NAME : SOHAM GANGULY

ROLL NO. : 00191100101 8

DEPARTMENT : INFORMATION TECHNOLOGY

YEAR : 2ND

SEMESTER : 4TH

SUBJECT : GRAPHICS & GEOMETRIC MODELLING

DATE : 5TH JUNE, 2021

NAME: SOHAM GANGULY ROLL NO.: 001911001018

CO1

1a) Resolution = 1280 x 1024

Refresh rate = 60 frames per second

Scan rate for each pixel row = 60 frames/second × 1024 lines/frame

= 61440 lines/second

[Scan rate = refresh rate × number of scan lines per frame]

Each of these 1024 lines contains 1280 pixels

As refresh rate = 60 frames/second

$\Rightarrow$ 1 frame takes time = $\frac{1}{60}$ second

$\therefore$ Number of scan lines in 1 frame buffer = 1024

$\therefore$ 1024 scan lines take time = $\frac{1024}{1024 \times 60}$ second

= 16.27 µs

≈ 16.3 µs

Ans:- Time taken for scanning = 16.3µs.

1b) A megapixel (MP) is equal to a unit of million pixels. For a digital camera this also expresses the number of image sensor elements it possesses. For a digital display it is the number of display elements.

For example if we have a ~~1024~~ screen of resolution 1280x800 pixels, we would say the resolution is 1.024 Megapixels as it is the total pixel count.

NAME: SOHAM GANGULY ROLL NO.: 001911001018

1b) A camera can have various aspect ratios. Considering the common aspect ratio of 4:3, a 48 megapixel camera, thus possessing 48 million pixels will have a resolution of approximately 8000x 6000 pixels i.e. 8000 horizontal pixels by 6000 vertical pixels.

~~1c)~~

1c) Liquid Crystal Display (LCD):

LCD device produces a picture through passing polarized light from surroundings or an internal light source through liquid crystal material that transmits the light.

At the two far ends of the LCD panel, non-alkaline, transparent glass substrates are attached to polarizer film which will transmit or absorb a specific component of the polarised light.

In between the 2 glass substrates there exists a layer of nematic liquid crystal, along with a colour filter containing the 3 primary colours (red, green, blue).

~~Each of~~ The polarized glasses are arranged at right angles to each other, and when electric current is passed through the LCD panel, the liquid crystals get aligned with the first polarized glass and make a 90° twist when approaching the other polarized glass at the end.

As a result of this, the light from the fluorescent backlight source is able to pass through and create an illuminated pixel on the screen.

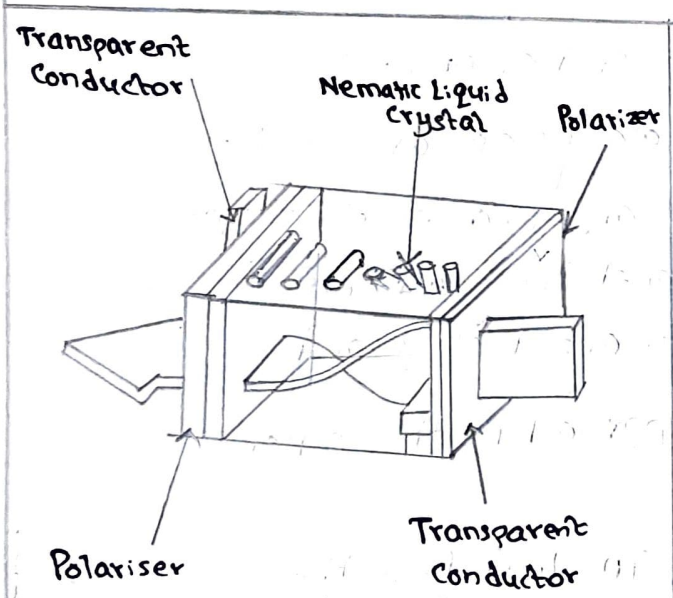
In the absence of electric current, liquid crystals ~~twist~~ do not twist and hence light does not pass creating a black pixel on the appropriate position on the screen.

