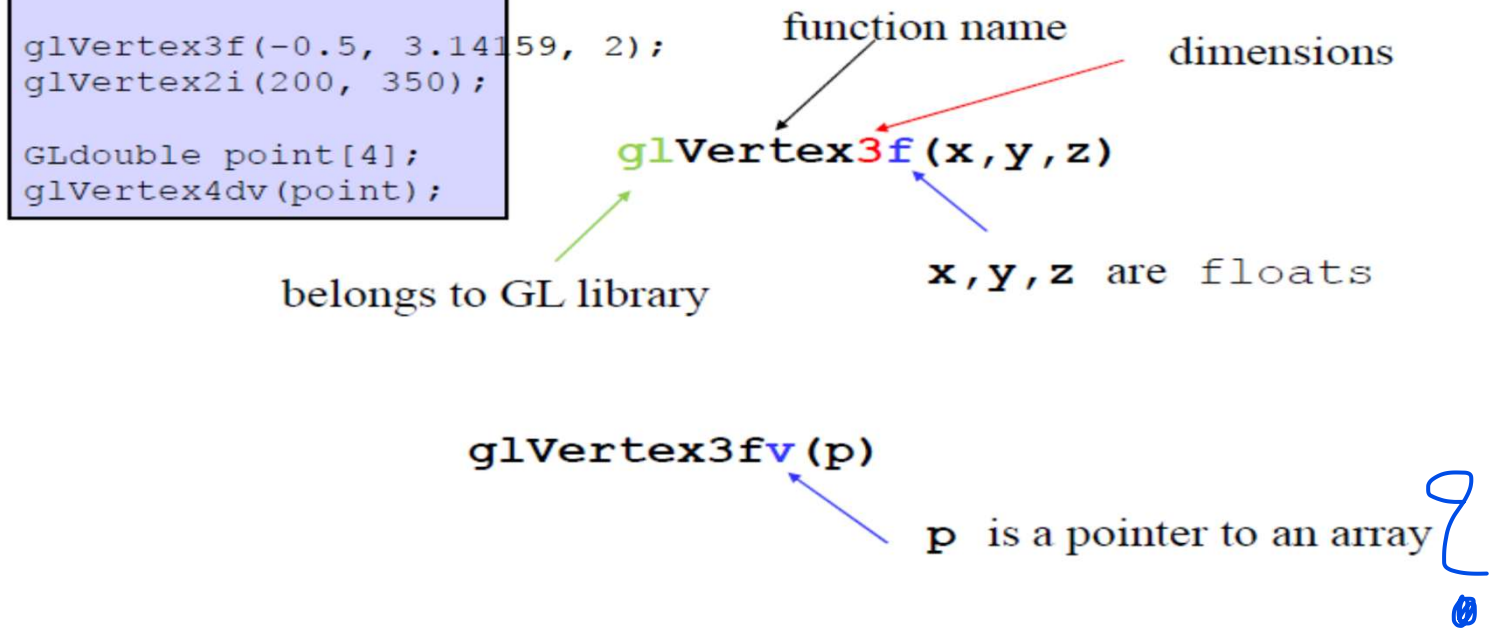

- Can query this state:

```
GLfloat buf[4];
glGetFloatv(GL_VIEWPORT,buf);
x = buf[0]; y = buf[1];
w = buf[2]; h = buf[3];
```

# OpenGL function format

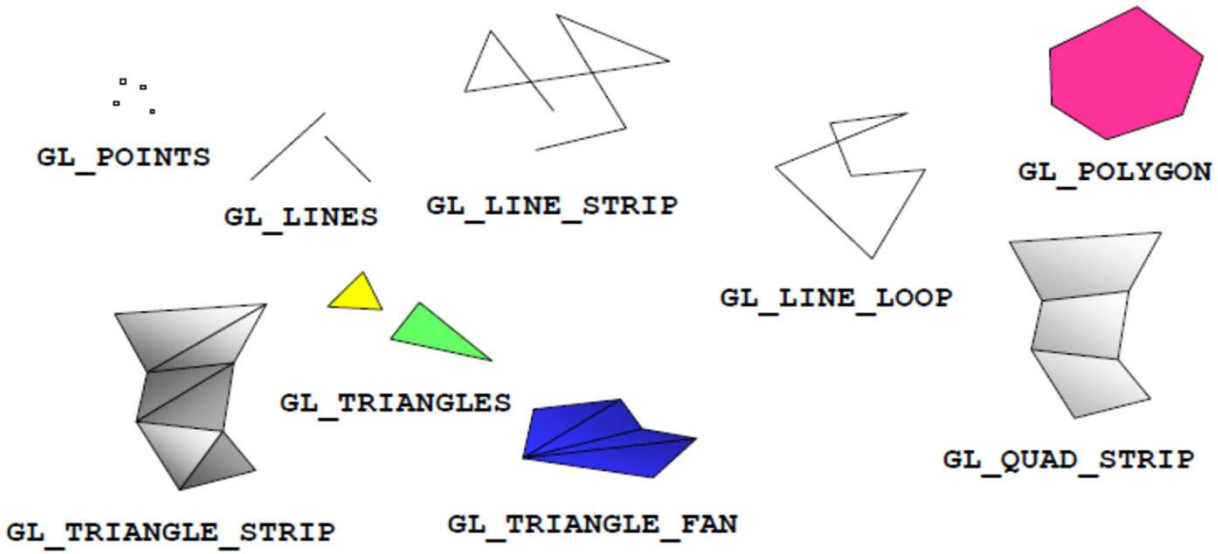
Examples:

```
glVertex3f(-0.5, 3.14159, 2);
glVertex2i(200, 350);

GLdouble point[4];
glVertex4dv(point);
```



# OpenGL Primitives

10




GL_POINTS

GL_LINES

GL_LINE_STRIP

GL_LINE_LOOP

GL_TRIANGLES

GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

GL_QUADS

GL_QUAD_STRIP

GL_POLYGON

# OpenGL Command Syntax (1)

what does glBegin do?

function *glBegin()* lists the ten possible arguments and the corresponding type of primitive.

**void *glBegin*(GLenum *mode*);**

Marks the beginning of a vertex list that describes a geometric primitive. The type of primitive is indicated by *mode*, which can be any of the values shown in **Table (below)**.

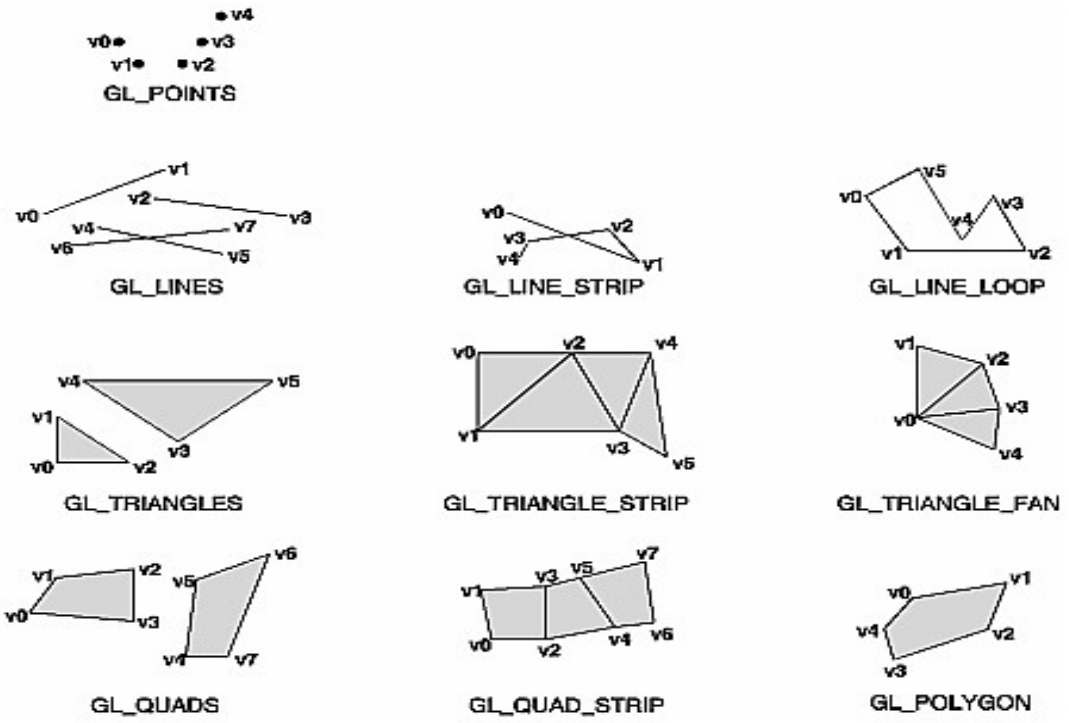
| Value | Meaning |
|---|---|
| GL_POINTS | individual points |
| GL_LINES | pairs of vertices interpreted as individual line segments |
| GL_POLYGON | boundary of a simple, convex polygon |
| GL_TRIANGLES | triples of vertices interpreted as triangles |
| GL_QUADS | quadruples of vertices interpreted as four−sided polygons |
| GL_LINE_STRIP | series of connected line segments |
| GL_LINE_LOOP | same as above, with a segment added between last and first vertices |
| GL_TRIANGLE_STRIP | linked strip of triangles |
| GL_TRIANGLE_FAN | linked fan of triangles |
| GL_QUAD_STRIP | linked strip of quadrilaterals |

# OpenGL Command Syntax (1)

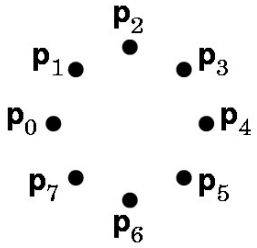
- Primitives are specified using

```
glBegin( primType );
glVertex…
…
glEnd();
```

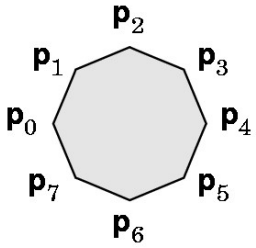
  - *primType* determines how vertices are combined



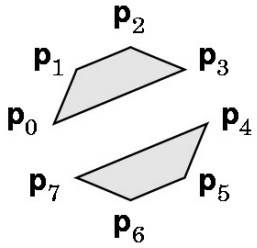
# OpenGL Command Syntax (1)



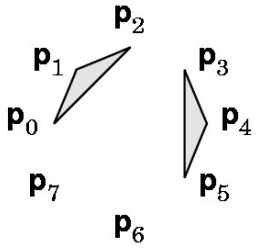

GL_POINTS



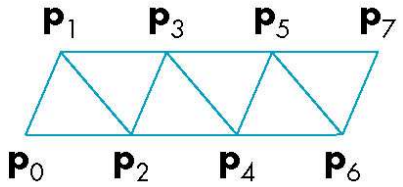

GL_POLYGON



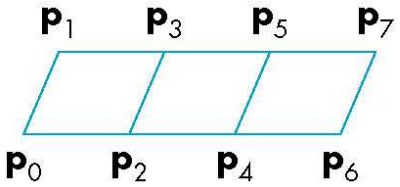

GL_QUADS



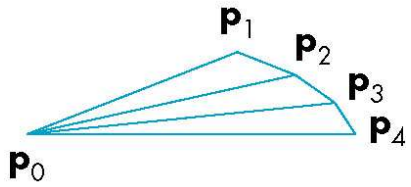

GL_TRIANGLES



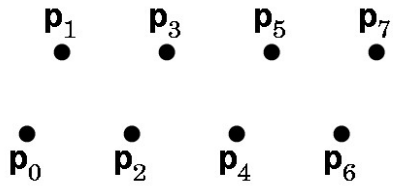

GL_TRIANGLE_STRIP



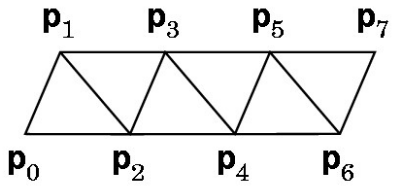

GL_QUAD_STRIP



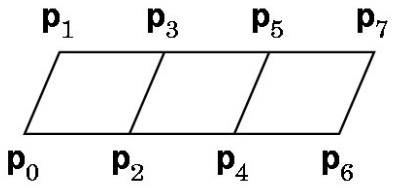

GL_TRIANGLE_FAN




GL_POINTS




GL_TRIANGLE_STRIP

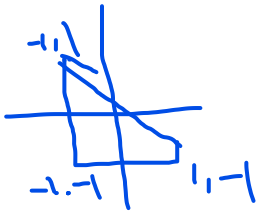



GL_QUAD_STRIP

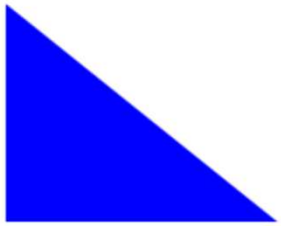
# Assigning Color

didn't understand this vertex numbering ?
understood now

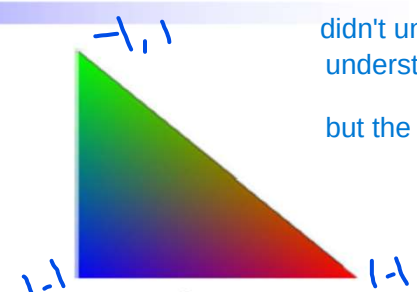
but the code?



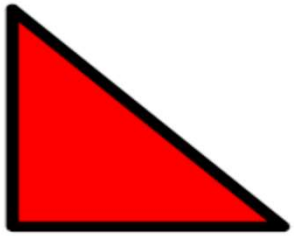
also this code:(



```
glColor3f(0,0,1);

glBegin(GL_POLYGON);
  glVertex2f(-1,1);
  glVertex2f(-1,-1);
  glVertex2f(1,-1);
glEnd();
```



```
glBegin(GL_POLYGON);
  glColor3f(0,1,0);
  glVertex2f(-1,1);

  glColor3f(0,0,1);
  glVertex2f(-1,-1);

  glColor3f(1,0,0);
  glVertex2f(1,-1);
glEnd();
```



```
glColor3f(1,0,0);
glBegin(GL_POLYGON);
  glVertex2f(-1,1);
  glVertex2f(-1,-1);
  glVertex2f(1,-1);
glEnd();

glColor3f(0,0,0);
glBegin(GL_LINE_LOOP);
  glVertex2f(-1,1);
  glVertex2f(-1,-1);
  glVertex2f(1,-1);
glEnd();
```

I still don't know what glBegin() and gelEnd() do

# Graphics Functions

**Graphics functions can be grouped as:**

**1.Primitive functions**

**2.Attribute functions**

**3.Viewing functions**

**4.Transformation functions**

**5.Input functions**

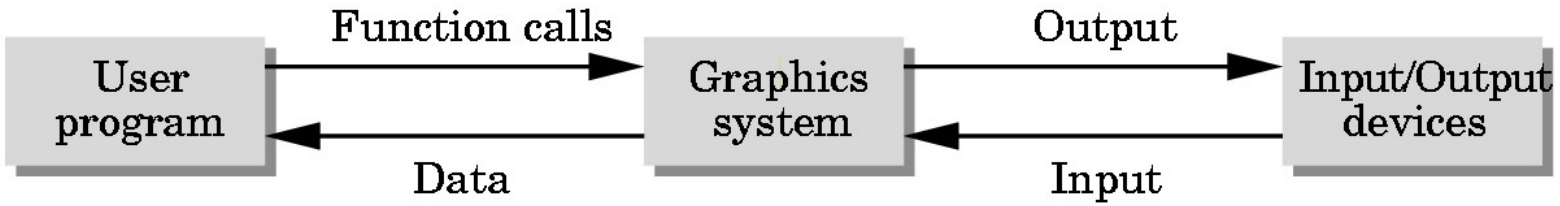
**6.Control functions**

**7.Query functions**

# Graphics Functions

practice this

Graphics package as a BLACK BOX

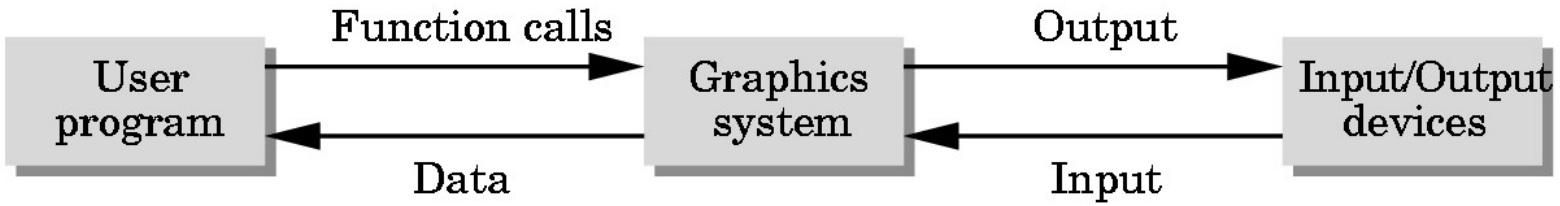



Hence graphics system can be viewed as a box whose

- inputs are function calls from a user program (say measurements from input devices, such as the mouse and keyboard and possibly other input, such as messages from the operating system)
- outputs are primarily the graphics sent to the output devices

# Graphics Functions

Graphics package as a BLACK BOX




Functionality can be grouped to:

leave this slide

1. *primitive functions* – low level primitives (points, lines, polygons, text, etc.)
2. primitives API – what should be displayed *attribute functions* are – how should be displayed –lines, points, polygons etc.
3. Synthetic camera must describe how to create an image – *viewing functions* (camera properties – lens, viewing field, several cameras, stereoscopic views etc.)

# Graphics Functions

4. *geometric transformation* – scaling, rotation, translation, sheering etc. and use of ***transformation matrix.*** This forms the main advantage as the geometric engine can be used for substantial speed up.
5. *input functions* – allow interaction with graphics primitives or objects; functions for communication with a keyboard, mouse, data tablets etc.
6. *control functions* – initialization, errors handling etc. in multiprocessor multi-window & network environment
7. *Query functions* - to know properties of a particular implementation(e.g Color ).

leave this slide

# Graphics Functions

***Primitive functions*** enable the programmer to create some ***basic primitives*** such as points, line segments, polygons, text etc.

***Example :***

look at how * is used

```
glVertex*();
glBegin();
glEnd();
```

# Graphics Functions

***Primitive functions*** enable the programmer to create some ***basic primitives*** such as points, line segments, polygons, text etc.

***Example :***

glVertex*();

glBegin();

glEnd();

**Example :**

/* Draw a triangle */

above - we have used polygon

```
glBegin (GL_LINE_LOOP);
   glVertex3f(-0.3, -0.3, 0.0);
   glVertex3f(0.0, 0.3, 0.0);
   glVertex3f(0.3, -0.3, 0.0);
glEnd();
```

# Graphics Functions

***Attribute functions*** perform operations ranging from ***choosing the color***, to ***picking a pattern*** with which to fill the inside of a polygon, to selecting a type face for the titles on a graph.

