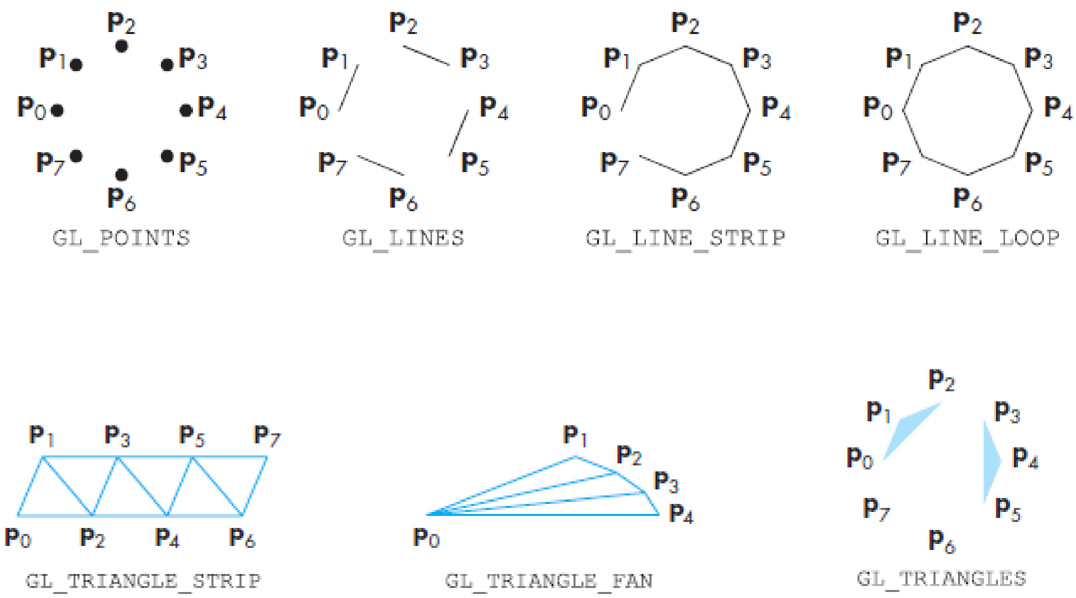
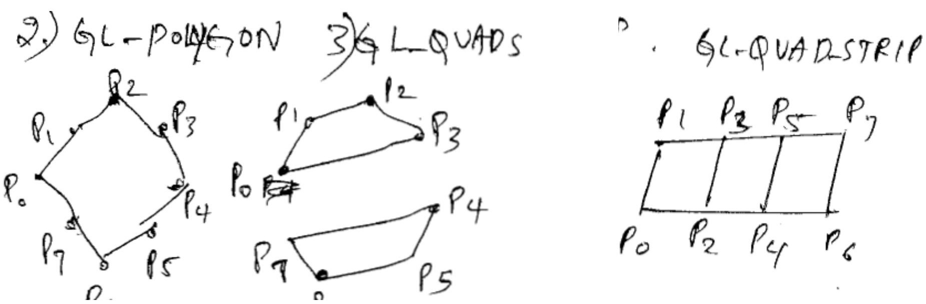
**1. Write the different OpenGL primitives, explain each primitive with an example.**

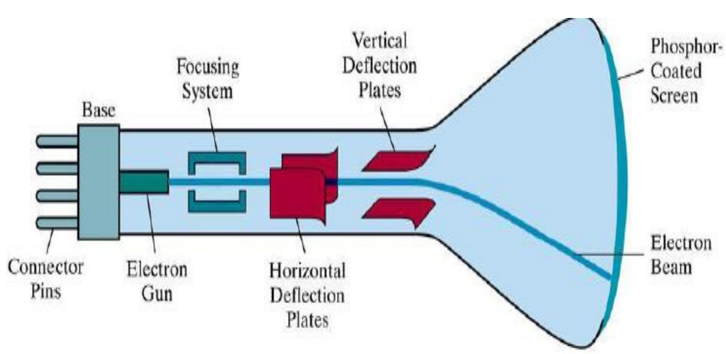


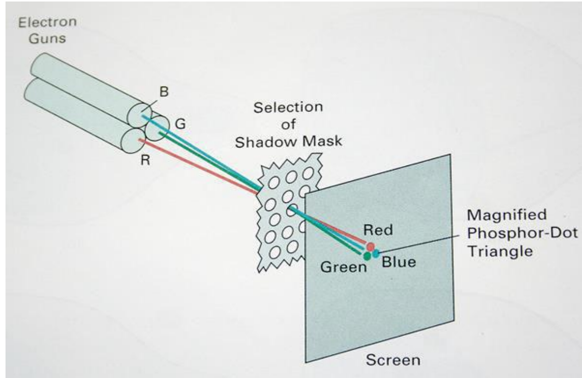




**2. With a neat diagram, explain the design and operation of cathode ray tube.**

a. Diagram. : 4Marks

b. Explanation : 6Marks






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

Various parts of a CRT:

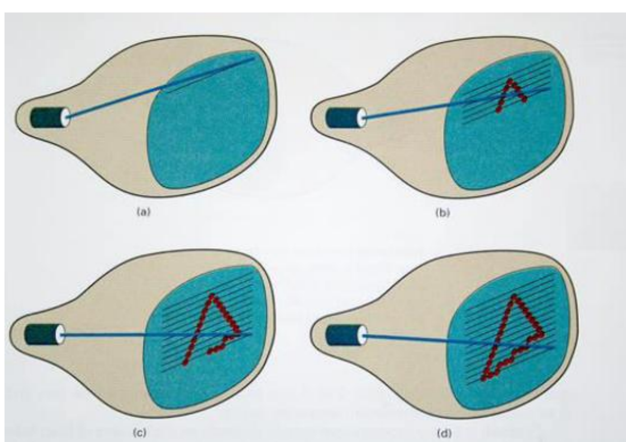
- Electron Gun – emits electron beam which strikes the phosphor coating to emit light.
- Deflection Plates – controls the direction of beam. The output of the computer is converted by digital-to-analog converters o voltages across x & y deflection plates.
- Refresh Rate – In order to view a flicker free 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:**

- No interlaced display: Pixels are displayed row by row at the refresh rate.
- Interlaced display: Odd rows and even rows are refreshed alternately.

3. **With neat diagrams, discuss about the architectures of video controller and raster scan display processor.**

**Raster-Scan Displays**

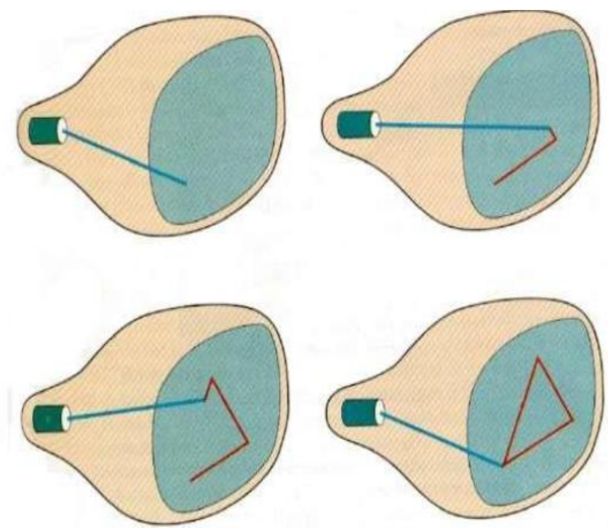



The electron beam is swept across the screen one row at a time from top to bottom.

- As it moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.

- This scanning process is called refreshing. Each complete scanning of a screen is normally called a frame.
- The refreshing rate, called the frame rate, is normally 60 to 80 frames per second, or described as 60 Hz to 80 Hz.
- Picture definition is stored in a memory area called the frame buffer.
- This frame buffer stores the intensity values for all the screen points. Each screen point is called a pixel (picture element).
- Property of raster scan is Aspect ratio, which defined as number of pixel columns divided by number of scan lines that can be displayed by the system.

**Random-Scan Displays**



When operated as a random-scan display unit, a CRT has the electron beam directed only to those parts of the screen where a picture is to be displayed.

- Pictures are generated as line drawings, with the electron beam tracing out the component lines one after the other.
- For this reason, random-scan monitors are also referred to as vector displays (or stroke writing displays or calligraphic displays).
- The component lines of a picture can be drawn and refreshed by a random-scan system in any specified order
- A pen plotter operates in a similar way and is an example of a random-scan, hard-copy device.
- Refresh rate on a random-scan system depends on the number of lines to be displayed on that system.

- Picture definition is now stored as a set of line-drawing commands in an area of memory referred to as the display list, refresh display file, vector file, or display program
- To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn.
- After all line-drawing commands have been processed, the system cycles back to the first line command in the list.
- Random-scan displays are designed to draw all the component lines of a picture 30 to 6 times each second, with up to 100,000 "short" lines in the display list.
- When a small set of lines is to be displayed, each refresh cycle is delayed to avoid very high refresh rates, which could burn out the phosphor.