NAME: SOHAM GANGULY ROLL NO.: 00191100101 8

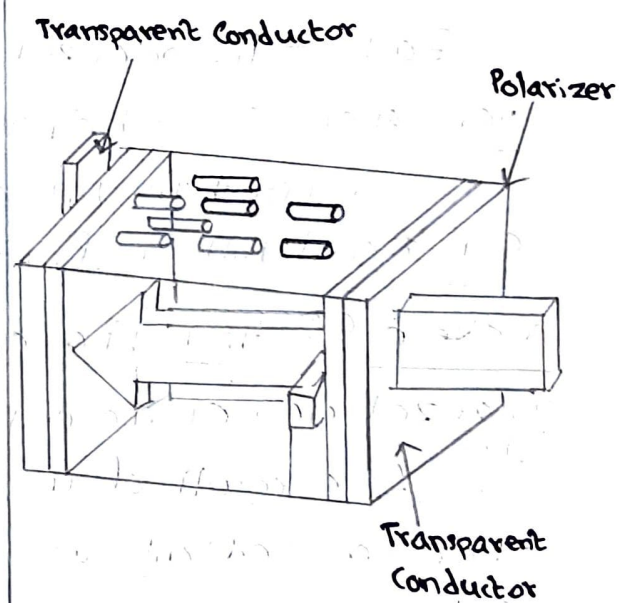
10) In LCDs, we also have a pair of polarising filters alongwith the liquid crystals to control the emitted light.

With nematic crystals in the natural state, the light wave will turn horizontally. The first polarising filter only allows vertically aligned light waves to remain. When the electric current passes, the liquid crystals straighten up and the cell will not bend the light.

The second filter will only allow transmission of horizontal light waves and thus, light passing through straight liquid crystals will get blocked by it. And so, in the off state, it will reflect back towards the source.



LCD - ON STATE



LCD - OFF STATE

NAME: SOHAM GANGULY ROLL NO.: 001911001018

1d) Given:-

Length of video monitor (L) = 12 inches

Width of video monitor (B) = 9.6 inches

Monitor resolution = 1280 x 1024 pixels

Aspect ratio (r) = 1

Number of pixels along the length = 1280

Number of pixels along the width = 1024

With aspect ratio = 1,

$$1280 \text{ pixels} \equiv 12 \text{ inches}$$

$$\Rightarrow 1 \text{ pixel} \equiv \frac{12}{1280} \text{ inches} = 0.009375 \text{ inches}$$

For width,

$$1024 \text{ pixels} \equiv 9.6 \text{ inches}$$

$$\Rightarrow 1 \text{ pixel} = \frac{9.6}{1024} \text{ inches} = 0.009375 \text{ inches}$$

∴ Diameter of each point on screen = 0.009375 inches.

NAME: SOHAM GANGULY ROLL NO.: 001911001018

CO2

2. Line generation using midpoint algorithm:-

considering $0 < \text{slope} < 1$

Given endpoints:- (x_0, y_0), (x_n, y_n)

At some iteration of our algorithm let us be at point (x_k, y_k)

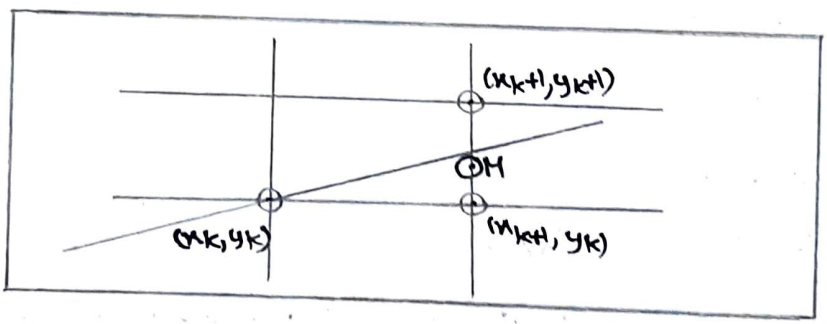
For next point we have 2 possible candidates:

(x_{k+1}, y_k) and (x_{k+1}, y_{k+1}) $[x_{k+1} = x_k + 1,\ y_{k+1} = y_k + 1]$

We find the midpoint of the two:-

$$M\left(\frac{x_k + x_{k+1}}{2}, \frac{y_k + y_{k+1}}{2}\right)$$

$$= M\left(x_{k+1}, y_k + \frac{1}{2}\right)$$



If midpoint M lies above the line we plot the lower point (x_{k+1}, y_k)