***Example :***

```
glColor3f(1.0,0.0,0.0);
glClearColor(1.0,1.0,1.0,1.0);
```

all ones- white  why clear color then

***Viewing functions*** enable the programmer to specify ***views*** such as front view, top view, side view. Also provide ***zoom in*** and ***zoom out*** facilities.

***Example :***

```
glViewport();
glMatrixMode(GL_PROJECTION);
glMatrixMode(GL_MODELVIEW);
gluOrtho2D();
```

# Graphics Functions

***Transformation Functions*** enable the programmer to carryout geometric transformations such as ***rotation, scaling, translation***.

***Example :***

glTranslatef();

glRotatef();

***Input functions*** enable the programmer to design interactive programs.

These functions can accept ***inputs from mouse, keyboard, data tablets*** etc.

***Example :***

glutMouseFunc(mouse);

# Graphics Functions

***Control functions*** enable the programmer to ***communicate*** with the window system, initialize programs, deal with errors that occur during the execution of the program etc.

***Example :***

glutInitDisplayMode();

glutInitWindowSize();

glutInitWindowPosition();

***Query functions*** enable the programmer to access information such as ***camera parameters***, contents of the frame buffer etc which the programmer can design device independent programs.

***Example :***

glGetBooleanV();

glGetIntegerV();

# OpenGL – Text

Graphical output in applications requires text output

Fonts – Times, Helvetica, Computer Moderr etc.

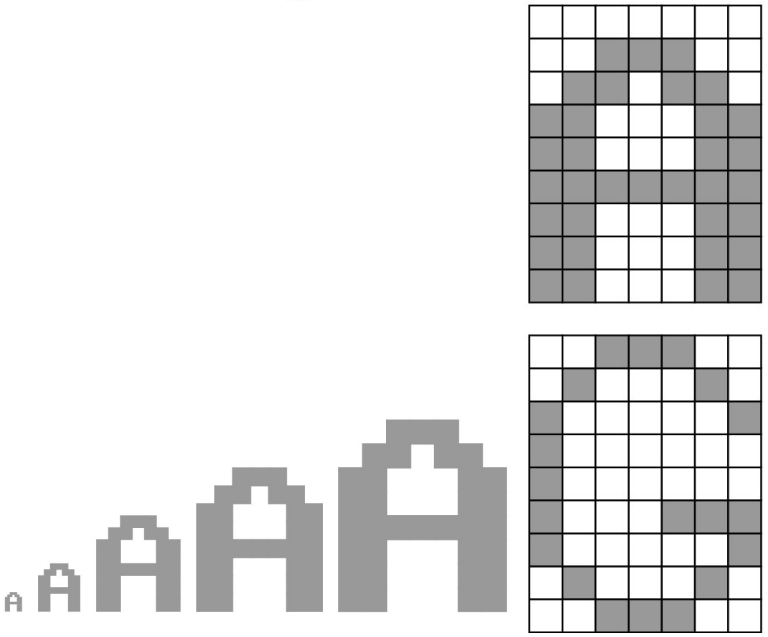
Forms – **stroke vs. raster text**

**stroke** – constructed as other graphics primitives – points etc. – the standard PostScript fonts defined by polynomial curves – easy to enlarge

**raster** – fast; using bit-block-transfer (bitblt) operations

Computer

Graphics



# OpenGL – Curved Objects

- **Curved primitives** (like circles, spheres, curves and surfaces etc.) are not directly supported – must be approximated by a mesh polygons – a tessellation – at the rendering stage or within the user program

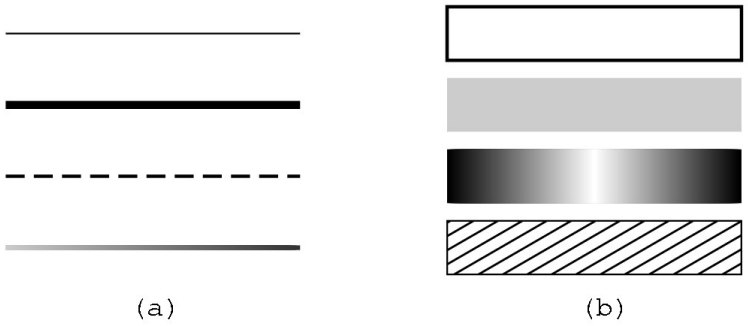
- **GLU** can be used to approximate common curved surfaces

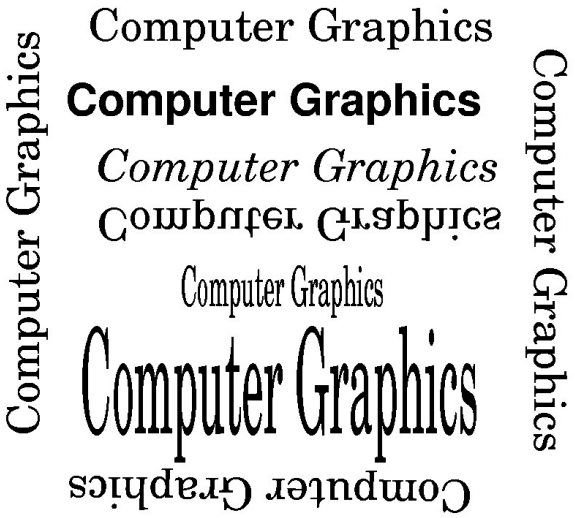
- **Advanced OpenGL functions** can be used to work with parametric curves and surfaces

# OpenGL – Attributes

- Distinction between
- what the type of a primitive is
- how the primitive is displayed
- A red solid line and green dashed line are the the same geometric type, but displayed differently due to *attributes*

Eg



# OpenGL – Color

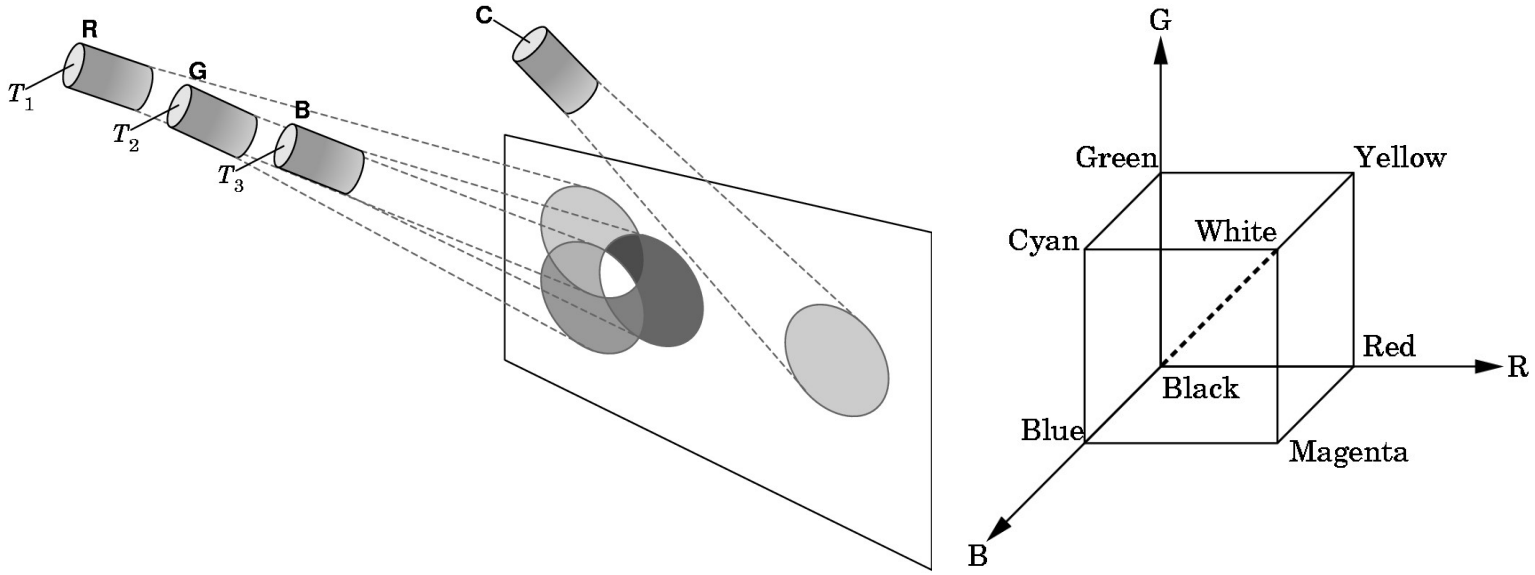
- Color is one of the most interesting aspects of both human perception and computer graphics
- Light is a part of electromagnetic spectrum 350 – 780 nm

- Color $C = rR + gG + bB$ what are R. G. B then
- *r,g,b* are *tristimulus values*

?

- assumption of three-color theory
  *if two colors produce the same tristimulus values then they are visually indistinguishable*

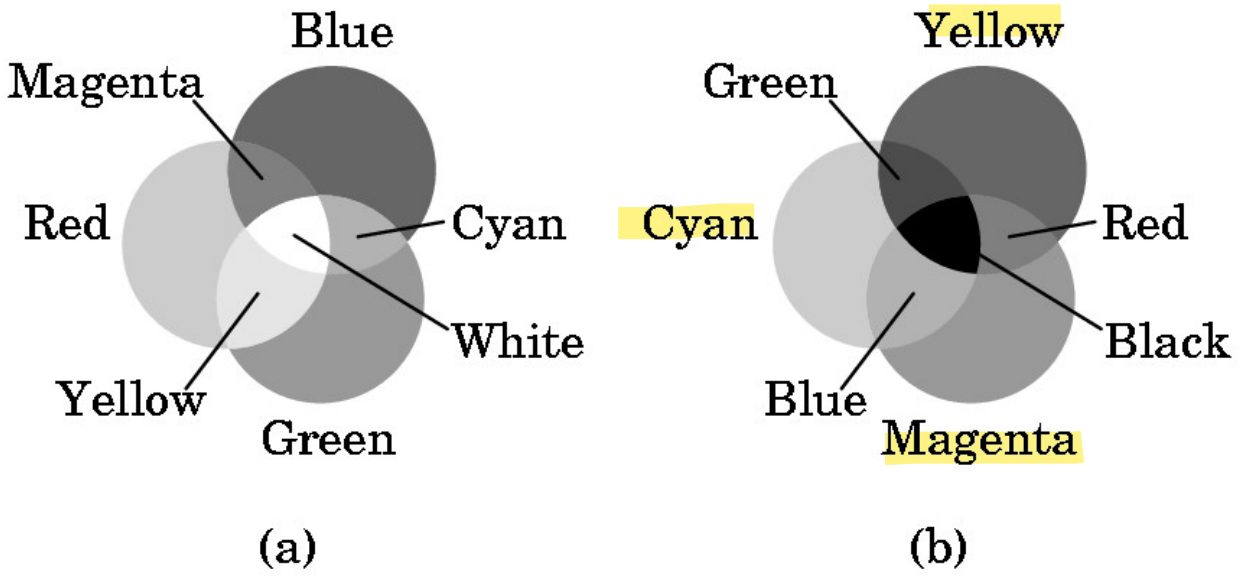
# OpenGL – Color




Color space can be represented as RGB cube

Gray levels are represented by the Black-White line segment

# OpenGL – Color

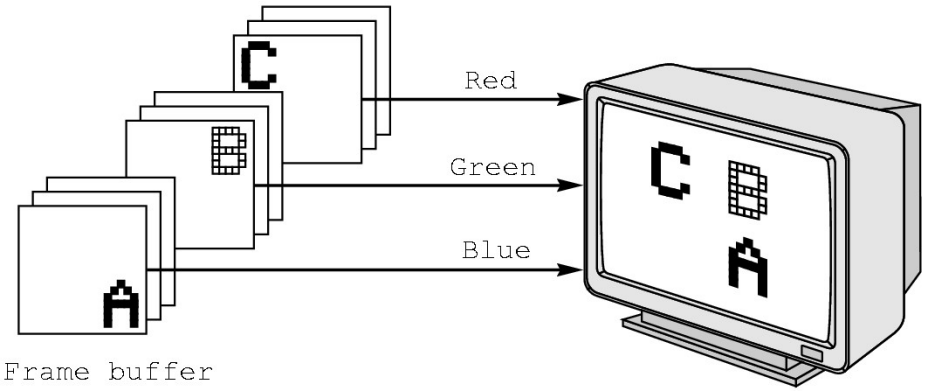



Range of colors that can be produced on the given system is called color gamut

**Additive vs. subtractive color system**

**RGB vs. CMY (K) color systems**

# OpenGL – RGB Color




**Additive RGB primary color system** is used

- 8 bits/pixel for each channel R,G,B $2^{24}$ possible colors
- 1024 x 1024 output resolution
- 3MB RAM for graphics purposes

**RGBA system (Alpha channel) – for transparency/opacity**

glColor3f(1.0, 0.0, 0.0)

glClearColor(1.0, 1.0, 1.0, 1.0) – screen solid and white

# OpenGL – RGB Color – Indexed Color

| Input | | Red | Green | Blue |
|---|---|---|---|---|
| 0 | | 0 | 0 | 0 |
| 1 | | $2^m - 1$ | 0 | 0 |
| . | | 0 | $2^m - 1$ | 0 |
| . | | . | . | . |
| . | | . | . | . |
| $2^k - 1$ | | . | . | . |
| | | $m$ bits | $m$ bits | $m$ bits |

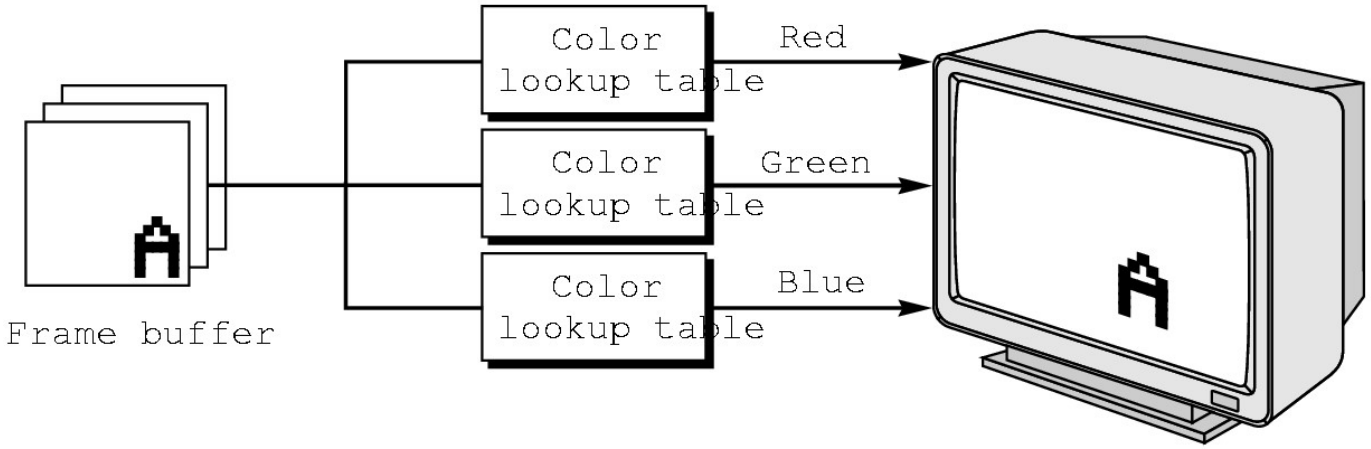
Color palette enables to change in the whole image

Limit of the frame buffer to *k* bits per pixel

$2^k - 1$ colors can be displayed from a palette of $2^{3m}$ of colors table of $2^k \times 3m$ is to be specified

for k=m=8 – a common configuration – 256 colors out of 16M. 256 entries constitute the user's color palette

# OpenGL – RGB Color – Indexed Color




In color-index mode – present color is selected by
glIndexi(element) – selects particular color out of the table

System can have just one common table or multiple tables for different windows – GLUT allows that

glutSetColor(int color, GLfloat red, GLfloat blue, GLfloat green)

- we will work directly with RGB colors (not indexed)

# Setting of Color Attributes

For the Sierpinski gasket program we need to set:

clear color

```
glClearColor(1.0, 1.0, 1.0, 1.0);
```

rendering color for points – set to RED

```
glColor3f(1.0, 0.0, 0.0);
```

set point size – let us use 2 pixels size

```
glPointSize(2.0); /* size is in pixels not in [mm] */
```

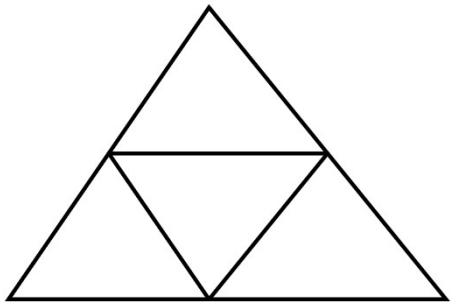
These attributes will be valid for all the time if not changed explicitly.