4. **Write the algorithm for Bresenham's line drawing algorithm for m<1.0, and digitize the line segment with end points (20, 10) to (30, 18).**

**Bresenham's Algorithm:**

- It is an efficient raster scan generating algorithm that uses incremental integral calculations
- To illustrate Bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.0.
- Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x0, y0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.
- Consider the equation of a straight line y=mx+c where m=dy/dx

**Bresenham's Line-Drawing Algorithm for |m| < 1.0**

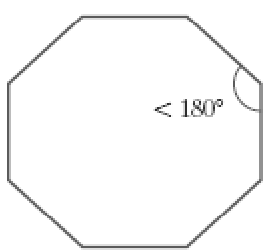
- Input the two line endpoints and store the left endpoint in (x0, y0).
- Set the color for frame-buffer position (x0, y0); i.e., plot the first point.
- Calculate the constants Δx, Δy, 2Δy, and 2Δy − 2Δx, and obtain the starting value for the decision parameter as,
    - p0 = 2Δy –Δx
- At each xk along the line, starting at k = 0, perform the following test:
    - If pk < 0, the next point to plot is
        - (xk + 1, yk ) and pk+1 = pk + 2Δy
    - Otherwise, the next point to plot is
        - (xk + 1, yk + 1) and pk+1 = pk + 2Δy − 2Δx
- Repeat step 4 Δx − 1 more times.

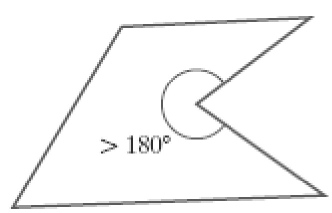
## 5. Write an OpenGL recursive program for generating 3D Sierpinski gasket

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point                    v[]={{0.0,0.0,1.0},{0.0,0.942809,-0.33333},{-0.816497,-0.471405,-
0.33333},{0.816497,-0.471405,-0.333333}};
static GLfloat theta[]={0.0,0.0,0.0};
int n;

void triangle(point a,point b,point c)
{
        glBegin(GL_POLYGON);
        glNormal3fv(a);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}

void divide_triangle(point a,point b,point c,int m)
{
        point v1,v2,v3;
        int j;
        if(m>0)
        {
                for(j=0;j<3;j++)
                        v1[j]=(a[j]+b[j])/2;
                for(j=0;j<3;j++)
                        v2[j]=(a[j]+c[j])/2;
                for(j=0;j<3;j++)
                        v3[j]=(b[j]+c[j])/2;
                divide_triangle(a,v1,v2,m-1);
                divide_triangle(c,v2,v3,m-1);
                divide_triangle(b,v3,v1,m-1);
        }
        else(triangle(a,b,c));
}

        void tetrahedron(int m)
        {
                glColor3f(1.01,0.0,0.0);
                divide_triangle(v[0],v[1],v[2],m);
                glColor3f(0.1,1.0,0.0);
    divide_triangle(v[3],v[2],v[1],m);
                glColor3f(0.0,0.0,0.11);
    divide_triangle(v[0],v[3],v[1],m);
                glColor3f(0.0,0.0,0.01);
                divide_triangle(v[0],v[2],v[3],m);
        }
        void display(void)
        {
                glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_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,-2.0,2.0,-10.0,10.0);
            else
            glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);