If it lies below the line we plot the upper point (x_{k+1}, y_{k+1})

We need a function that tells us if point lies above or below a line

Equation of straight line:- $y = mx + c$ where m = slope, c = y-intercept

We know, $m = \frac{dy}{dx}$

NAME: SOHAM GANGULY ROLL NO.: 001911001018

2. $\Rightarrow y = \frac{dy}{dx}.x + c$

$\Rightarrow (dx)y = (dy)x + (dx)c$

$\Rightarrow (dy)x - (dx)y + (dx)c = 0$

Let $F(x,y) = (dy)x - (dx)y + (dx)c$

For all points above the line, $F(x,y) < 0$

[∵ value of y will be large and $-(dx)y$ is highly negative]

For all points above line $F(x,y) > 0$

For all points on the line, $F(x,y) = 0$

Thus our decision parameter $d_k = F(M)$

if $d_k < 0$ we choose lower point

elseif $d_k \geq 0$ we choose upper point

As M is $(x_{k+1}, y_{k+1/2})$

$d_k = F(x_{k+1}, y_{k+1/2})$

Let $F(x,y) = Ax + By + c$

where $A = dy$, $B = (-dx)$, $C = (dx)c$

$\therefore d_k = F(x_{k+1}, y_{k+1/2}) = A(x_{k+1}) + B(y_{k+1/2}) + c$

Also,

$d_{k+1} = F(x_{k+1}+1, y_{k+1}+1/2) = A(x_{k+1}+1) + B(y_{k+1}+1/2) + c$

As we choose x values successively for x, $x_{k+1} = x_{k+1}$

$\therefore d_{k+1} - d_k = A(x_{k+1}+1) - A(x_{k+1}) + B(y_{k+1}+1/2) - B(y_k + 1/2) + c - c$

$= A(x_{k+1}+1 - x_{k-1}) + B(y_{k+1} + 1/2 - y_{k-1/2})$

$= A + B(y_{k+1} - y_k)$

Since we choose either (x_{k+1}, y_k) or (x_{k+1}, y_{k+1})

NAME: SOHAM GANGULY ROLL NO.: 001911009018

2.

With $y_{k+1} = y_k$ if $d_k < 0$

and $y_{k+1} = y_k + 1$ if $d_k \geq 0$

$$\therefore d_{k+1} = \begin{cases} d_k + A, & d_k < 0 \\ d_k + A + B, & d_k \geq 0 \end{cases}$$

$$\Rightarrow \boxed{d_{k+1} = \begin{cases} d_k + dy, & d_k < 0 \\ d_k + dy - dx, & d_k \geq 0 \end{cases}}$$

Calculating initial value

$d_0 = F(x_0 + 1, y_0 + 1/2)$

$= A(x_0 + 1) + B(y_0 + 1/2) + C$

$= Ax_0 + By_0 + C + A + B/2$

$= F(x_0, y_0) + A + B/2$

$= 0 + A + B/2$ [$\because (x_0, y_0)$ lies on the line]

$= dy - \frac{dx}{2}$

$$\therefore \boxed{d_0 = dy - \frac{dx}{2}}$$

and

$$\boxed{d_{k+1} = \begin{cases} d_k + dy, & \cancel{d_k \to 0}\ d_k < 0 \\ d_k + dy - dx, & \cancel{d_k < 0}\ d_k \geq 0 \end{cases}}$$

In Bresenham's line drawing algorithm,

The decision parameter $p_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$

$$p_{k+1} = \begin{cases} p_k + 2\Delta y, & p_k < 0 \\ p_k + 2\Delta y - 2\Delta x, & p_k \geq 0 \end{cases}$$

NAME: SOHAM GANGULY ROLL NO.: 001911001018

2. $\boxed{p_0 = 2\Delta y - \Delta x}$

Here Δy and Δx carry the same meaning as dy and dx in Midpoint algorithm

$$\boxed{\begin{aligned}\Delta y &= dy = y_n - y_0 \\ \Delta x &= dx = x_n - x_0\end{aligned}}$$

Inspecting, we see that the Bresenham line drawing algorithm's decision parameters are same as Midpoint algorithm's decision parameter scaled by a factor of 2.

Thus, we can see that the two decision parameters are ultimately the same, since our only comparisons are with zero and scaling doesn't matter in such ~~eom~~ cases.

