UNIT - 1 **7 Hours**

INTRODUCTION

Applications of computer graphics

A graphics system

Images:

Physical and synthetic

Imaging systems

The synthetic camera model

The programmer’s interface

Graphics architectures

Programmable pipelines

Performance characteristics

Graphics Programming:

The Sierpinski gasket

Programming two-dimensional applications

UNIT -1

Graphics Systems and Models

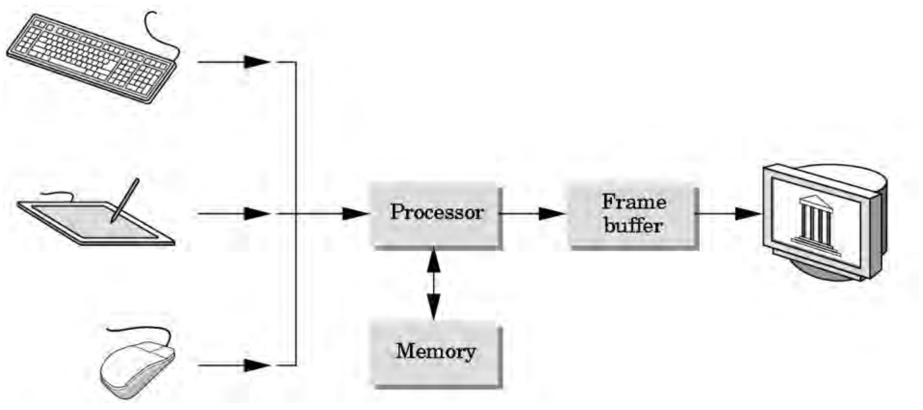
1.1 Applications of computer graphics:

- Display Of Information
- Design
- Simulation & Animation
- User Interfaces

1.2 Graphics systems

A Graphics system has 5 main elements:

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



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.

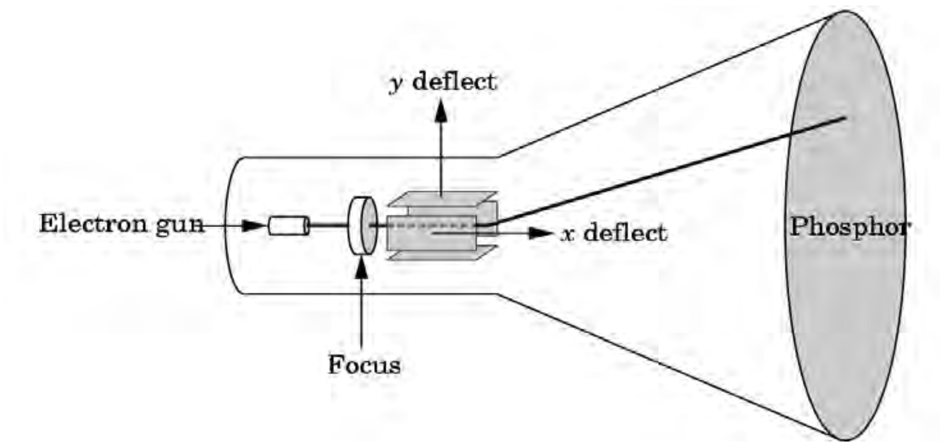
A Frame buffer is implemented either with special types of memory chips or it can be a part of system memory.

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 It is also known as Rasterization or scan conversion.

Output Devices

The most predominant type of display has been the Cathode Ray Tube (CRT).



Various parts of a CRT :

- Electron Gun – emits electron beam which strikes the phosphor coating to emit light.
- Deflection Plates – controls the direction of beam. The output of the computer is converted by digital-to-analog converters o voltages across x & y deflection plates.
- 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.