# Sierpinski Gasket Program - triangles

A similar approach – a triangle is split to triangles recursively

original triangle given by the array point2 v[3];

```
void triangle (point2 a, point2 b, point2 c)
{ glBegin(GL_TRIANGLES);
  glVertex2fv(a); glVertex2fv(b); glVertex2fv(c);
  glEnd( );
}
------ mid-points computations ------------
for (j=0; j<2; j++) m[0][j]=(v[0][j]+ v[1][j])/2.0;
for (j=0; j<2; j++) m[1][j]=(v[0][j]+ v[2][j])/2.0;
for (j=0; j<2; j++) m[2][j]=(v[1][j]+ v[2][j])/2.0;
```



# Sierpinski Gasket Program - triangles

Modification for recursive triangles

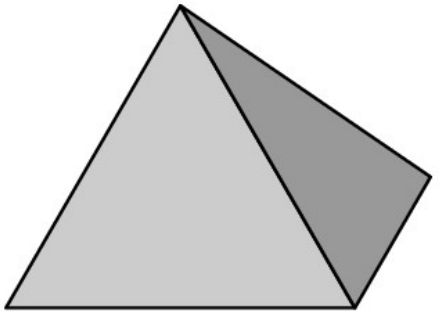
```
void display(void)
{  glClear(GL_COLOR_BUFFER_BIT);
   divide_triangle(v[0], v[1], v[2], n);
   /* n determines No of subdivision steps */
   glFlush( );
}
```

**Note**: *there is no convenient way to pass variables to OpenGL functions and callback other than through global parameters*

# Sierpinski Gasket Program - triangles

```
void divide_triangle(point2 a, point2 b, point2 c, int k)
{ point2 ab, ac, bc;   int j;
  if ( k>0) {   /* compute midpoints */
          for (j=0; j<2; j++) ab[j]=(a[j]+ b[j])/2.0;
          for (j=0; j<2; j++) ac[j]=(a[j]+ c[j])/2.0;
          for (j=0; j<2; j++) bc[j]=(b[j]+ c[j])/2.0;
          /* subdivide all but inner triangle */
          divide_triangle(a, ab, ac, k-1);
          divide_triangle(c, ac, bc, k-1);
          divide_triangle(b, bc, ab, k-1);
     } else triangle(a,b,c);
     /*draw a triangle at end of recursion */
}
```

# 3D Sierpinski Gasket Program



Now we expand 2D to 3D case

data type definition:

```
typedef GLfloat point3[3];
```

```
int n; /* global variable */
```

tetrahedron specifications:

```
point3 vertices[4] = { {0.0,0.0,0.0}, {250.0,500.0,100.0},
    {500.0,250.0,250.0}, {250.0,100.0,250.0}};
point3 p= {250.0,50.0,250.0};
```

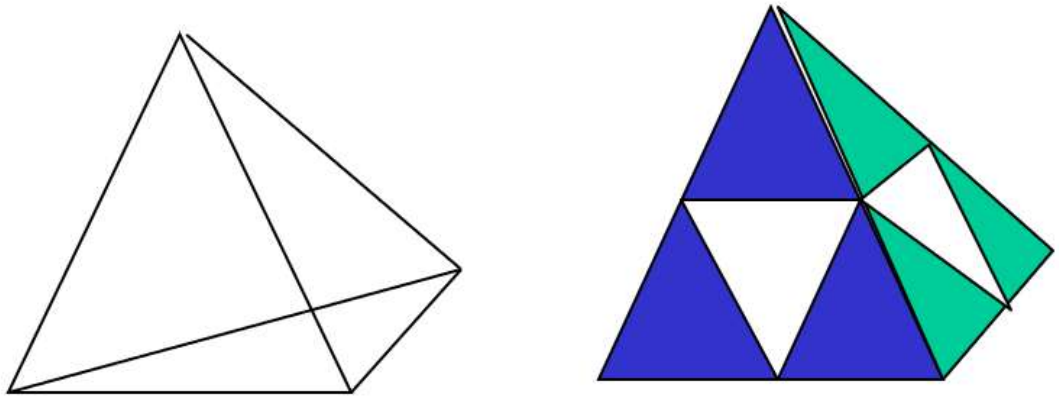
function glPoints3fv is used to define points

different colors used for points to distinguish them

drawing polygons – visibility should be solved

# 3D Sierpinski Gasket Program

- We can subdivide each of the four faces



- Appears as if we remove a solid tetrahedron from the center leaving four smaller tetrahedra

# 3D Sierpinski Gasket Program

```
void triangle (point3 a, point3 b, point3 c)
{ point3 norm;
  cross(a, b, c, norm);
  /* computes the normal vector of the triangle */
  glBegin(GL_POLYGON);
  glNormal3fv(a);
  glVertex3fv(a); glVertex3fv(b); glVertex3fv(c);
  glEnd( );
}
/* normal computation  - using cross product */
```

# 3D Sierpinski Gasket Program

```
void divide_triangle(point3 a, point3 b, point3 c, int k);
{  point3 v1, v2, v3; int j;
   if ( k>0) {      /* compute midpoints */
                for (j=0; j<3; j++) v1[ j]=(a[ j]+ b[ j])/2.0;
                for (j=0; j<3; j++) v2[ j]=(a[ j]+ c[ j])/2.0;
                for (j=0; j<3; j++) v3[ j]=(b[ j]+ c[ j])/2.0;
                divide_triangle(a, v1, v2, k-1);
                divide_triangle(c, v2, v3, k-1);
                divide_triangle(b, v3, v1, k-1);
        } else triangle(a,b,c);
        /*draw a triangle at end of recursion */
}
```

# 3D Sierpinski Gasket Program

```
void tetrahedron (int m);
{ /* apply triangle subdivision to faces of tetrahedron */
  glColor3f(1.0, 0.0, 0.0);
      divide_triangle (v[0], v[1], v[2], m);
  glColor3f(0.0, 1.0, 0.0);
      divide_triangle (v[3], v[2], v[1], m);
  glColor3f(0.0, 0.0, 1.0);
      divide_triangle (v[0], v[3], v[1], m);
  glColor3f(0.0, 0.0, 0.0);
      divide_triangle (v[0], v[2], v[3], m);
}
```

# 3D Sierpinski Gasket Program

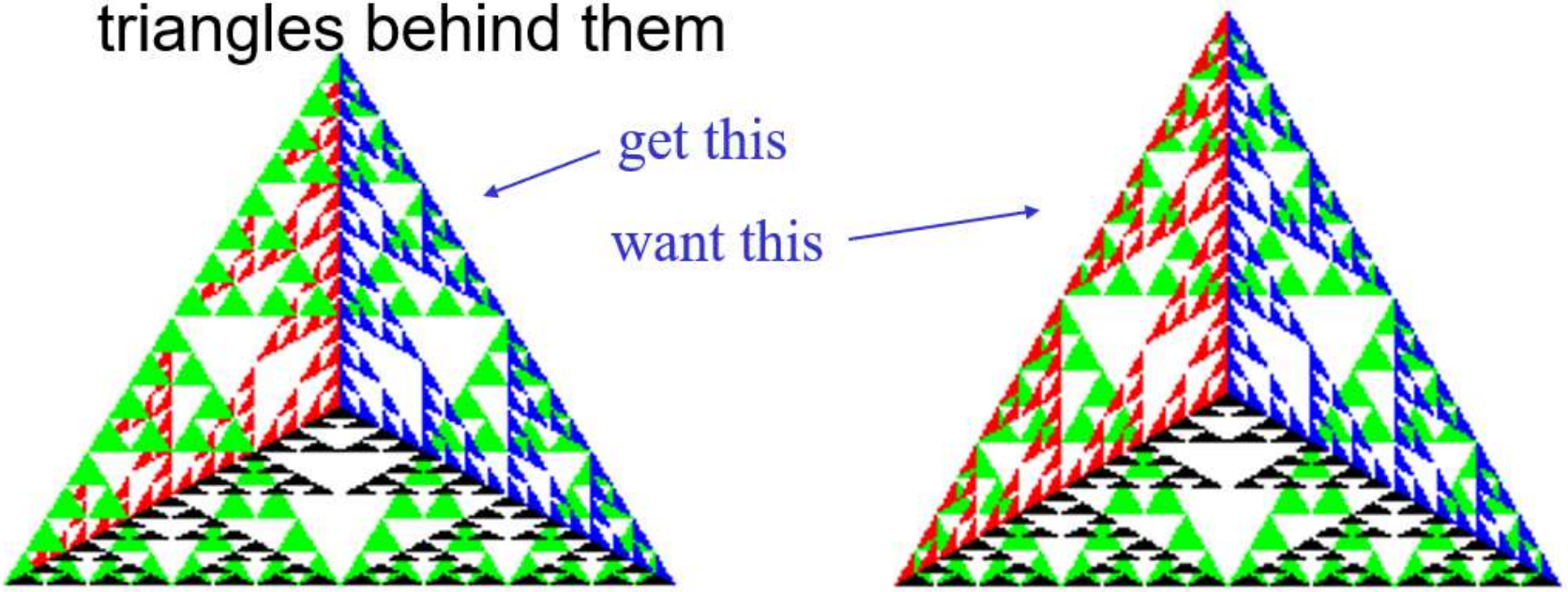
```
void display ( );
{  glClear(GL_COLOR_BUFFER_BIT | GLDEPTH_BUFFER_BIT);
   glLoadIdentity ();  tetrahedron(n);        glFlush ( );
}
void myReshape(int w, int h)
{  glViewport (0,0,w,h);    glMatrixMode(GL_PROJECTION);
   glLoadIdentity ();
   if (w<=h)
   glOrtho(-2.0,2.0, (GLfloat)h/(GLfloat)w, (GLfloat)h/(GLfloat)w, -10.0, 10.0);
   else
   glOrtho(-2.0,2.0, (GLfloat)w/(GLfloat)h, (GLfloat)w/(GLfloat)h, -10.0, 10.0);
   glMatrixMode(GL_MODELVIEW);
   glutPostRedisplay ( );
}
```

# 3D Sierpinski Gasket Program

```
void main ( int argc, char **argv);
{  n = atoi(argv[1]); /* parameter on the command line */
   /* n is a global variable */
   glutInit ( int argc, char **argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize (500,500);
   glutCreateWindow ("3D Gasket");
   glutReshapeFunc(myReshape);
   glutDisplayFunc(display);
   glEnable(GL_DEPTH_TEST);
   glClearColor (1.0, 1.0, 1.0, 1.0);
   glutMainLoop( );
}
```

# 3D Sierpinski Gasket Program

- Because the triangles are drawn in the order they are defined in the program, the front triangles are not always rendered in front of triangles behind them



# 3D Sierpinski Gasket Program

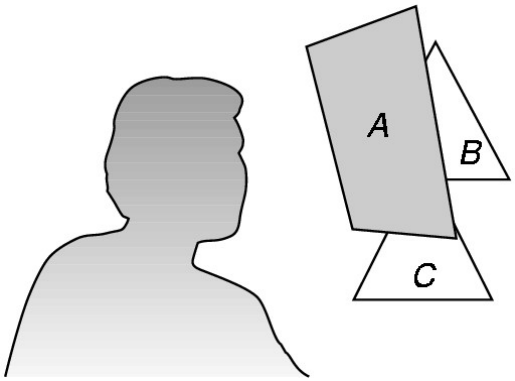
Triangles are drawn in order of generation (given by the algorithm) not according to their geometric position

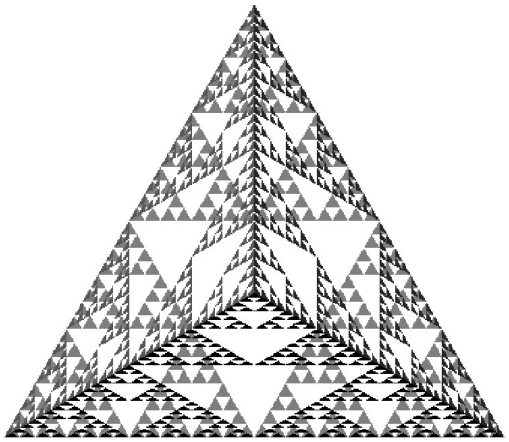
Hidden surface removal is enabled by setting

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

and enable that by (using z-buffer algorithm)

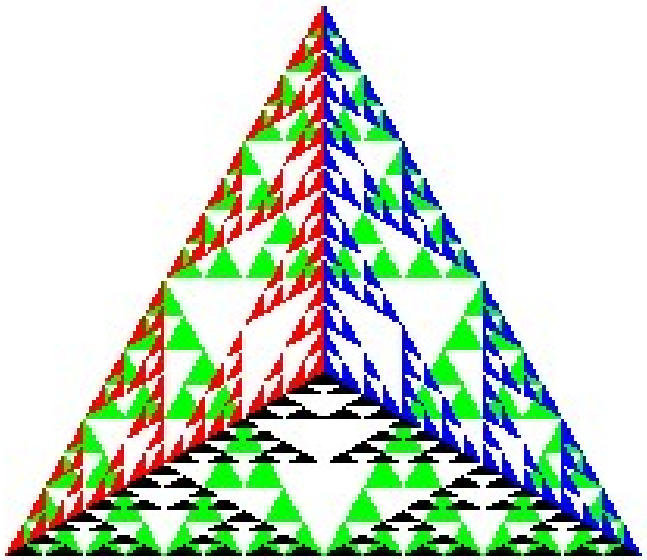
```
glEnable(GL_DEPTH_TEST);
```



# 3D Sierpinski Gasket Program

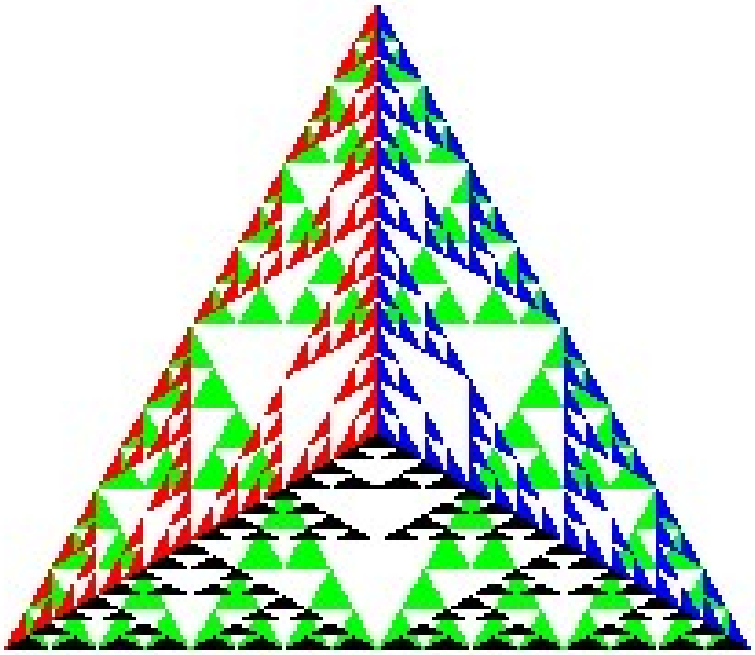
after 5 iterations

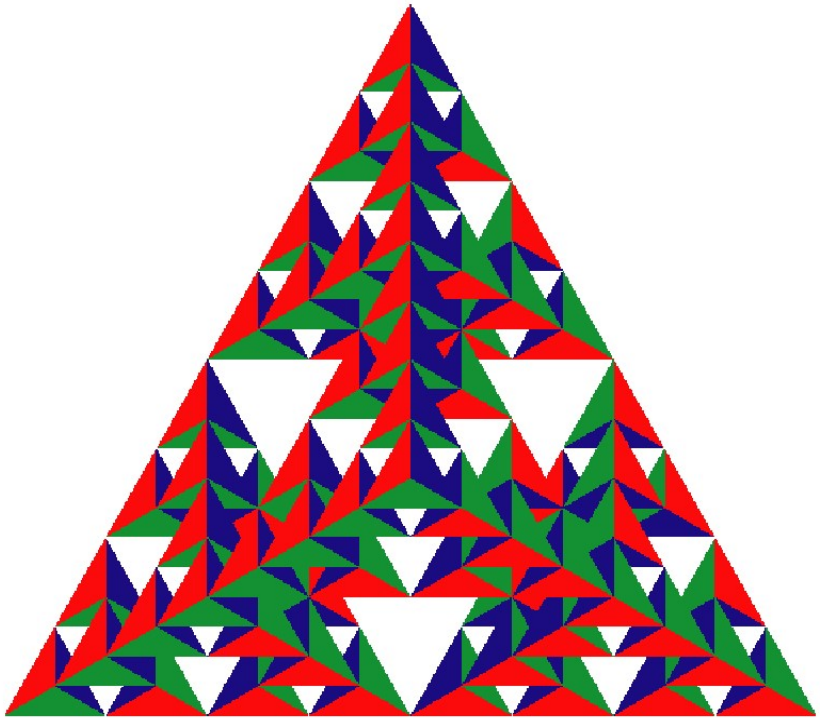


# Surface vs Volume Subdvision

- In our example, we divided the surface of each face
- We could also divide the volume using the same midpoints
- The midpoints define four smaller tetrahedrons, one for each vertex
- Keeping only these tetrahedrons removes a *volume* in the middle
- See text for code

# Surface vs Volume Subdvision





# Curved Primitives

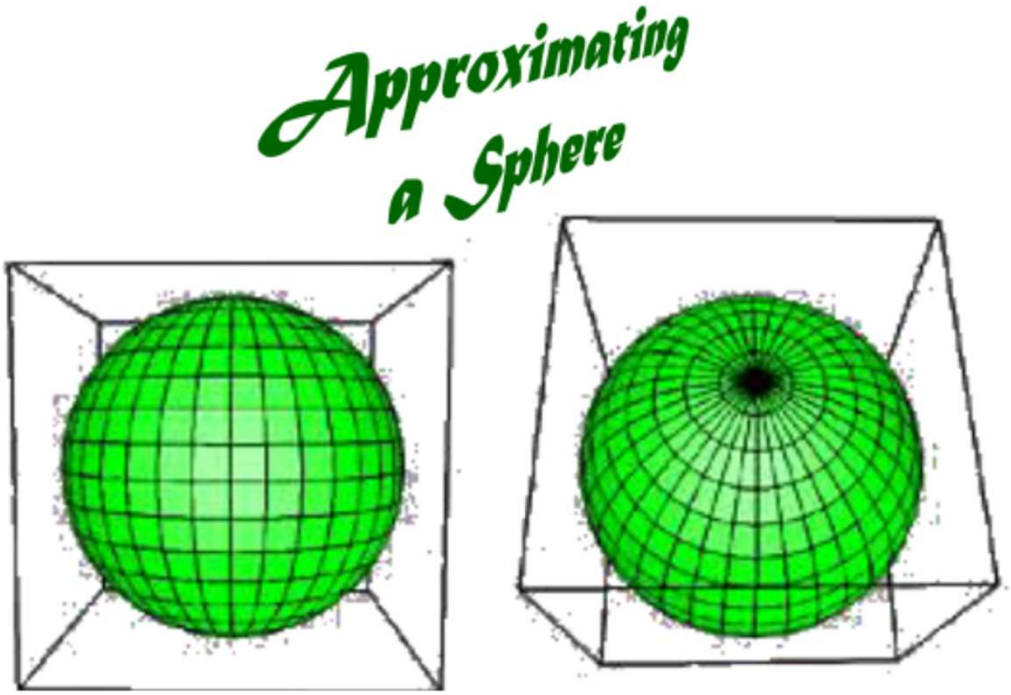
Curved primitives such as sphere, oval, parabola can be created

--------- using ***tessellation.***

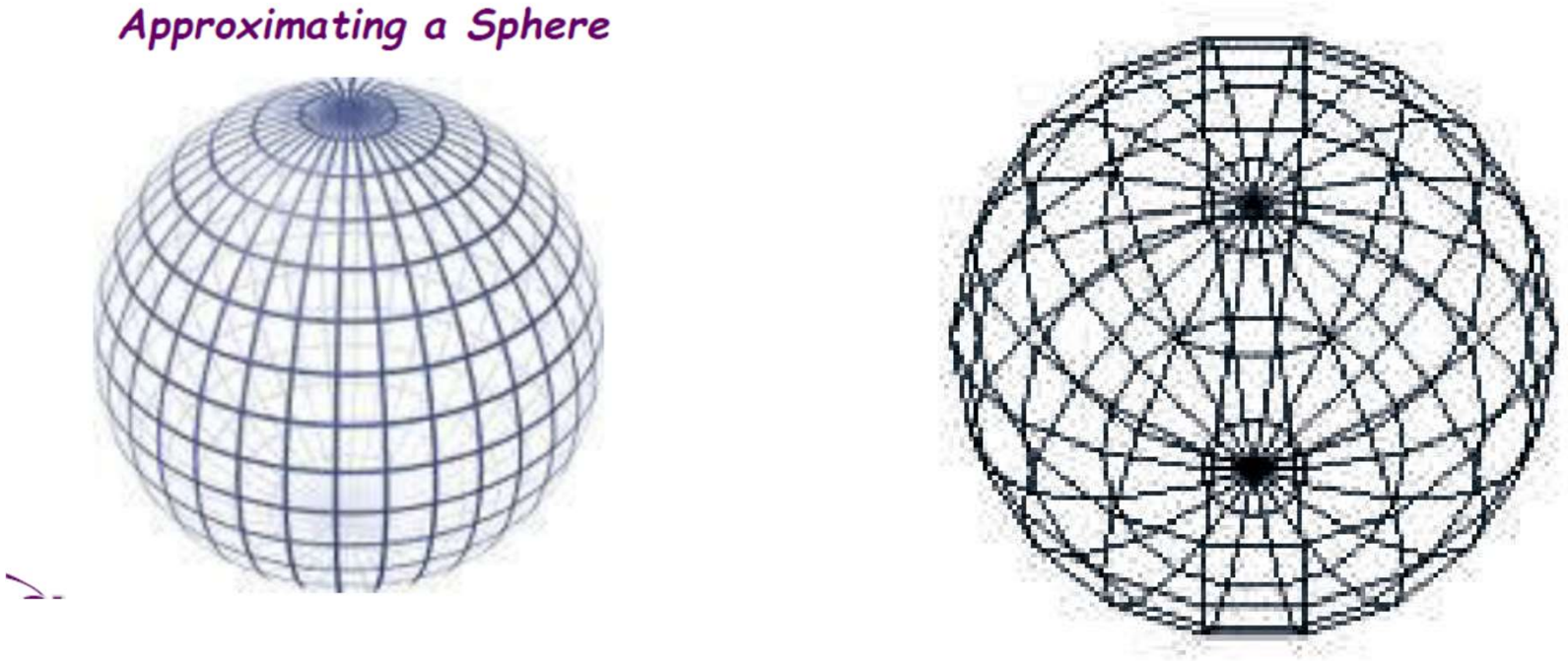
***Tessellation*** is a process of creating curved primitives by using a mesh of polygon of n sides.