NAME: SOHAM GANGULY ROLL NO.: 001911001018

CO3

3. Let us take a point (x, y)

Converting to homogeneous coordinates, the point is represented as $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

For rotation, around origin O,

the rotation matrix $R_1(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

the scaling matrix $S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ [Scaling factor for x-value $= S_x$, scaling factor for y-value $= S_y$]

Let us consider the products,

i) $S \times R_1(\theta) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$= \begin{bmatrix} S_x\cos\theta & -S_x\sin\theta & 0 \\ S_y\sin\theta & S_y\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

ii) $R_1(\theta) \times S = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$= \begin{bmatrix} S_x\cos\theta & -S_y\sin\theta & 0 \\ S_x\sin\theta & S_y\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

NAME: SOHAM GANGULY ROLL NO.: 001911001018

3. For rotation and scaling to be commutative,

$$R_1(\theta) \times S = S \times R_1(\theta)$$

$$\Rightarrow \begin{bmatrix} S_x\cos\theta & -S_y\sin\theta & 0 \\ S_x\sin\theta & S_y\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} S_x\cos\theta & -S_x\sin\theta & 0 \\ S_y\sin\theta & S_y\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Comparing, these two are equal only if :- (considering $\sin\theta \neq 0$)

$$-S_y \sin\theta = -S_x \sin\theta$$

$$\Rightarrow S_x = S_y$$

$$S_x \sin\theta = S_y \sin\theta$$

$$\Rightarrow S_x = S_y$$

∴ We see that these two matrix products are only equal if $S_x = S_y$ i.e. the scaling is uniform and not otherwise.

Thus it is proved that uniform scaling and rotation form a commutative pair of operations but in general, scaling and rotation are not a commutative pair of operations.

NAME: SOHAM GANGULY ROLL NO.: 001911001018

COA

4.

```
#include<stdio.h>
#include<stdlib.h>

struct RGB
{
  int red; //red value
  int green; //green value
  int blue; //blue value
};
typedef struct RGB RGB;

RGB interpolate_colour(RGB startcolour, RGB endcolour, int itr, int totalitr)
{
  RGB newcolour; //store interpolated colour
  float step = (1.0*itr)/tot_itrs

  newcolour.red = (int)(startcolour.red + (endcolour.red-startcolour.red)*step);
  newcolour.green = (int)(startcolour.green + (endcolour.green - startcolour.green)*
                                                                        step);
  newcolour.blue = (int)(startcolour.blue + (endcolour.blue - startcolour.blue)*step);
  return newcolour;
}

void interpolation(RGB startcolour, RGB endcolour, int iterations)
{
   RGB setofcolours[iterations]; //store set of colours
   for(int i=1; i<= iterationss i++)
   {
   }
   setofcolours[i-1] = interpolate_colour(startcolour, endcolour, i, iterations);
   printf("Start Colour : (%d, %d, %d)\n", startcolour.red, startcolour.green,
                                                        startcolour.blue);
```

NAME: SOHAM GANGULY ROLL NO.: 001911001018

4.
```
for(int i=0; i<iterations; i++)
{
  printf("Interpolated colour no. %d : (%d, %d, %d)\n", (i+1),
        setofcolours[i].red, setofcolours[i].green, setofcolours[i].blue);
}
printf("End colour: (%d, %d, %d)\n", endcolour.red, endcolour.green,
                        endcolour.blue);
}

int main()
{
  RGB startcolour, endcolour;
  int maxiter;
  printf("Enter RGB values of start colour: ");
  scanf("%d %d %d", &startcolour.red, &startcolour.green, &startcolour.blue);
  printf("Enter RGB values of end colour: ");
  scanf("%d %d %d", &endcolour.red, &endcolour.green, &endcolour.blue);
  printf("Enter number of steps of interpolation: ");
  scanf("%d", &maxiter);
  maxiter++;
  interpolation(startcolour, endcolour, maxiter);
}
```

In our program we treat the RGB colours as vectors in the RGB space, linearly interpolating each component to get the intermediate colours. The number of intermediate colours is equal to the number of steps of interpolation with the set of colours we store ending with our end colour (hence increment maxiter by 1).