1.3 Images: Physical and synthetic

Elements of image formation:

- Objects
- Viewer
- Light source (s)

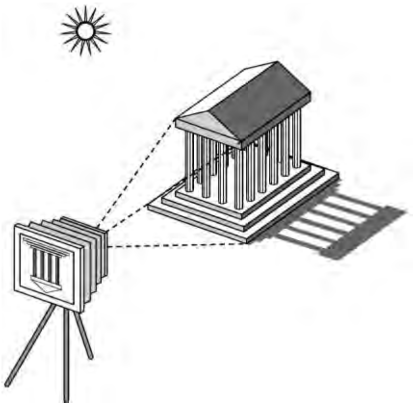
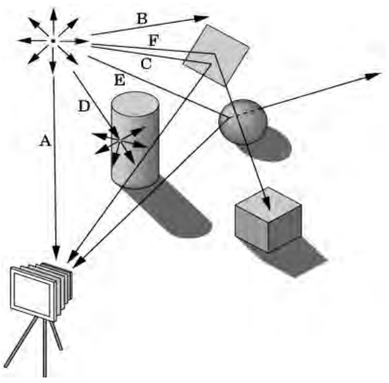


Image formation models

Ray tracing :

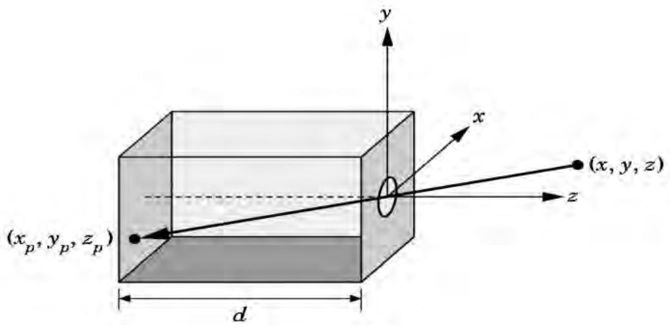
One way to form an image is to follow rays of light from a point source finding which rays enter the lens of the camera. However, each ray of light may have multiple interactions with objects before being absorbed or going to infinity.



1.4 Imaging systems

It is important to study the methods of image formation in the real world so that this could be utilized in image formation in the graphics systems as well.

1. Pinhole camera:

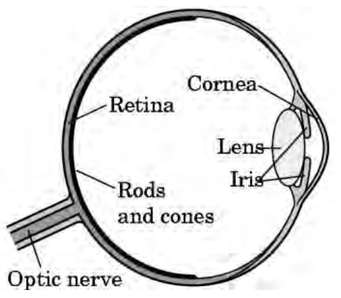


Use trigonometry to find projection of point at (x,y,z)

xp= -x/z/d yp= -y/z/d zp= d

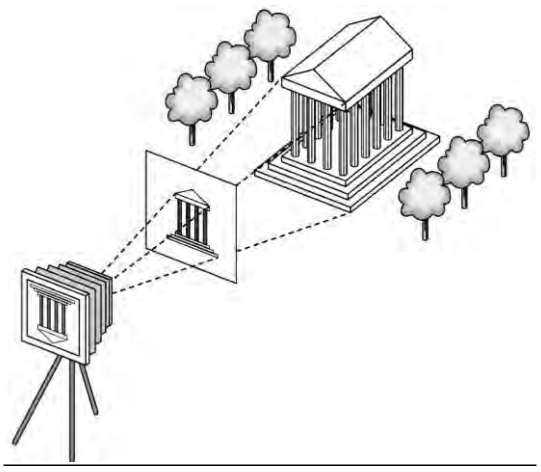
These are equations of simple perspective

2. Human visual system



- Rods are used for : 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

1.5 The Synthetic camera model



The paradigm which looks at creating a computer generated image as being similar to forming an image using an optical system.

Various notions in the model :

Center of Projection

Projector lines

Image plane

Clipping window

- In case of image formation using optical systems, the image is flipped relative to the object.
- In synthetic camera model this is avoided by introducing a plane in front of the lens which is called the image plane.