***Example*** : Approximating a Sphere

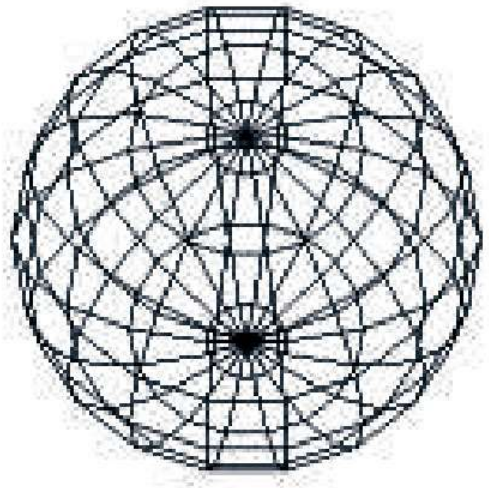
# Curved Primitives



# Drawing a Sphere



# Drawing a Sphere



## Approximating a Sphere

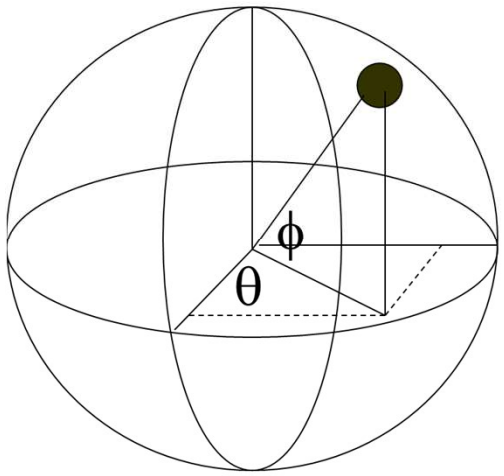
- Many curved surfaces can be approximated using fans and strips.
- E.g.To approximate to a sphere, **a set of polygons defined by lines of longitude and latitude** as shown can be used.
- Either quadstrips or trianglestrips can be used for the purpose.
- Consider a unit sphere(r=1). It can be described by the following three equations:

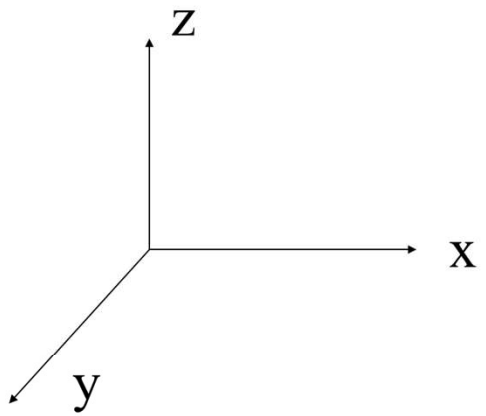
$$x(\theta,\emptyset) = \sin\theta \cos\emptyset$$

$$y(\theta,\emptyset) = \cos\theta \cos\emptyset$$

$$z(\theta,\emptyset) = \sin\emptyset$$

# Drawing a Sphere




$$x(\theta,\phi) = \sin\theta\cos\phi$$

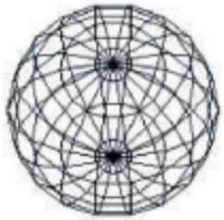
$$y(\theta,\phi) = \cos\theta\cos\phi$$

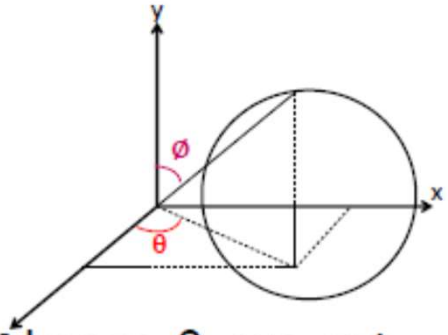
$$z(\theta,\phi) = \sin\phi$$

# Drawing a Sphere

## Approximating a Sphere

$x(\theta, \phi) = \sin\theta \cos\phi$
$y(\theta, \phi) = \cos\theta \cos\phi$
$z(\theta, \phi) = \sin\phi$






- If we **fix Ø** and draw curves, as we change θ, we get circles of **constant latitude.**
- If we **fix θ** and draw curves, as we change Ø, we get circles of **constant longitude.**
- By generating points at fixed increments of θ and Ø, we can define quadrilaterals.
- Ø must be ranged between -80 to +80.
- θ must be ranged between -180 and 180.

# Drawing a Sphere

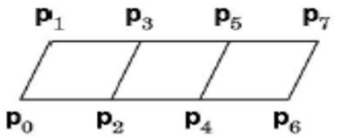
We must convert degrees to radians for the standard trigonometric functions,

the ***code for the quadrilaterals*** corresponding to ***increments of 20 degrees in θ*** and

to ***20 degrees in φ***

# Drawing a Sphere

```
//   Degrees must be converted to radians for the standard trigonometric functions.
c=M_PI/180.0; // degrees to radians, M_PI=3.14159…
// To get Longitudes
for(phi=-80.0; phi<=80.0; phi+=20.0) {
        //   phir=c*phi;  //   phi in radians – for the 1stpoint
       //  phir20=c*(phi+20);  //for the 2ndpoint
    glBegin(GL_QUAD_STRIP);
    // To obtain Latitudes
    for(theta=-180.0; theta<=180.0; theta+=20.0) {
          x=sin(c*theta)*cos(c*phi);
          y=cos(c*theta)*cos(c*phi);
          z=sin(c*phi);
          glVertex3d(x, y, z);                          // 1st  point
          x=sin(c*theta)*cos(c*(phi+20.0));
          y=cos(c*theta)*sin(c*(phi+20.0));
          z=sin(c*(phi+20.0));
          glVertex3d(x, y, z);                           // 2nd point
    }     glEnd();
}
```



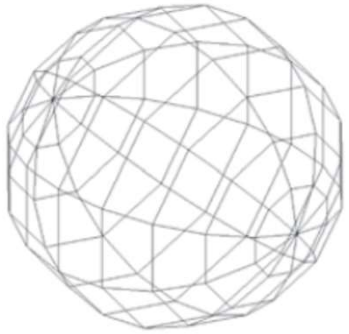



FIGURE 2.15 Sphere approximation with quadrilaterals.

**Drawing the portion of lower latitudes**

# Drawing a Sphere

```
x=y=0;
z=1;
glBegin(GL_TRIANGLE_FAN);
glVertex3d(x, y, z);
c=M_PI/180.0;
z=sin(c*80.0);
for(theta=-180.0; theta<=180.0; theta+=20.0) {
   x=sin(c*theta)*cos(c*80.0);
   y=cos(c*theta)*sin(c*80.0);
   glVertex3d(x, y, z);
}
glEnd();
```

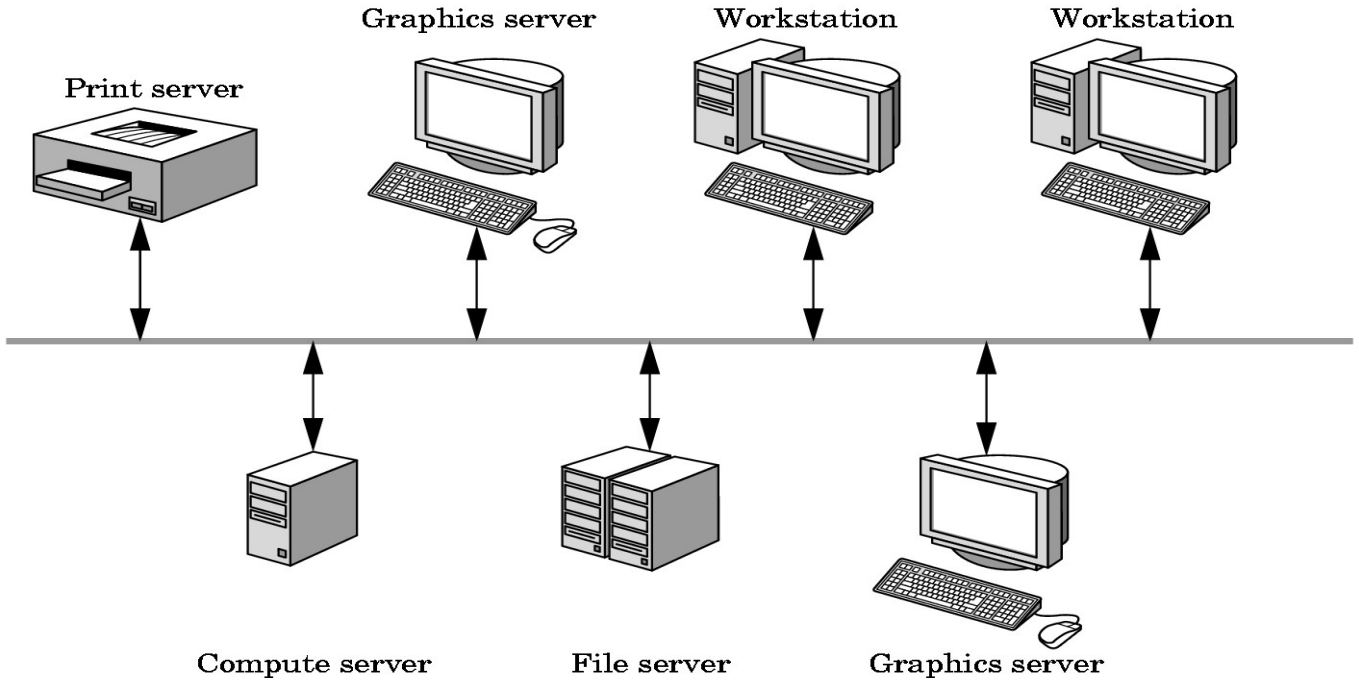
Drawing the portion around the north pole

# Drawing a Sphere

```
x=y=0;
z=-1;
glBegin(GL_TRIANGLE_FAN);
glVertex3d(x, y, z);
z=-sin(c*80.0);
for(theta=-180.0; theta<=180.0; theta+=20.0) {
   x=sin(c*theta)*cos(c*80.0);
   y=cos(c*theta)*sin(c*80.0);
   glVertex3d(x, y, z);
}
glEnd();
```

Drawing the portion around the south pole

# Clients-Servers

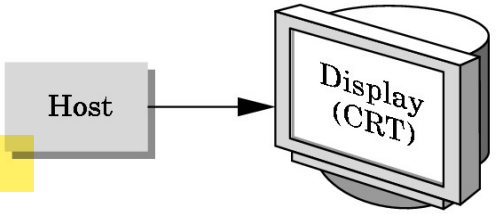



- Our programs worked on single & isolated system so far, but it should also work in distributed computing and networks
- Distributed graphics, projection walls etc.

what is this in graphics

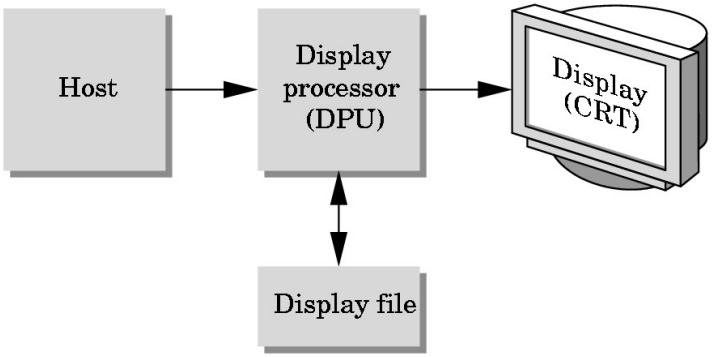
# Display Lists

- Instructions are stored in a *display memory – display file – **display list***
- **Modes:**
  - ***immediate*** – each element is processed and displayed
  - ***retained*** – objects are defined & stored in a display list on a server & redisplayed on the client request




write this diagram

**Early graphics systems - 1960**




**Display processor architecture**

# Definition & Execution of Display Lists

```
#define  BOX  1
glNewList(Box, GL_COMPILE); /* sends to the server*/
        glBegin(GL_POLYGON);
         glColor3f(1.0, 1.0, 1.0);
         glVertex2f(-1.0, -1.0);
         glVertex2f( 1.0, -1.0);
         glVertex2f( 1.0,  1.0);
         glVertex2f(-1.0,  1.0);
        glEnd( );
glEndList ( );
/* GL_COMPILE_AND_EXECUTE – immediate display */
```

***Drawing – execution***

glCallList(BOX);

# Display Lists & Transformations

If model-view or projection matrices changed between execution of the display list – the drawn model will appear at different positions

```
glMatrixModel(GL_PROJECTION);
for (i=1; i<5; i++);
{
    glLoadIdentity( );
    gluOrtho2D(-2.0*i, 2.0*i, -2.0*i, 2.0*i );
    glCallList(BOX);
}
```

# Display Lists & Push/Pop

! Color is changed whenever the list is executed

Execution of the display list changes the state and attributes in general
– may cause unexpected effects

it is possible and recommended to store them in the stack

at the beginning of the display list specification

```
glPushAttrib(GL_ALL_ATTRIB_BITS);
glPushMatrix( );
```

at the end of the display list specification

```
glPopAttrib( );
glPopMatrix ( );
```

# *Application Programmer's Interface*

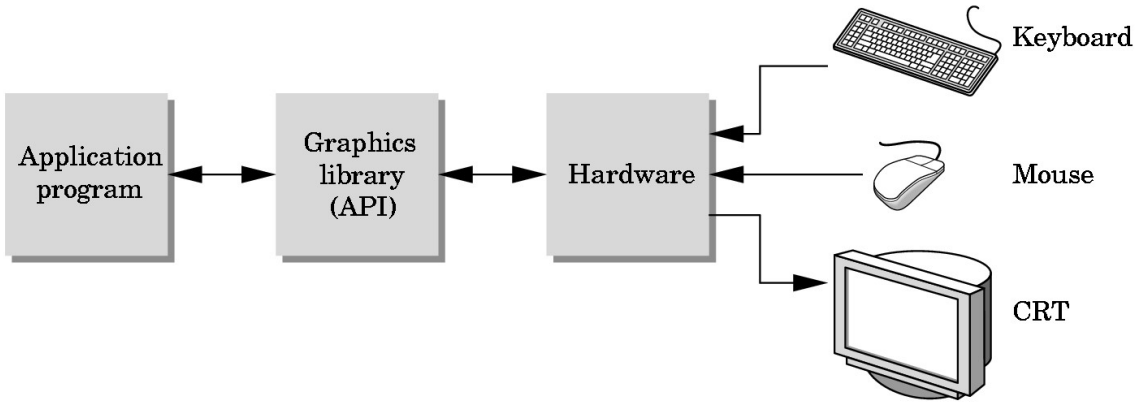



Fig shows the application programmer's model of the system.

API functionality should match the conceptual model

Synthetic Camera Model used for APIs like OpenGL, PHIGS, Direct 3D, Java3D, VRML etc.