            glMatrixMode(GL_MODELVIEW);
            glutPostRedisplay();
}
void main(int argc,char **argv)
{
            printf("no of divisions ?");
            scanf("%d",&n);
            glutInit(&argc,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();
}
```

**6. What are the polygon classifications? How to identify a convex polygon? Illustrate how to split a concave polygon**




Polygons are classified into two types

- Convex Polygon and
- 2. Concave Polygon

Convex Polygon:

- The polygon is convex if all interior angles of a polygon are less than or equal to 180◦, where an interior angle of a polygon is an angle inside the polygon boundary that is formed by two adjacent edges.

- An equivalent definition of a convex polygon is that its interior lies completely on one side of the infinite extension line of any one of its edges.
- Also, if we select any two points in the interior of a convex polygon, the line segment joining the two points is also in the interior.

Concave Polygon:

- A polygon that is not convex is called a concave polygon.

**Identifying interior and exterior region of polygon**

- We may want to specify a complex fill region with intersecting edges.
- For such shapes, it is not always clear which regions of the xy plane we should call "interior" and which regions.
- We should designate as "exterior" to the object boundaries.
- Two commonly used algorithms
    - Odd-Even rule and
    - The nonzero winding-number rule.

**Inside-Outside Tests**

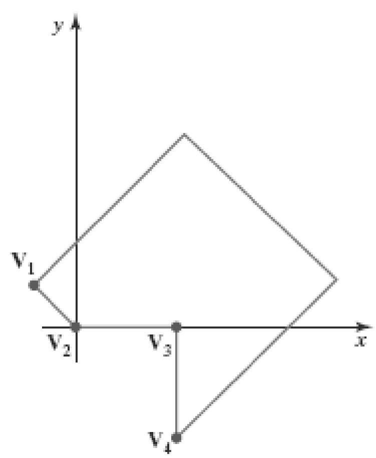
- Also called the odd-party rule or the even-odd rule.
- Draw a line from any position P to a distant point outside the coordinate extents of the closed polyline.
- Then we count the number of line-segment crossings along this line.
- If the number of segments crossed by this line is odd, then P is considered to be an interior point Otherwise, P is an exterior point
- We can use this procedure, for example, to fill the interior region between two concentric circles or two concentric polygons with a specified color.

**Nonzero Winding-Number rule**

- This counts the number of times that the boundary of an object "winds" around a particular point in the counterclockwise direction termed as winding number,
- Initialize the winding number to 0 and again imagining a line drawn from any position P to a distant point beyond the coordinate extents of the object.
- The line we choose must not pass through any endpoint coordinates.

- As we move along the line from position P to the distant point, we count the number of object line segments that cross the reference line in each direction
- We add 1 to the winding number every time we intersect a segment that crosses the line in the direction from right to left, and we subtract 1 very time we intersect a segment that crosses from left to right
- If the winding number is nonzero, P is considered to be an interior point. Otherwise, P is taken to be an exterior point
- The nonzero winding-number rule tends to classify as interior some areas that the oddeven rule deems to be exterior.
- Variations of the nonzero winding-number rule can be used to define interior regions in other ways define a point to be interior if its winding number is positive or if it is negative; or we could use any other rule to generate a variety of fill shapes

**Rotational method**




Proceeding counterclockwise around the polygon edges, we shift the position of the polygon so that each vertex Vk in turn is at the coordinate origin.

- We rotate the polygon about the origin in a clockwise direction so that the next vertex Vk+1 is on the x axis.
- If the following vertex, Vk+2, is below the x axis, the polygon is concave.
- We then split the polygon along the x axis to form two new polygons, and we repeat the concave test for each of the two new polygons

7. **Write and explain midpoint circle algorithm.**

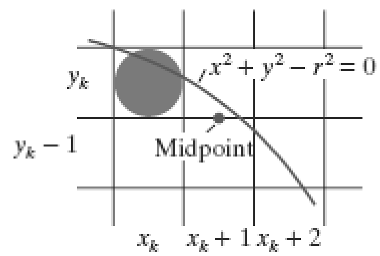
**Midpoint Circle Concept**

- Midpoint circle algorithm generates all points on a circle centered at the origin by incrementing all the way around circle.
- The strategy is to select which of 2 pixels is closer to the circle by evaluating a function at the midpoint between the 2 pixels
- To apply the midpoint method, we define a circle function as

$$f_{circ}(x, y) = x^2 + y^2 - r^2$$

- To summarize, the relative position of any point (x, y) can be determined by checking the sign of the circle function as follows:

$$f_{circ}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases}$$




**Midpoint Circle Algorithm**

- Input radius r and circle center (xc , yc ), then set the coordinates for the first point on the circumference of a circle centered on the origin as

  (x0, y0) = (0, r )
- Calculate the initial value of the decision parameter as

  p0 = 1-r
- At each xk position, starting at k = 0, perform the following test:
- If pk <0, the next point along the circle centered on (0, 0) is (xk+1, yk ) and

  pk+1 = pk + 2xk+1 + 1
- Otherwise, the next point along the circle is (xk + 1, yk − 1) and

  pk+1 = pk + 2xk+1 + 1 – 2yk+1 where 2xk+1 = 2xk + 2 and 2yk+1= 2yk − 2.
- Determine symmetry points in the other seven octants.
- Move each calculated pixel position (x, y) onto the circular path centered at (xc , yc ) and plot the coordinate values as follows: x = x + xc , y = y + yc
- Repeat steps 3 through 5 until x ≥ y.