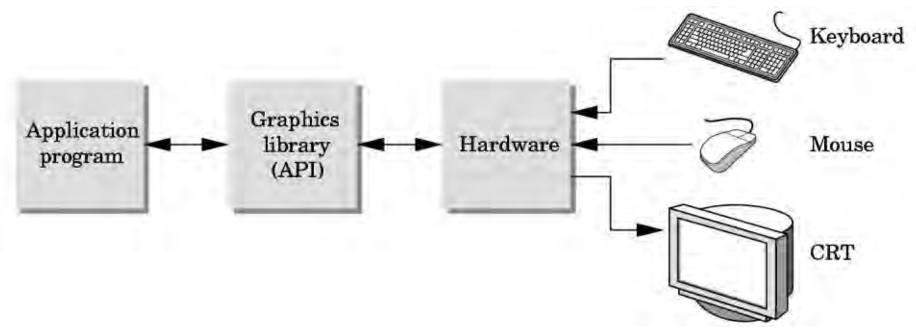
The angle of view of the camera poses a restriction on the part of the object which can be viewed.

This limitation is moved to the front of the camera by placing a Clipping Window in the projection plane.

1.6 Programer's interface :

A user interacts with the graphics system with self-contained packages and input devices. E.g. A paint editor.

This package or interface enables the user to create or modify images without having to write programs. The interface consists of a set of functions (API) that resides in a graphics library



- The application programmer uses the API functions and is shielded from the details of its implementation.
- The device driver is responsible to interpret the output of the API and converting it into a form understood by the particular hardware.

- The pen-plotter model

This is a 2-D system which moves a pen to draw images in 2 orthogonal directions.

E.g. : LOGO language implements this system.

moveto(x,y) – moves pen to (x,y) without tracing a line.

lineto(x,y) – moves pen to (x,y) by tracing a line.

Alternate raster based 2-D model :

Writes pixels directly to frame buffer

E.g. : write_pixel(x,y,color)

In order to obtain images of objects close to the real world, we need 3-D object model.

<u>3-D APIs (OpenGL - basics)</u>

To follow the synthetic camera model discussed earlier, the API should support:

Objects, viewers, light sources, material properties.

OpenGL defines primitives through a list of vertices.

Primitives: simple geometric objects having a simple relation between a list of vertices

Simple prog to draw a triangular polygon :

```
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( );
```

Specifying viewer or camera:

Position - position of the COP

Orientation – rotation of the camera along 3 axes

Focal length – determines the size of image

Film Plane – has a height & width & can be adjusted independent of orientation of lens.

Function call for camera orientation :

gluLookAt(cop_x,cop_y,cop_z,at_x,at_y,at_z,up_x,up_y,up_z);

gluPerspective(field_of_view,aspect_ratio,near,far);

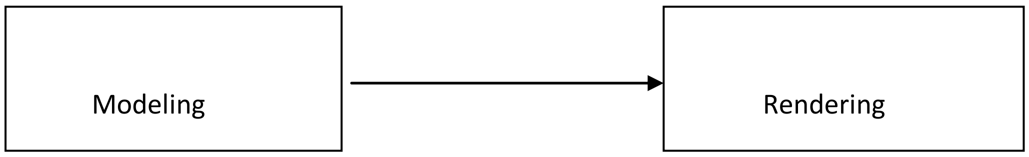
Lights and materials :

- Types of lights
  - Point sources vs distributed sources
  - Spot lights
  - Near and far sources
  - Color properties

- Material properties
  - Absorption: color properties
  - Scattering

Modeling Rendering Paradigm :

Viewing image formation as a 2 step process



E.g. Producing a single frame in an animation:

1st step : Designing and positioning objects

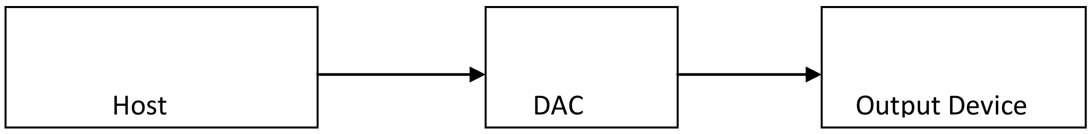
2nd step : Adding effects, light sources and other details

The interface can be a file with the model and additional info for final rendering.