Functionality needed in the API to specify:

- Objects
- Viewers
- Light sources
- Material properties

# *Application Programmer's Interface*

- Objects are defined by points or vertices, line segments, polygons etc. to represent complex objects
- API primitives are displayed rapidly on the hardware
- usual API primitives:
  - points
  - line segments
  - polygons
  - text

to summarize - API primitives are used to represent complex objects

10 primitives

but adrallellu text irlilvalla

# *Application Programmer's Interface*

OpenGL defines primitives through list of vertices – triangular polygon is drawn by:

```
glBegin(GL_POLYGON);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.0, 1.0);
 glEnd( );
```

3dimension - see attributes

ok , triangle code ishte

attribute GL_POLYGON actually defines the primitive to be drawn – others

- GL_LINE_STRIP - draws a strip – n+1 points define n triangles
- GL_POINTS – draws only points

# *Application Programmer's Interface*

Some APIs :

work with frame buffer – read/write pixel level

- provides curves & surfaces / approximated by a series of simpler primitives
- OpenGL provides access to frame buffer, curves and surfaces

# *Application Programmer's Interface*

Camera specification in APIs:

*position* – usually center of lens (COP)

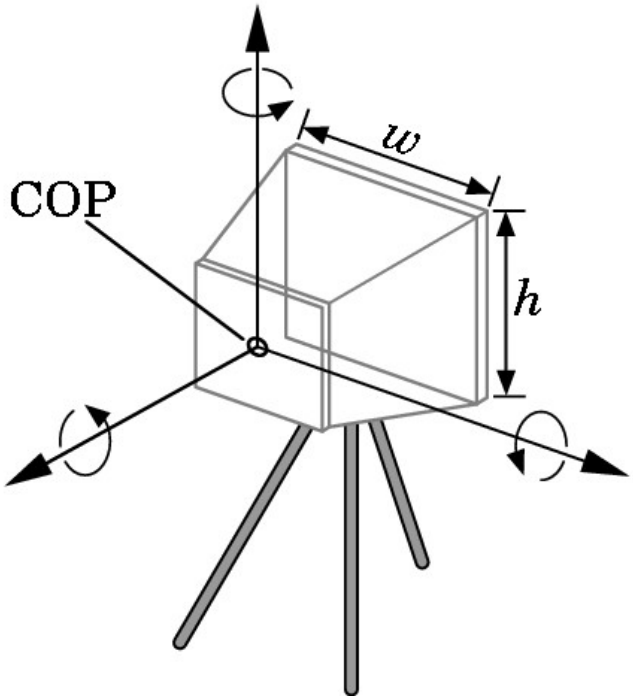
**orientation** – camera coordinate system with its origin at center of lens (COP) . camera can rotated around those three axes

*focal length* of lens determines the size of the image on the film
actually viewing angle

*film plane* – the back of the camera has a height and a width

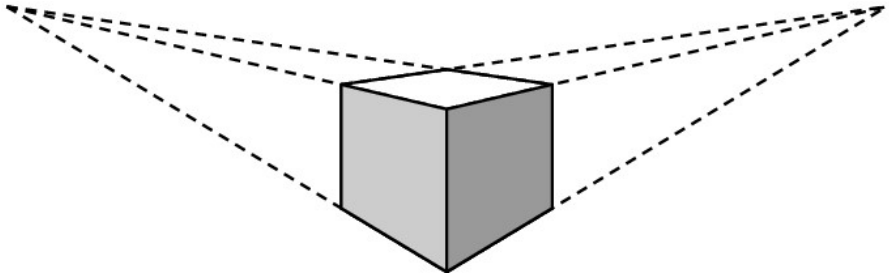
In some APIs, the orientation of the back of the camera can be adjusted independently of the orientation of the lens.

what is all this



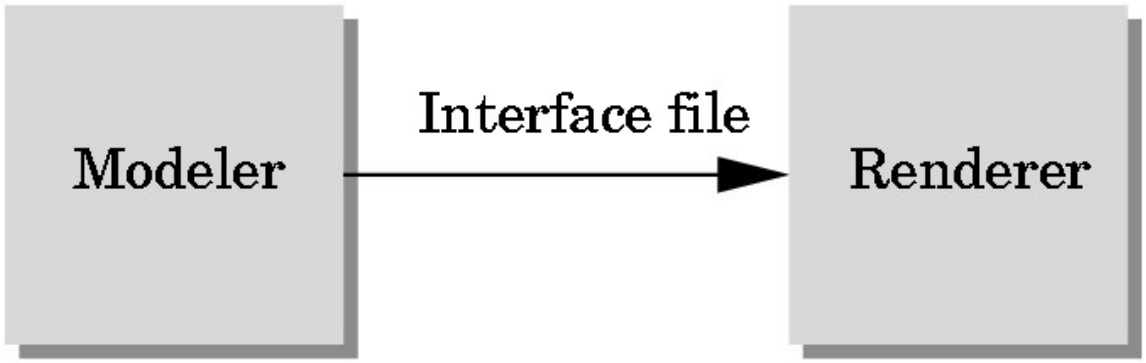
# *Application Programmer's Interface*

What is this



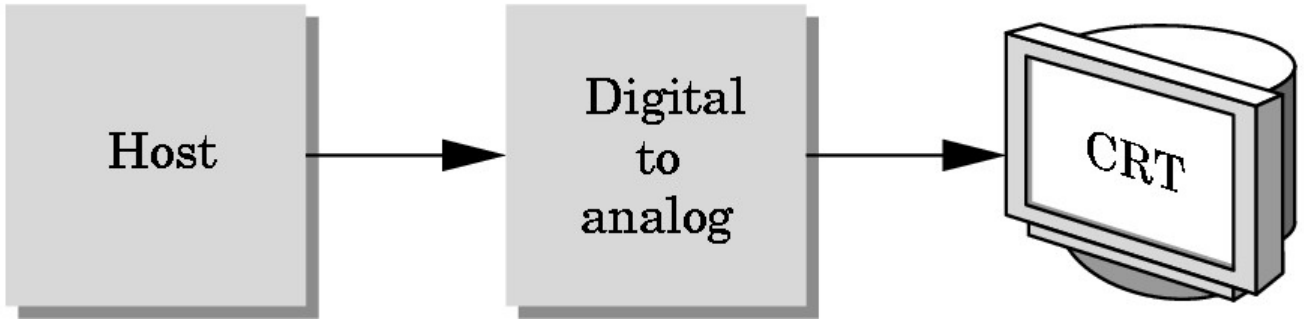
- Two coordinate systems are used:
  - world coordinates, where the object is defined
  - camera coordinates, where the image is to be produced
- Transformation for conversion between coordinate systems or
  - gluLookAt(cam_x, cam_y,cam_z, look_at_x, look_at_y, look_at_z,…)
  - glPerspective( field_of_view)
- Lights sources – defined by location, strength, color, directionality. APIs provide a set of functions to specify these parameters
- Material properties - are ***attributes*** of objects and are specified through a series of function calls at the time of defining the objects.
- Observed visual properties of objects are given by material and light properties

# *Modeling – Rendering Paradigm*



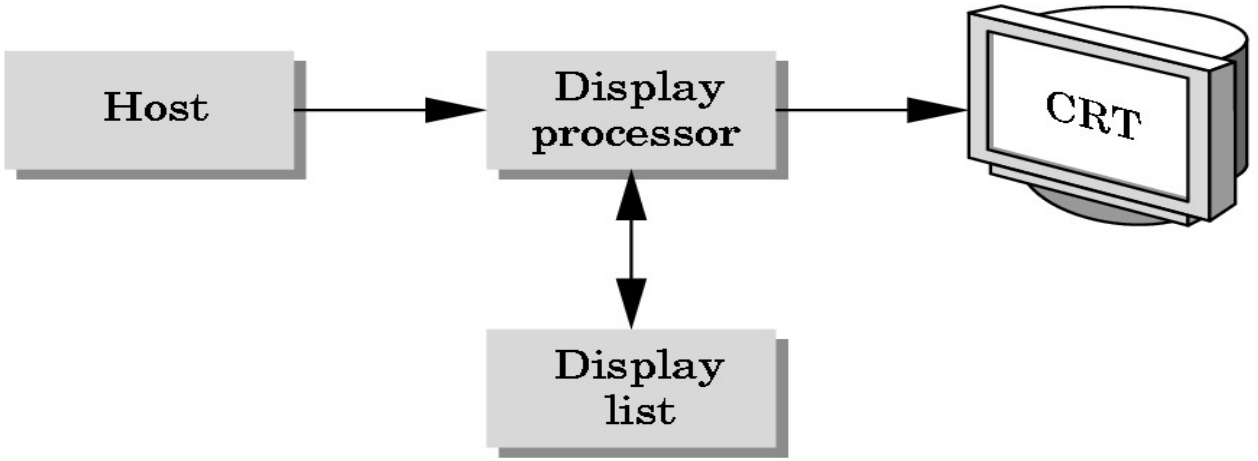

- In many applications the modeling is separated from production of an image – ***rendering*** (CAD systems, animations etc.)
- In this case the modeling SW/HW might be different from the renderer
- the connection between both parts can be simple or highly complex using distributed environments
- Popular method for generating computer games and images over Internet
- Models ( including the geometric objects, lights, cameras and material properties) , are placed in a data structure called a ***scene graph*** that is passed to a renderer or a game engine.

# *Graphics Architectures*



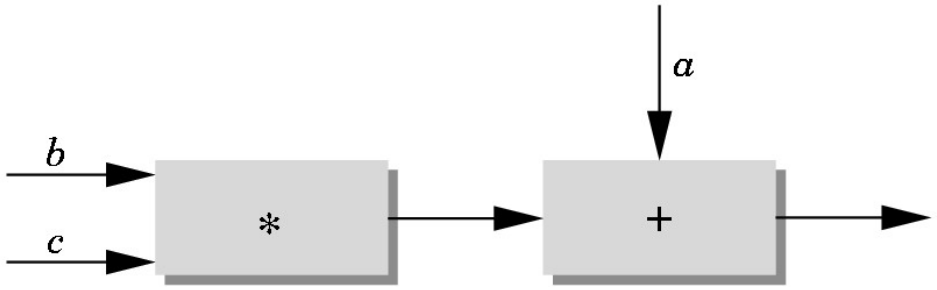

- Early graphics systems – CRT had just basic capability to generate line segments connecting two points
- vector based with refreshing – length of line segments limited light pen often used for manipulation
- systems with memory CRT – the whole picture redrawn if changed

# *Graphics Architectures*




- Display processors
  - standard architecture with capabilities to display primitives
  - composition made at the host
- memory – display list – contains primitives to be displayed.

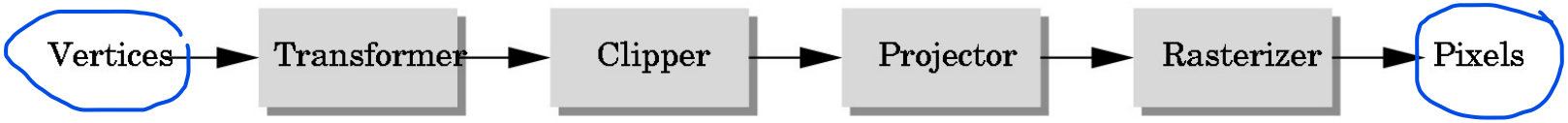
# *Pipeline Architectures*




VLSI circuits enabled major advances in graphics architectures

- simple arithmetic pipeline $a + b * c$
- when addition of *(b * c)* and *a* is performing new *b * c* is computed in parallel – pipelining enabled significant speed up
- similar approach can be used for processing of geometric primitives as well

# *Pipeline Architectures*




There are 4 major steps in the geometric pipeline:

- transformations – like scaling, rotations, translation, mirroring, sheering etc.
- clipping – removal of those parts that are out of the viewing field
- projection
- rasterization

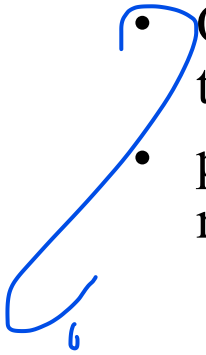
homogeneous coordinates and matrix operations geometric transformations are used

all these processes yake antane gottilla nange - imagine madake bartilla

# *Clipping, Projection & Rasterization*

i have downloaded one pdf - see that

- Clipping is used to remove those parts of the world that cannot be seen.
- Objects representation is “kept” in 3D as long as possible. After transformation and clipping must be projected to 2D somehow
- projected objects or their parts must be displayed – and therefore rasterized.

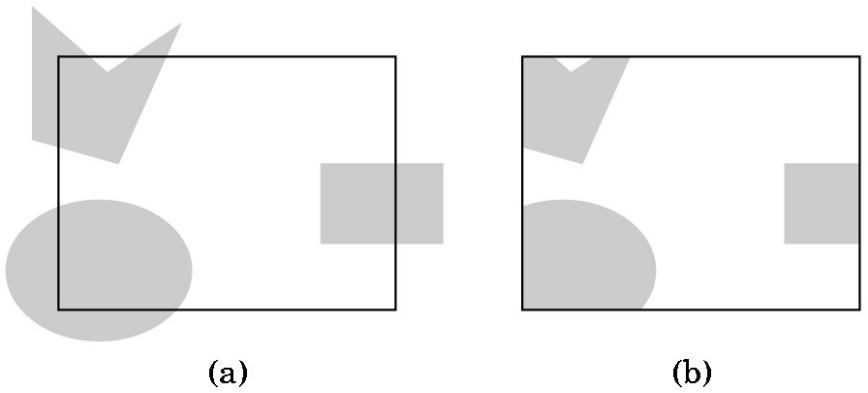


- All those steps are performed on your graphics cards in hardware nowadays.

# Viewing

- In the synthetic camera concept the object specification is completely independent from
  - the camera specification
  - lights properties and positions used
  - material properties as well
- We used some default settings for those expecting that objects are placed “in the right position”, sufficiently far from the camera etc.
- We will concentrate on
  - 2D viewing,
  - orthographic projection,
  - matrix mode application

# 2D Viewing



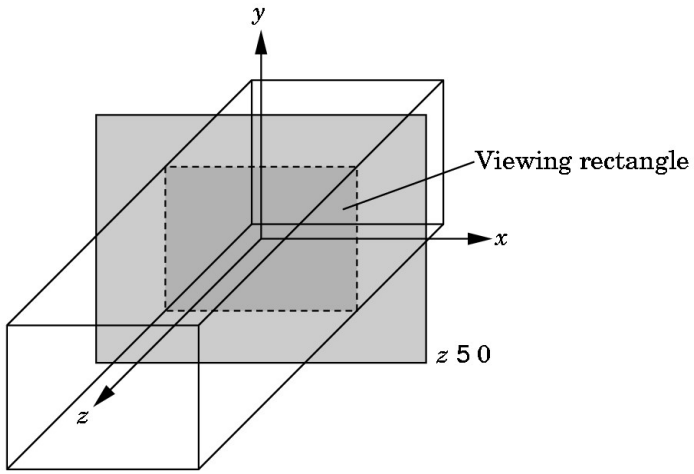

**Clipping rectangle or viewing rectangle**

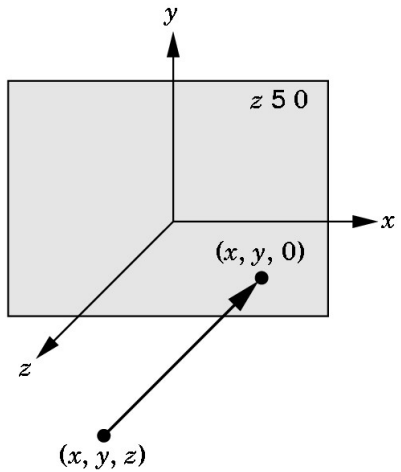
- Area of the world that we image – is the viewing or clipping rectangle.
- Objects before and after 2D clipping

Objects

- outside are clipped out and are not displayed
- partially inside – only parts inside are displayed

# 2D Viewing – Orthographic View



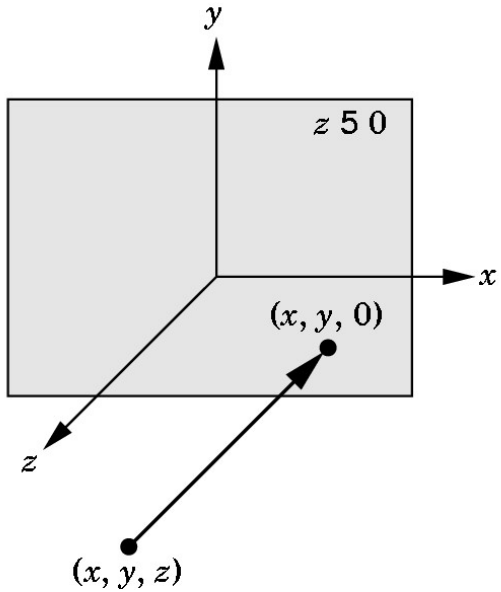



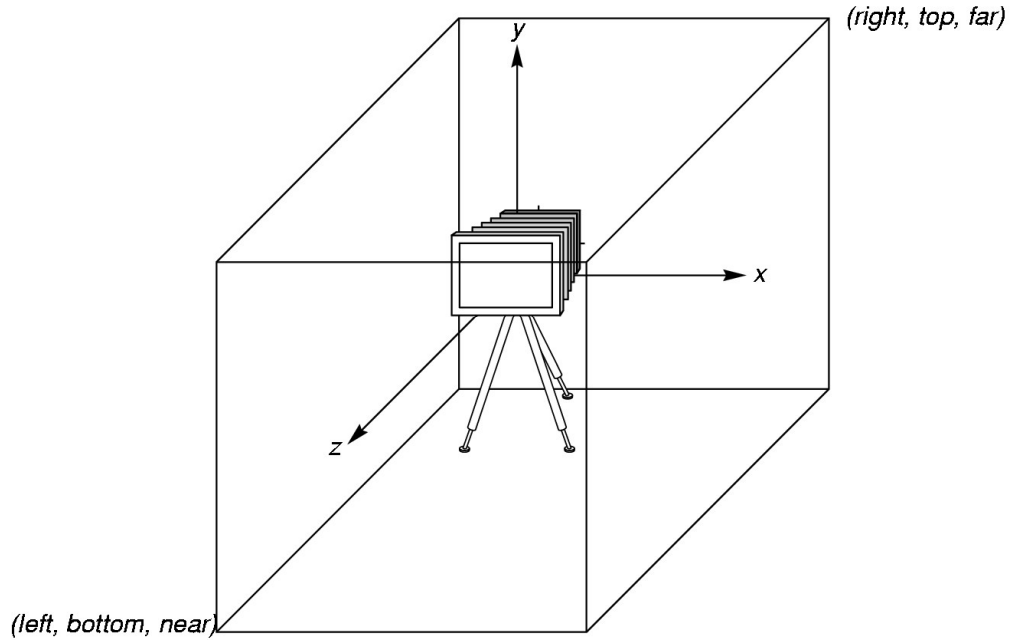

- Viewing rectangle is at $z = 0$ with corners

  Bottom-left(-1.0,-1.0) & Upper-right(1.0, 1.0)

- Implicit viewing volume in OpenGL is 2 x 2 x 2 cube – center in the origin

# 2D Viewing – Orthographic View




void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top) /* *near=-1, far=1*/

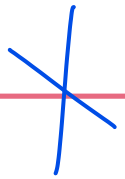
# 2D Viewing - Matrix Mode

- Pipeline of the graphics system utilizes matrices multiplications for geometric transformations.
- Two the most important matrices:

  Model-view: defines COP and orientation

  Projection: defines the projection matrix
- Setting matrix mode ( it is the status definition of the system):

```
glMatrixMode(GL_PROJECTION); /* change to PROJECTION status */
glLoadIdentity( ); /* sets the matrix to the IDENTITY matrix */
gluOrtho2D(0.0, 500.0, 0.0, 500.0); /* sets window to 500 x 500 image */
glMatrixMode(GL_MODELVIEW);
/* return back to the MODEL-VIEW state – highly recommended */
```



# Control Functions

- Minimal interaction between the program and the particular OS must be used (X-Windows - UNIX, Windows)
- GLUT library provides minimal & simple interface between API and particular OS (full functionality for interaction between a program and OS is not a part of this course)
- Programs using GLUT interface should run under multiple window system.

# Interaction with the Window System

- Window or screen window – rectangular area of a display
- Window system – refers to X-Windows or MS Windows
- The origin – not always lower-left corner – OpenGL style (some have orientation: top to bottom, left to right as TV sets or position information returned from input devices like mouse)

- glutInit(int *argcp, char *argv) – enables to pass command-line arguments as in the standard C *main* function
- glutCreateWindow(char *title) – creates a window with the name specified in the string title with default values (size, position etc.)