1.7 Graphics Architectures

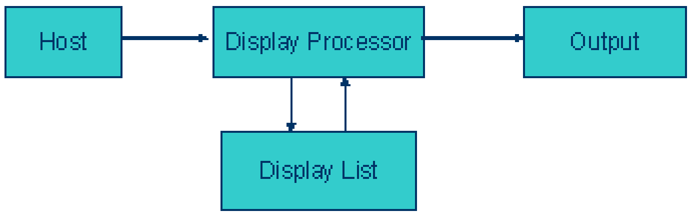
Combination of hardware and software that implements the functionality of the API.

- Early Graphics system :



Here the host system runs the application and generates vertices of the image.

Display processor architecture :



- Relieves the CPU from doing the refreshing action

- Display processor assembles instructions to generate image once & stores it in the Display List. This is executed repeatedly to avoid flicker.
- The whole process is independent of the host system.

1.8 Programmable Pipelines

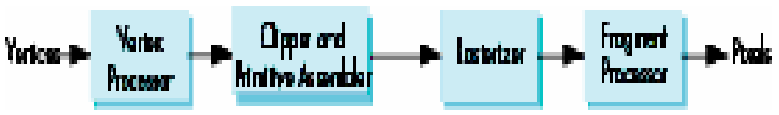
E.g. An arithmetic pipeline

Terminologies :

Latency : time taken from the first stage till the end result is produced.

Throughput : Number of outputs per given time.

Graphics Pipeline :



- Process objects one at a time in the order they are generated by the application
- All steps can be implemented in hardware on the graphics card

Vertex Processor

- 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

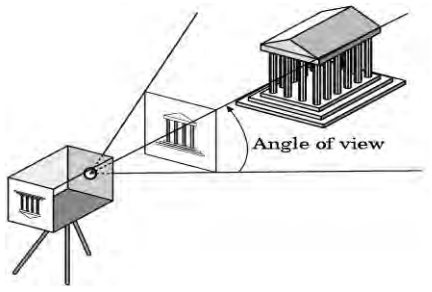
Vertices must be collected into geometric objects before clipping and rasterization can take place

- 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 Processor :

- 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

1.9 Graphics Programming

The Sierpinski Gasket :

It is an object that can be defined recursively & randomly

Basic Algorithm :

Start with 3 non-collinear points in space. Let the plane be z=0.

1. Pick an initial point (x,y,z) at random inside the triangle.
2. Select 1 of the 3 vertices in random.
3. Find the location halfway between the initial point & the randomly selected vertex.
4. Display the new point.
5. Replace the point (x,y,z) with the new point
6. Return to step 2.

Assumption : we view the 2-D space or surface as a subset of the 3-D space.

A point can be represented as p=(x,y,z). In the plane z=0, p = (x,y,0).

Vertex function genral form – glVertex*() - * is of the form ntv

n – dimensions (2,3,4)

t – data type (i,f,d)

v – if present, represents a pointer to an array.

Programing 2-D applications :

Definition of basic OpenGL types :

- E.g. – glVertex2i(Glint xi, Glint yi)

  or

```
#define GLfloat float.
GLfloat vertex[3]
glVertex3fv(vertex)
```

E.g. prog :

```
glBegin(GL_LINES);
        glVertex3f(x1,y1,z1);
        glVertex3f(x2,y2,z2);
glEnd();
```

The sierpinski gasket display() function :

```
void display()
{
        GLfloat vertices[3][3] = {{0.0,0.0,0.0},{25.0,50.0,0.0},{50.0,0.0,0.0}};
        /* an arbitrary triangle in the plane z=0 */
        GLfloat p[3] = {7.5,5.0,0.0}; /* initial point inside the triangle */
        int j,k;
        int rand();

        glBegin(GL_POINTS);
        for (k=0;k<5000;k++){
                j=rand()%3;
                p[0] = (p[0] + vertices[j][0])/2; /* compute new location */
                p[1] = (p[1] + vertices[j][1])/2;
                /* display new point */
                glVertex3fv(p);
        }
        glEnd();
        glFlush();
}
```

Coordinate Systems :

- One of the major advances in the graphics systems allows the users to work on any coordinate systems that they desire.
- The user's coordinate system is known as the "world coordinate system"
- The actual coordinate system on the output device is known as the screen coordinates.
- The graphics system is responsible to map the user's coordinate to the screen coordinate.