# Interaction with the Window System

- BEFORE creating the window GLUT functions can be used to specify properties:

```
glutDisplayMode(GLUT_RGB|GLUT_DEPTH|GLUT_DOUBLE);
/* parameters or-ed and stored in the argument to glutInitDisplayMode */
glutWindowSize(480, 640);
glutWindowPosition(0 , 0);
```

specifies the windows 480 x 640 in the top-left corner of the display

GLUT_RGB x GLUT_INDEX – type of color system to be used

GLUT_DEPTH – a depth buffer for hidden-line removal

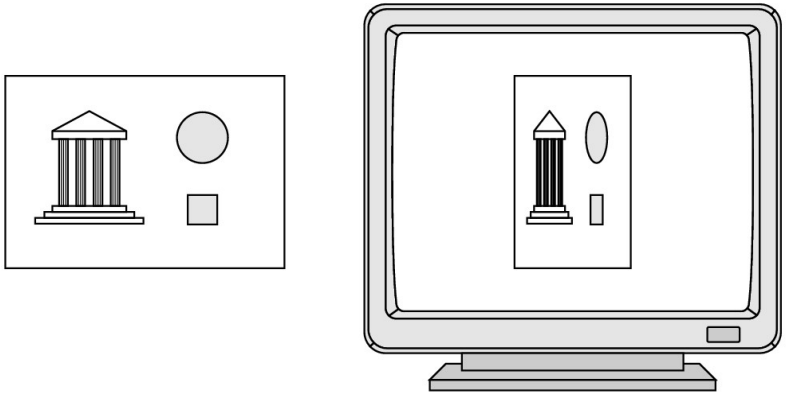
GLUT_DOUBLE x GLUT_SINGLE – double x single buffering

- Implicit options: RGB color, no hidden-line removal, single buffer

# Aspect Ratio and Viewports

which rectangle

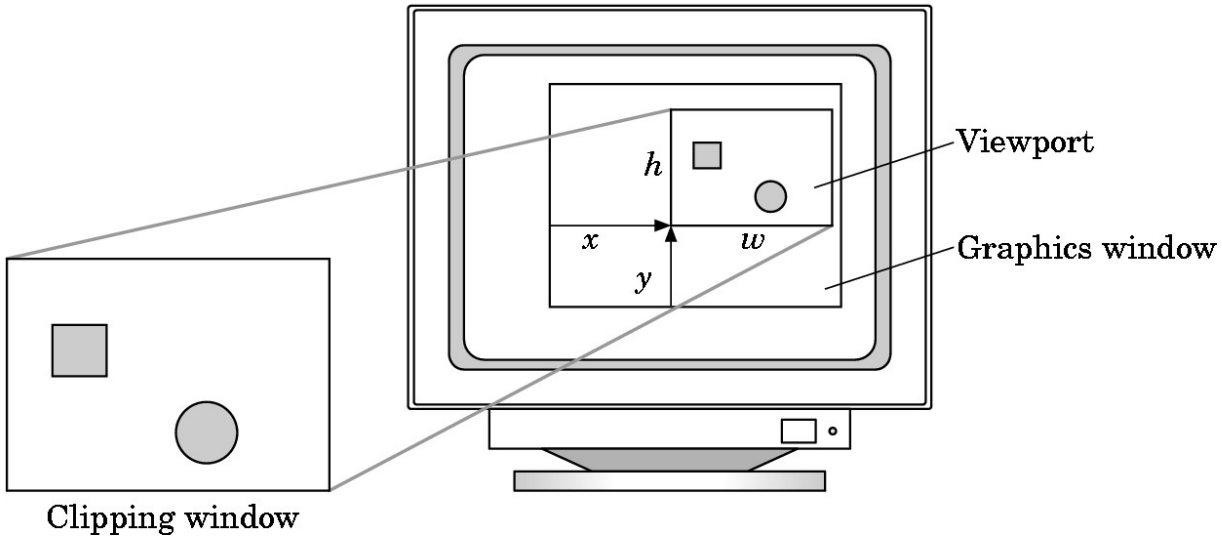
- Aspect ratio – the ratio of the rectangle's width and height
- If different in glOrtho and glutInitWindowSize – undesirable side effects
- Caused by the independence of object, viewing parameters and workstation window specifications
- Concept of a VIEWPORT



(a) (b)

# Aspect Ratio and Viewports

void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)




The viewport is part of the state – when changed between rendering objects or redisplay – different window-viewport transformations used to make the scene.

# Graphics Architectures

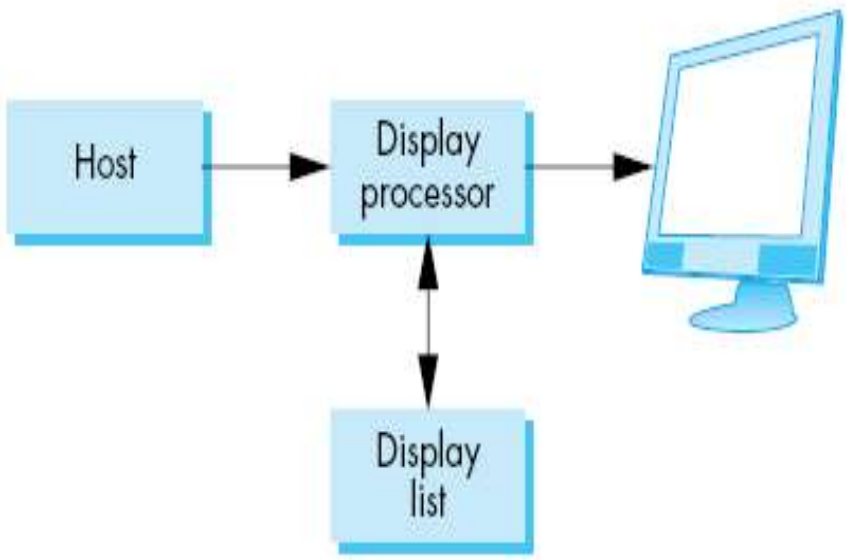



FIGURE 1.36 Display-processor architecture.

# Pipeline architecture

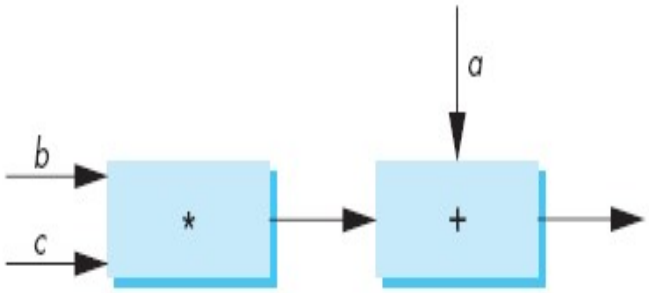



FIGURE 1.37 Arithmetic pipeline.

mele explain madiddu enu matte?

i'll leave this for now - later one nig pipe line pdf idhe alva , from there i'll study

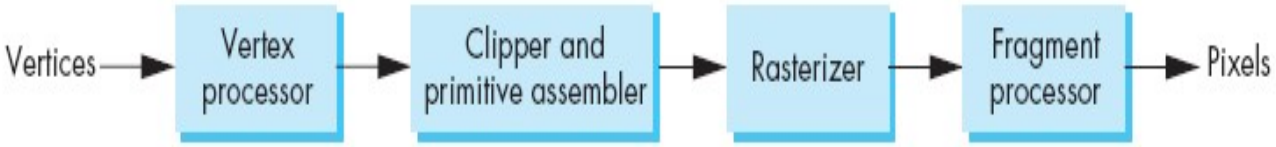
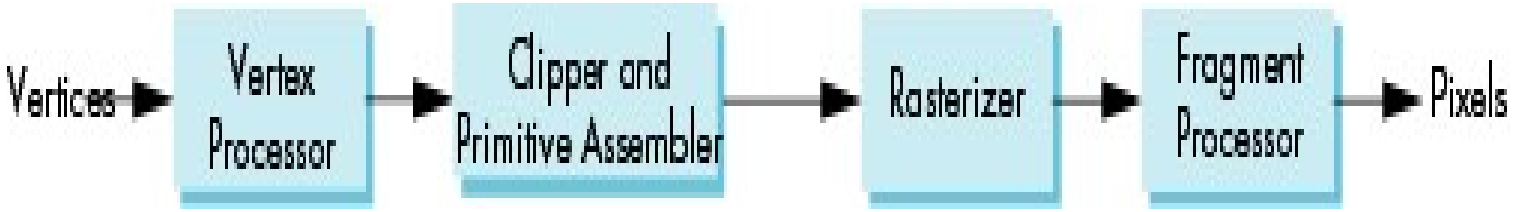



FIGURE 1.38 Geometric pipeline.

# Graphics Pipeline Architecture

- Process objects one at a time in the order they are generated by the application
  - Can consider only local lighting
- Pipeline architecture




application program

display

- All steps can be implemented in hardware on the graphics card

# Vertex Processing

- Much of the work in the pipeline is in converting object representations from one coordinate system to another
  - Object coordinates
  - Camera (eye) coordinates
  - Screen coordinates
- Every change of coordinates is equivalent to a matrix transformation
- Vertex processor also computes vertex colors

# Primitive Assembly

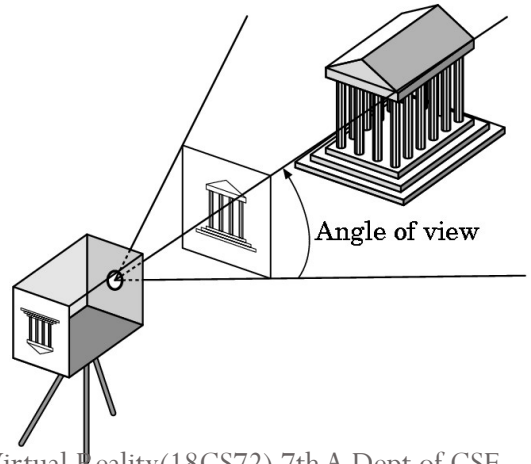
Vertices must be collected into geometric objects before clipping and rasterization can take place

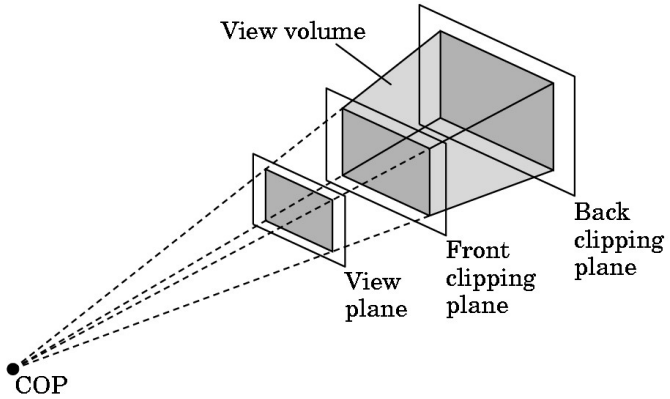
- Line segments
- Polygons
- Curves and surfaces

# Clipping

Just as a real camera cannot "see" the whole world, the virtual camera can only see part of the world or object space

- Objects that are not within this volume are said to be *clipped* out of the scene



# Rasterization

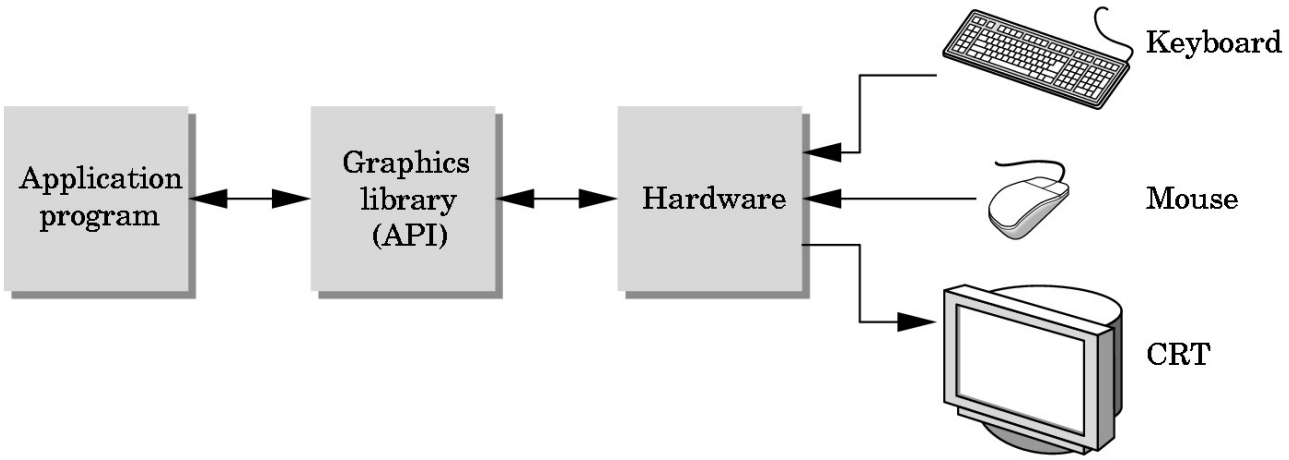
- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- Fragments are "potential pixels"
  - Have a location in frame bufffer
  - Color and depth attributes
- Vertex attributes are interpolated over objects by the rasterizer

# Fragment Processing

- Fragments are processed to determine the color of the corresponding pixel in the frame buffer
- Colors can be determined by texture mapping or interpolation of vertex colors
- Fragments may be blocked by other fragments closer to the camera
  - Hidden-surface removal

# The Programmer's Interface

- Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)



# Video Display Devices

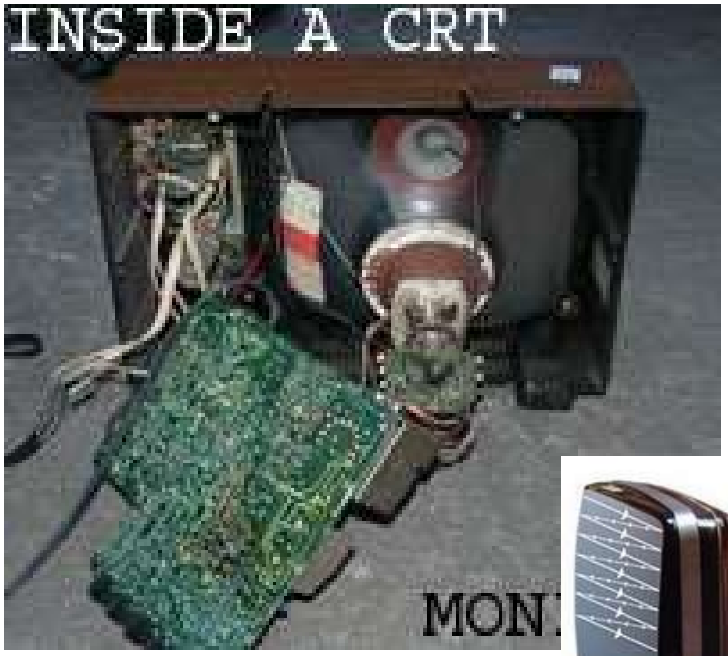
- Cathode-ray tube (CRT) Monitor
- Rater-Scan Displays
- Random-Scan Displays
- Color CRT Monitors
- Flat-Panel Displays

## Cathode-ray tube (CRT) Monitors

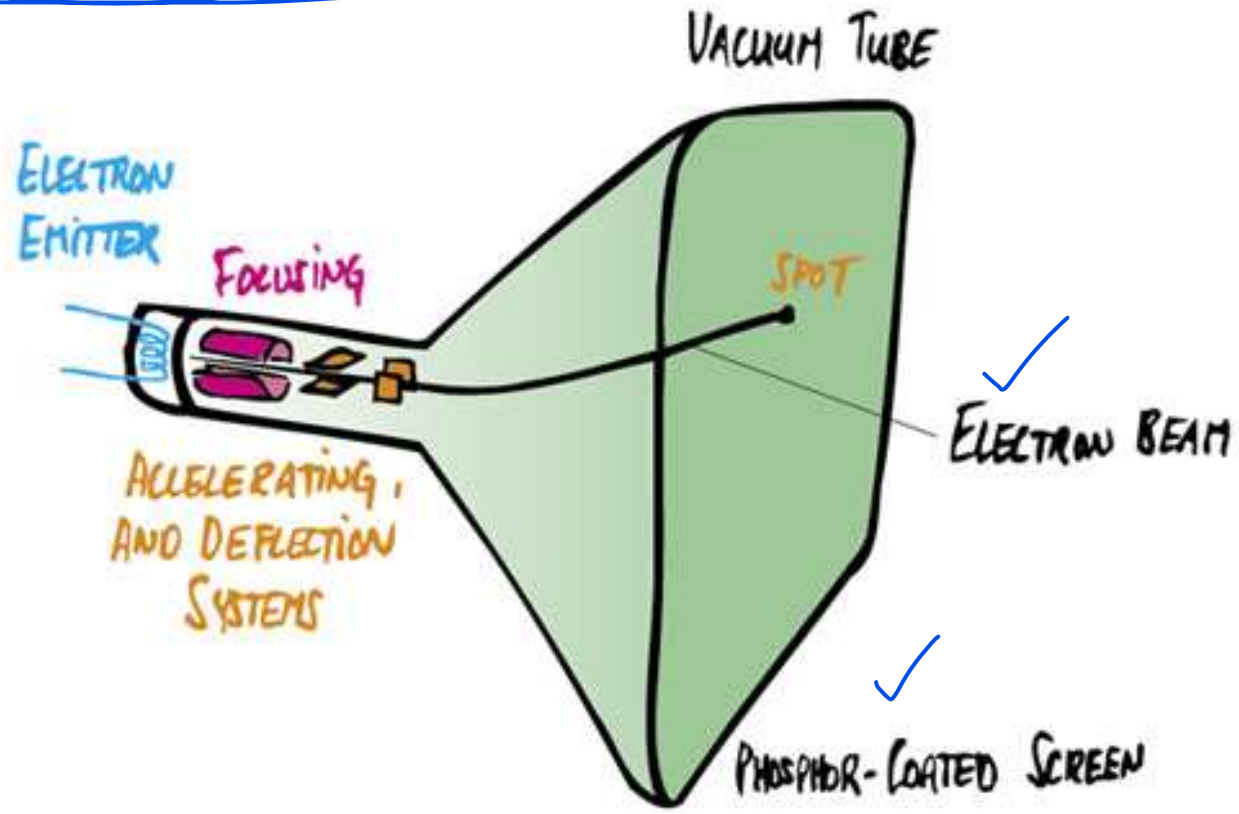
mele one image practice mabekittalva CRT - that one here as well

- **Primary output device – Video monitors**
  - **Standard design of video monitor: Cathode-ray tube (CRT)**

## Cathode-ray tube (CRT) Monitors




# Display Technologies



## Cathode-ray tube (CRT) Monitors

done - above '

working of CRT

- **Refresh CRT**
  - Beam of electrons hit phosphor-coated screen, light emitted by phosphor
  - Direct electron beam to the same screen repeatedly, keeping phosphor activated
  - The frequency at which a picture is redrawn on the screen is referred to as the "refresh rate"

nice and easy definition

  - The maximum number of points that can be displayed on a CRT is referred to as the "resolution"
  - Display principle
    - Raster Scan Display Principle
    - Random Scan Display Principle

types kelidre - write these two

# CRT Display Principles

- **Raster-Scan Displays**
  - **Based on TV technology**
    - **Electron beam swept across screen one row at a time from top to bottom**
    - **Each row is referred to as a scan line**

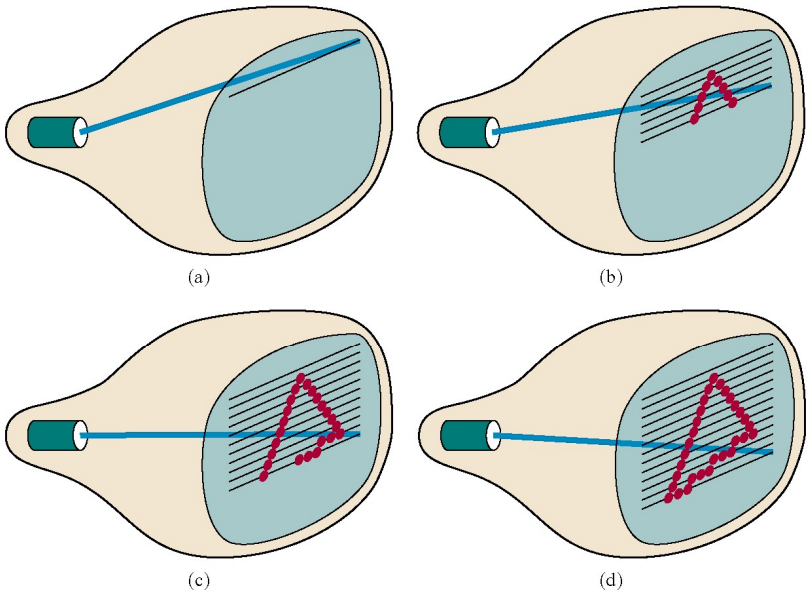
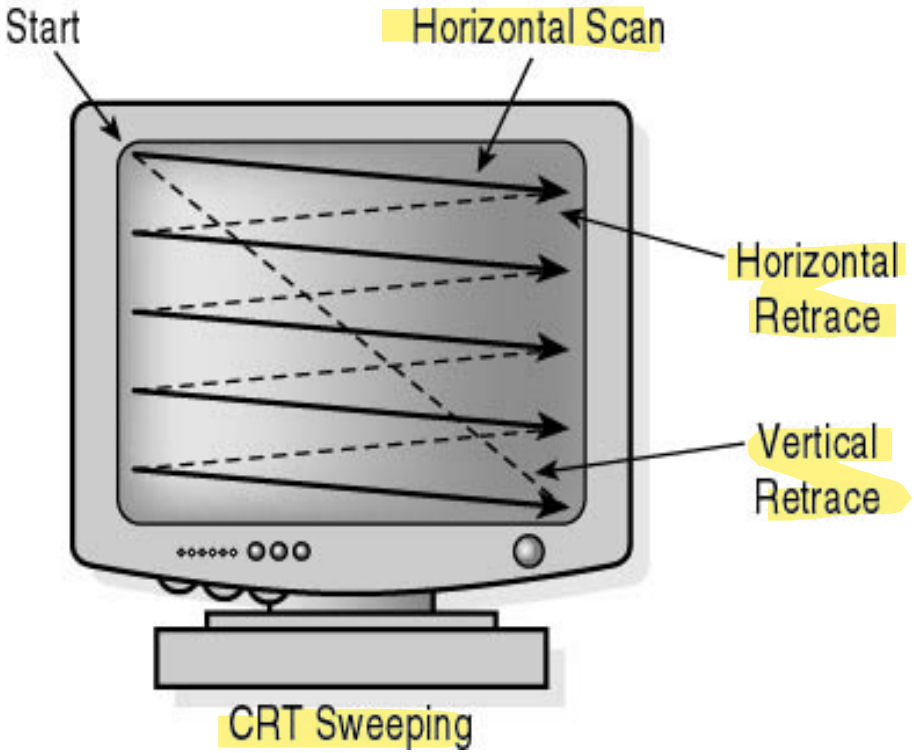



Figure 2-7
A raster-scan system displays an object as a set of discrete points across each scan line.

# Refresh Rates

- Frames per second (FPS)

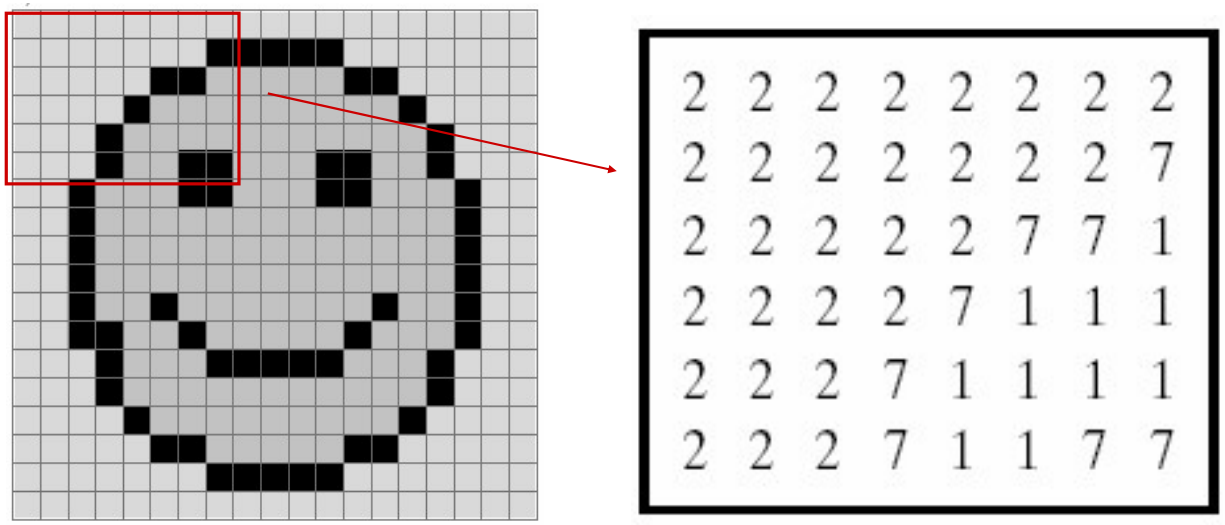
what frames




write this diagram as well

# CRT Display Principles

- **Raster-Scan Displays**
  - **Picture elements: screen point referred as "Pixel"**
  - **Picture information stored in refresh (frame) buffer**



everything related to random and raster - done above

# CRT Display Principles

- **Raster-Scan Displays**
  - **Picture information stored in refresh (frame) buffer**
    - **The number of bits per pixel in the frame buffer is called depth or bit planes**
    - **Buffer with 1 bit per pixel – Bitmap**
    - **Buffer with multiple bits per pixel – Pixmap**
  - **Interlaced refresh procedure**
    - **Beams sweeps across every other scan line**

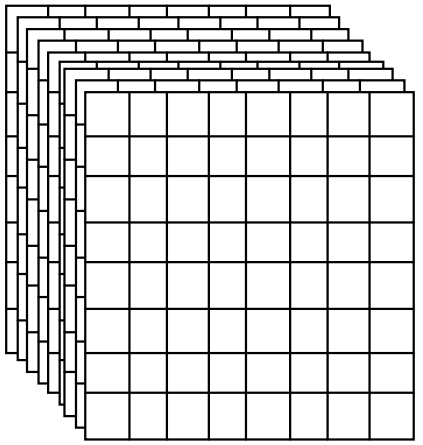
# Interlaced Scanning

- Scan frame 30 times per second
- To reduce flicker, divide frame into two fields—one consisting of the even scan lines and the other of the odd scan lines.
- Even and odd fields are scanned out alternately to produce an interlaced image.

this is done as well - above

# Frame Buffer

- A frame buffer is characterized by size, x, y, and pixel depth.
- the **resolution** of a frame buffer is the number of pixels in the display. e.g. 1024x1024 pixels.
- Bit Planes or Bit Depth is the number of bits corresponding to each pixel. This determines the **color resolution** of the buffer.



**Bilevel** or **monochrome** displays have 1 bit/pixel

8bits/pixel -> 256 simultaneous colors

24bits/pixel -> 16 million simultaneous colors

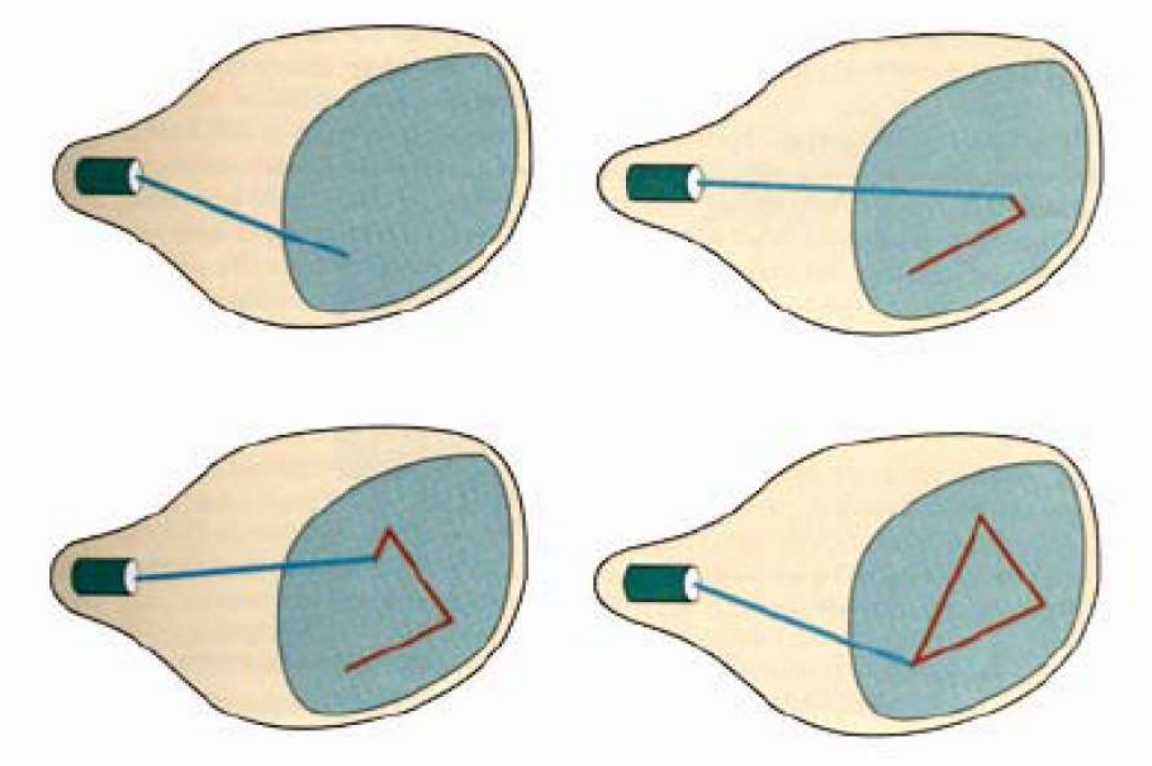
random scan - read again

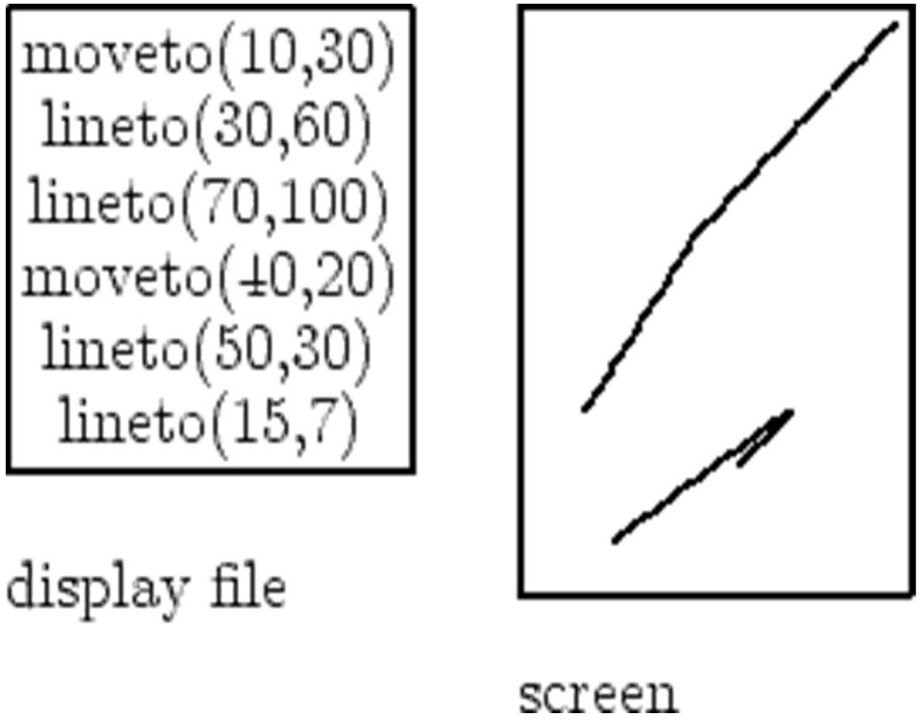
# CRT Display Principles

- Random-Scan Display Principles
  - Calligraphic Displays also called vector, stroke or line drawing graphics
  - Electron beam directed only to the points of picture to be displayed.
  - Vector displays, electron beams trace out lines to generate pictures
  - Picture stores as a set of line-drawing commands
    - Storage referred as display list, refresh display file, vector file or display program

# CRT Display Principles

- Sample of Random-Scan displays principles



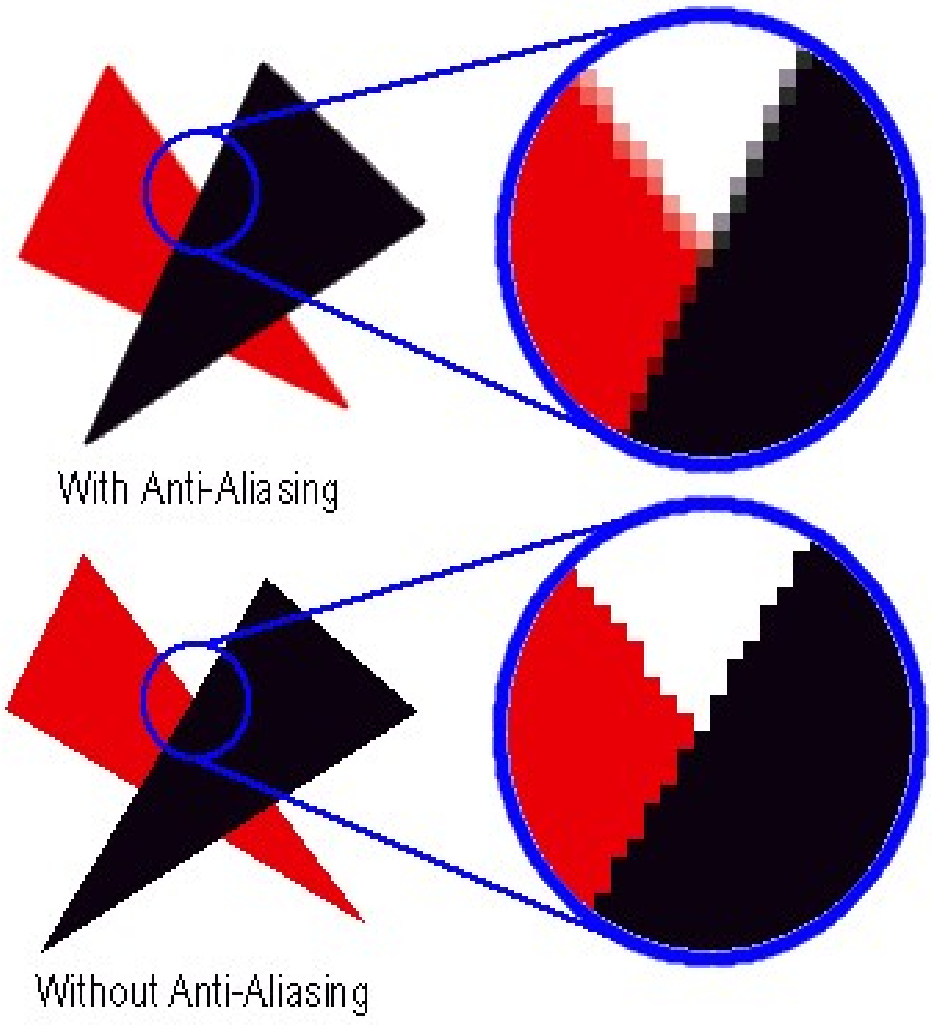



write this as well - for random scan

- Images are described in terms of line segments rather than pixels
- Display processor cycles through the commands

# Pros and Cons

- Advantages to Raster Displays
  - lower cost
  - filled regions/shaded images
- Disadvantages to Raster Displays
  - a discrete representation, continuous primitives must be **scan-converted** (i.e. fill in the appropriate scan lines)
  - **Aliasing** or "**jaggies**" Arises due to sampling error when converting from a continuous to a discrete representation

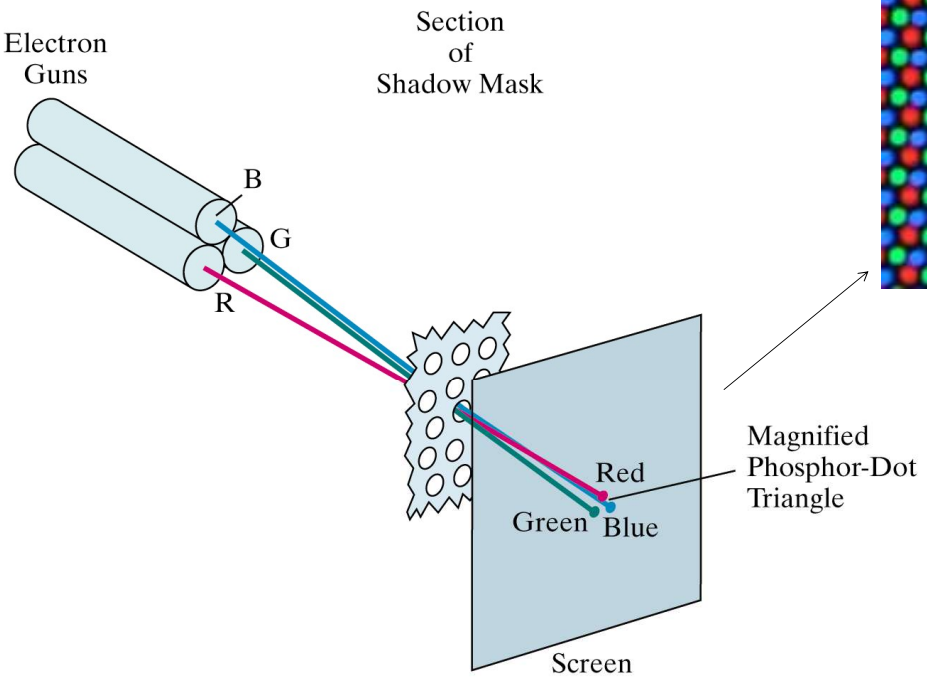

With Anti-Aliasing
Without Anti-Aliasing

# Color CRT Monitors

- **Using a combination of phosphors that emit different-colored light**
- **Beam-penetration**
  - **Used in random-scan monitors**
  - **Use red and green phosphors layers**
  - **Color depends on the penetrated length of electrons**
- **Shadow mask**
  - **Used in raster-scan systems**
  - **Produce wide range of color with RGB color model**

# Color CRT Monitors

- Operation of delta-delta, shadow mask CRT



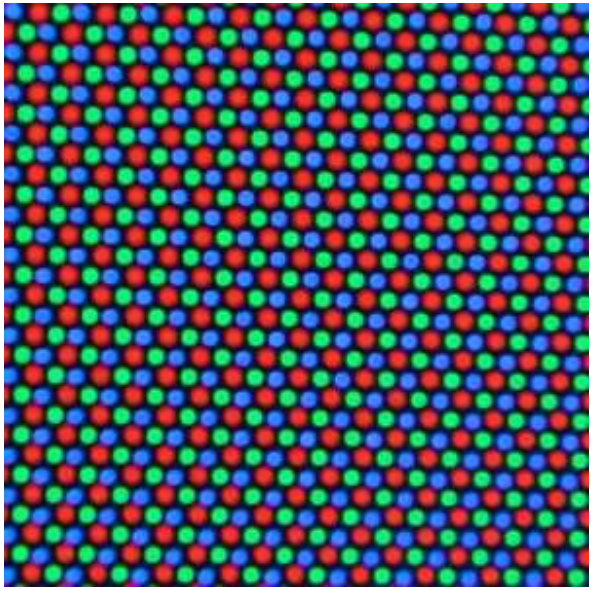



Figure 2-10

# Flat-Panel Displays

- A class of video devices that have reduced volume, weight and power requirement compared with CRT
- Two main categories
  - Emissive Displays
    - Convert electrical energy to light energy
    - e.g. Plasma panels, Light Emitting diodes (LEDs)
  - Non-emissive Displays
    - Use optical effects to convert light from other sources into graphics patterns
    - e.g. LCD monitors

# Plasma Panel Display

- **Plasma panels (gas-discharge display)**
  - **Contracted by filling the region between two glass plates with a mixture of gases**
  - **Refresh buffer used to store picture information**
  - **Firing voltages applied to refresh the pixel positions**

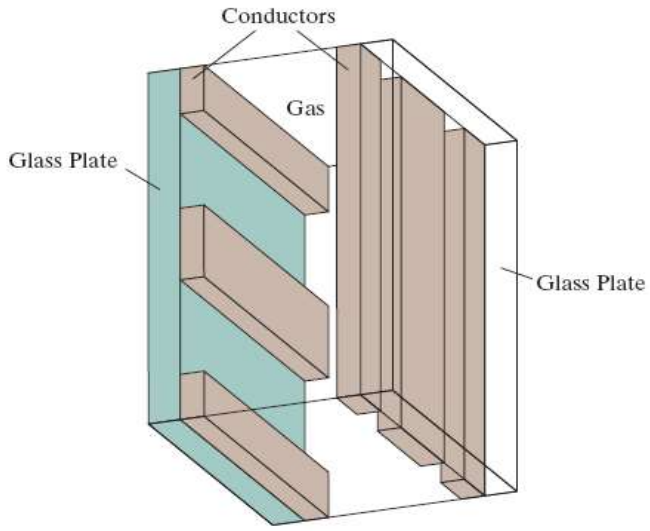


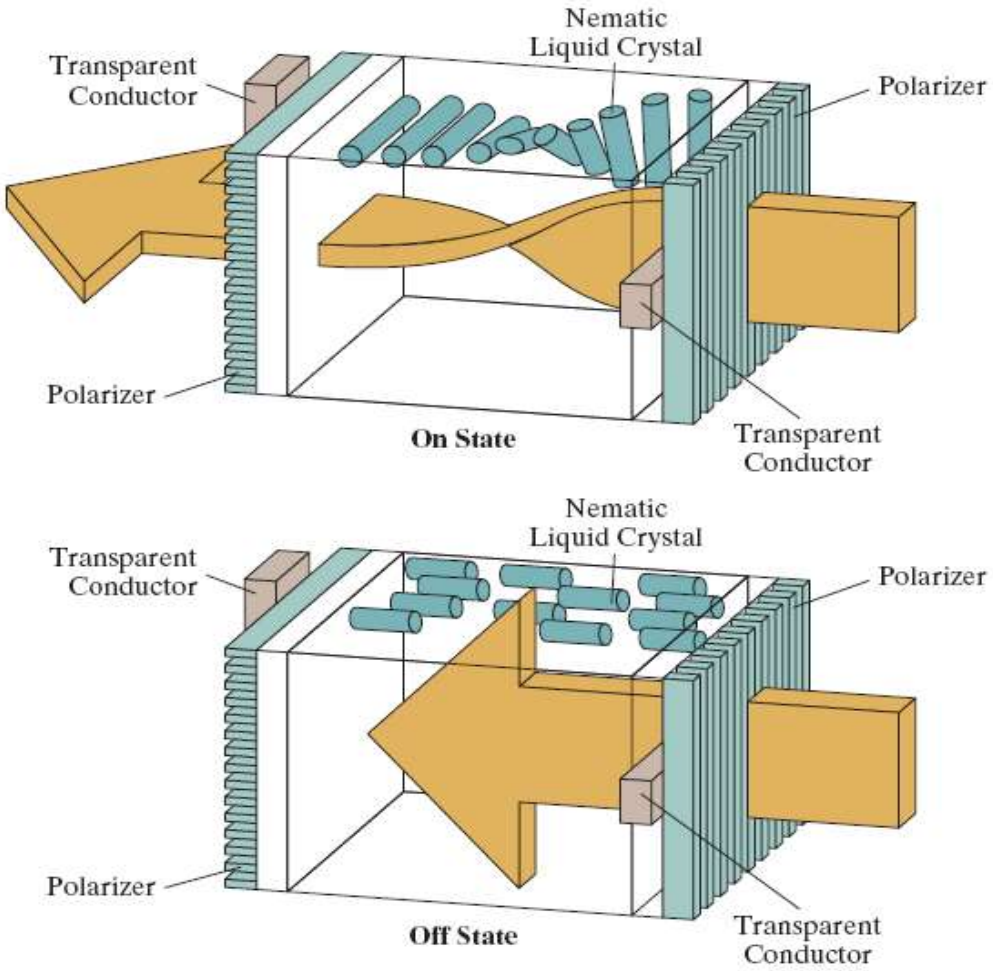

FIGURE 2-11 Basic design of a plasma-panel display device.

# Light-emitting diode (LED)

- LED is a type of emissive device.
- A matrix of diodes is arranged to form the pixel positions in the display, and picture definition is stored in a refresh buffer.
- As in scan-line refreshing of a CRT, information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.

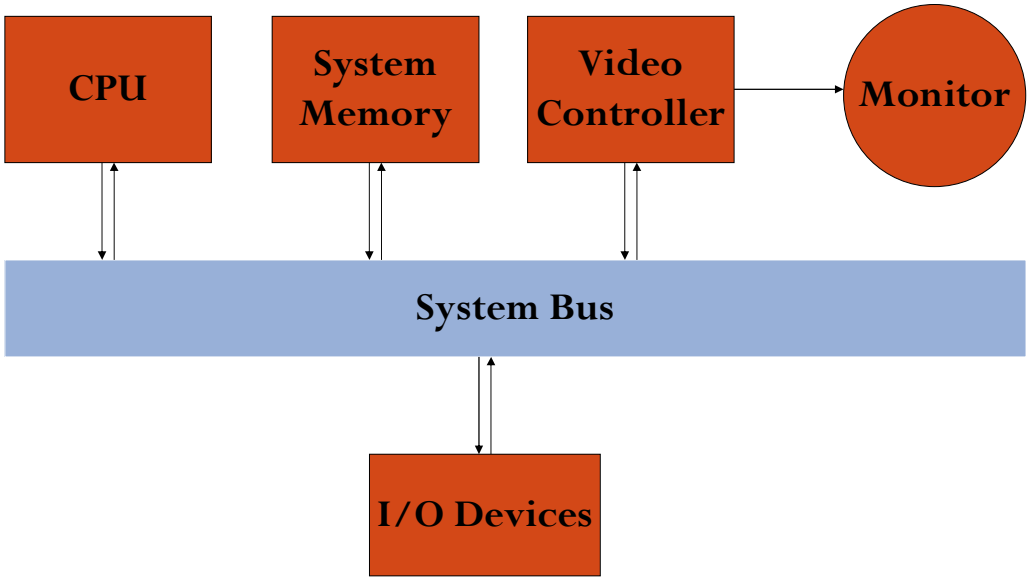
# Liquid-crystal Displays

- **Liquid-crystal displays (LCD) commonly used in small systems**
  - **Liquid crystal, compounds have a crystalline arrangement of molecules, flow like a liquid**
  - **Passive-matrix LCD**
    **To control light twisting, voltage applied to intersecting conductors to align the molecules**
  - **Active-matrix LCD**
    **Using thin-film transistor technology, place a transistor at each pixel location**


Nematic
Liquid Crystal
Transparent
Conductor
Polarizer
Polarizer
On State
Transparent
Conductor
Nematic
Liquid Crystal
Transparent
Conductor
Polarizer
Polarizer
Off State
Transparent
Conductor

# Raster-Scan systems

- **Organization of raster system**
  - **Fixed area of system memory reserved for frame buffer which can be directly accessed by video controller**

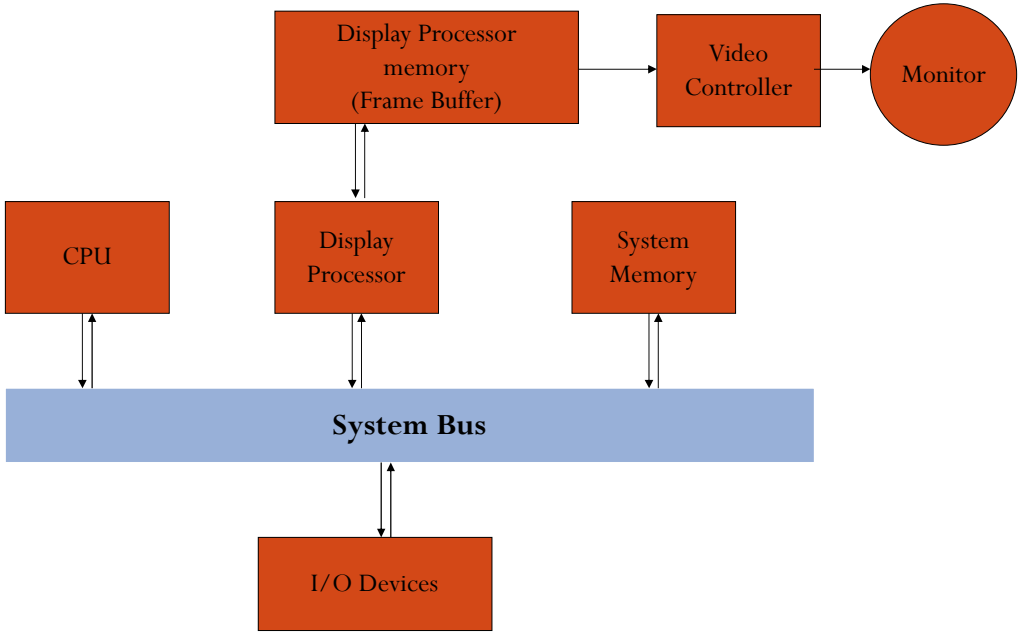


# Raster-Scan systems

- **Video Controller**
  - **Refresh operations**
    - **X, Y register used to indicate pixel position**
    - **Fix Y register and increment X register to generate scan line**
  - **Double buffering**
    - **Pixel value can be loaded in buffer while**
    - **Provide a fast mechanism for real-time animation generation**

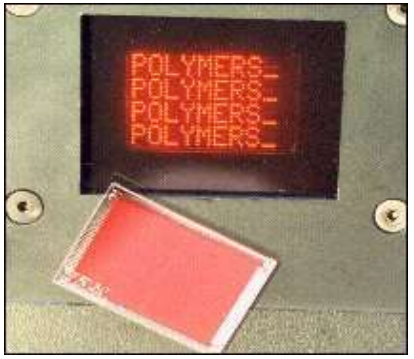
# Raster-Scan Systems

- **Raster-Scan Display Processor**
  - **Free the CPU from the graphics chores**
  - **Provide separate display-processor memory**
  - **Fig. Architecture of raster-scan display system with display processor**



# Video Output Devices

- Desktop
  - Vector display
  - CRT
  - LCD flatpanel
  - Plasma
  - workstation displays(Sun Lab)
  - PC and Mac laptops
  - Tablet computers
  - Wacom's display tablet
  - Digital Micromirror Devices (projectors)

# Program Structure

- Most OpenGL programs have a similar structure that consists of the following functions
  - **`main()`**:
    - defines the callback functions
    - opens one or more windows with the required properties
    - enters event loop (last executable statement)
  - **`init()`**: sets the state variables
    - Viewing
    - Attributes
  - callbacks
    - Display function
    - Input and window functions

what state variables

# simple.c revisited

- In this version, we shall see the same output but we have defined all the relevant state values through function calls using the default values
- In particular, we set
  - Colors
  - Viewing conditions
  - Window properties

86

main.c

```
#include <GL/glut.h>

int main(int argc, char** argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("simple");
  glutDisplayFunc(mydisplay);

  init();

  glutMainLoop();
}
```

includes gl.h

define window properties

display callback

set OpenGL state

enter event loop

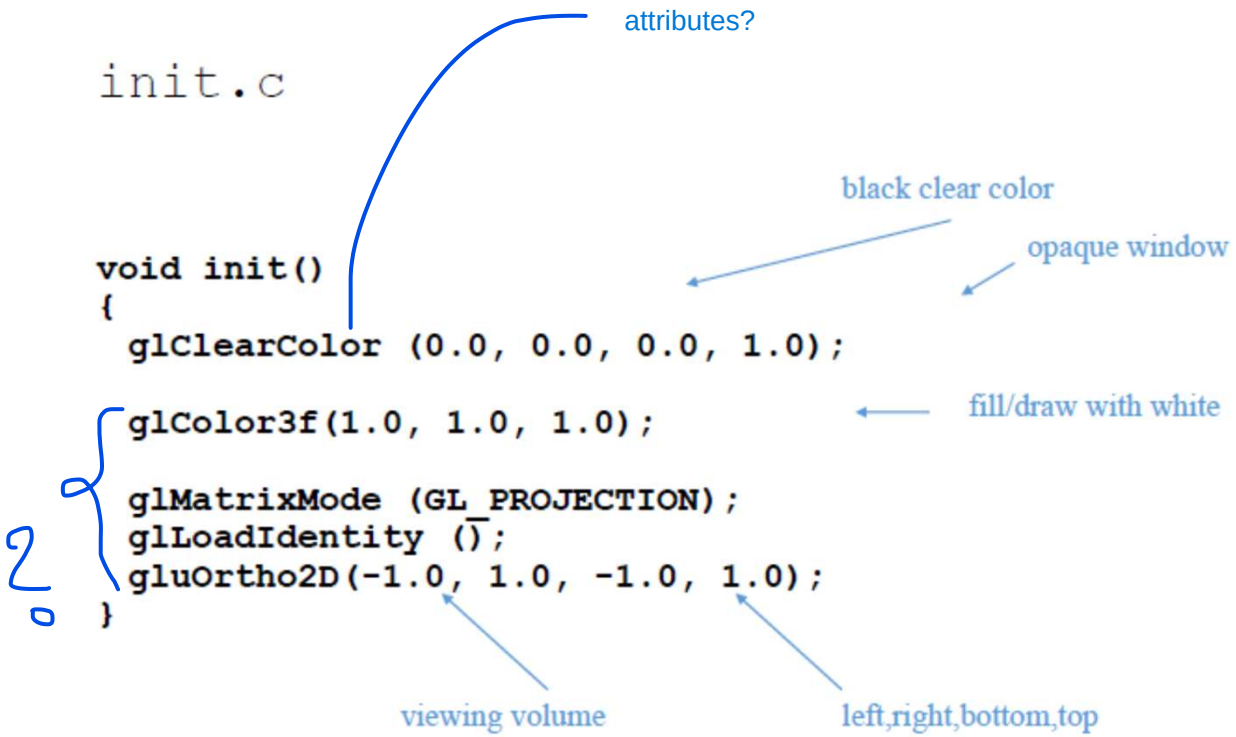
user defined ud?

also, what is the order in which these functions need to be called

# GLUT functions

- **`glutInit`** allows application to get command line arguments and initializes system
- **`gluInitDisplayMode`** requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically ORed together
- **`glutWindowSize`** in pixels
- **`glutWindowPosition`** from top-left corner of display
- **`glutCreateWindow`** create window with title "simple"
- **`glutDisplayFunc`** display callback
- **`glutMainLoop`** enter infinite event loop

Idella practical agi madidre ne gottagatte


attributes?
init.c
black clear color
opaque window
void init()
{
glClearColor (0.0, 0.0, 0.0, 1.0);
glColor3f(1.0, 1.0, 1.0);
fill/draw with white
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}
viewing volume
left,right,bottom,top

**Control Functions**

Control functions enable the programmer to communicate with the window system, initialize the program etc.

1.glutInit(int*argcp, char **argv)

2.glutCreateWindow(char *title)

3.glutInitDisplayMode()

4.glutInitWindowSize()

5.GlutInitWindowposition()

6.glutDisplayFunc()

7.glutMainLoop()

Redundant slide

**Control Functions**

Control functions enable the programmer to communicate with the window system, initialize the program etc.

1.glutInit(int*argcp, char **argv)

2.glutCreateWindow(char *title)

3.glutInitDisplayMode()

4.glutInitWindowSize()

5.GlutInitWindowposition()

6.glutDisplayFunc()

7.glutMainLoop()

# Control Functions

**1.glutInit(int*argcp, char **argv)**

•It initiates an interaction between the windowing system and OpenGL. It is used before opening a window in the program.

•The two arguments allow the user to pass command-line arguments, as in the standard C main function, and are usually the same as in main.

**2.glutCreateWindow(char *title)**

•It opens an OpenGL window.

•The title string provides title at the top to the window displayed.

# Control Functions

**3.glutInitDisplayMode()**

Used to initialize the display window created, by specifying colormodel ,buffering used etc.

**4.glutInitWindowSize()**

Used to specify the size of the window to be created.

**5.glutInitWindowPosition()**

Used to specify the position of the created window on
The monitor with respect to the top left corner.

# Control Functions

**6.glutDisplayFunc()**

•Used to specify the display callback.

•This function executes when the window is created for the firsttime.

•It is also called when the window is moved from one location on the screen to another.

**7.glutMainLoop()**

•Causes the program to begin an event processing loop.

•If there are noevents to process, then the program would enter the wait state with the output on the screen.

•It is similar to getch() in C program

idhu martogide