NAME: SOHAM GANGULY ROLL NO.:001911001018

CO5

5. A cylinder is a 3 dimensional object consisting of a pair of circular bases at a distance from each other connected by a curved surface which is always at a fixed distance from the axis connecting the centres of the two bases.

To represent a cylinder we utilise a polygon mesh representation.

Some of the advantages of a polygon mesh are:-

i) Unlike ordinary polygon ~~repre~~ surfaces, it can be used to efficiently represent arbitrarily large shapes.

ii) It can be used to render any object

iii) It is easy to represent and transform

iv) It is easy to draw on a computer screen.

For the efficient computer representation of the polygon mesh we use adjacency list and more specifically the winged edge representation. We keep three types of records:

i) vertex records

ii) edge records

iii) face records

Vertex records: stores each vertex's coordinates and reference to 1 edge

Face records: stores ~~each faces~~ a reference to one of the edges surrounding the face.

Edge records: stores 2 references to the two endpoint vertices of each edge, two references to faces on either side and

5. four references to the previous and next edges surraunding the two faces.

The advantages of this representation are that we can handle adjacency queries in O(1), it uses very little extra storage, and it can handle arbitrary polygons.

Some of our reasons for selecting polygon mesh are:-

i) Polygon meshes can represent arbitrarily large objects effiriently unlike polygonal surface.

ii) All lines connecting two points lie entirely within the structure of the cylinder. This, property, called convexity, goes well with mesh structures.

iii) A cylinder is a solid enclosed polygon enclosing a finite amount of space and meshes can represent such structures easily

iv) A mesh represents planar surfaces well and hence the curved external surface of the cylinder is hence represented easily.

~~v) There is no need for~~

Thus we can use meshing, most commonly with triangular meshes, to create the surface of the cylinder and use an efficient winged edge representation to handle adjacency related questions in O(1) time complexity.

Hence, polygonal meshes is our choice of method of representation.

CO6

6. We assume that the rectangular area will extend from (0,0) to (length, breadth) with (length, breadth) being its dimensions.

We consider that our filled circle is identified by:-

i) centre (x,y)
ii) radius (r)
iii) position indicated as (x_r, y_r)
iv) velocity indicated as (v_x, v_y)

The animation will be finite determined by number of iterations. In each iteration: we assume that 'dt' finite discrete time has passed and update position of circle according to:

i) if no collision occurs then the position is updated linearly as per the present velocity.
ii) if collision occurs velocity and position are updated as per elastic collision rules.

As there is no standard graphics header file, we assume a header file "graphics.h" with functionality including:

i) initwindow(width, height): initialises a graphics window with dimensions width x height. This window corresponds to the rectangular area.

ii) clearwindow(): clears graphics window so that the circle with updated position maybe drawn in the next iteration.

iii) fillcircle(x,y,radius, colour): draws a circle of given radius centred at (x,y) with given fill colour.

iv) sleep (milliseconds): pauses thread for amount of time in milliseconds and is used to fix the frame rate to prevent it from exceeding human visual capacity.

Our program:

```
#include<stdio.h>
#include<stdlib.h>
#include "graphics.h"

 struct Circle
{
  double x; //x-coordinate of centre
  double y; //y-coordinate of centre
  double vx; //x-component of velocity
  double vy; //y-component of velocity
  double rad; //radius
 };

typedef struct Circle Circle;

 int length, breadth; //dimensions of rectangle


void updatePos(Circle *c, double dt)
{
  double xfix=0, yfix=0;
  if(c->y + c->vy * dt < c->rad)
  { //~~c~~ indicates collision with horizontal edge at y=0
    c->vy = -c->vy;
    yfix = -(c->y - c->rad);
  }
 else if(c->y + c->vy*dt > breadth - c->rad)
 { //indicates collision with horizontal edge at y=breadth
   c->vy = -c->vy; yfix = breadth - (c->y + c->rad);}
```

6.

```
if(c->x + c->vx*dt < c->rad)
{//indicates collision with vertical edge at x=0
  c->vx = -c->vx;
  xfix = -((c->x - c->rad));
}
else if(c->x+c->vx*dt > length - c->rad)
{ //indicates collision with ~~horizontal~~ vertical edge at x=length
  c->vx = -c->vx;
  xfix = length - (c->x +c->rad~~xfix~~);
}

c->x += (c->vx*dt + 2*xfix); //update of
c->y += (c->vy*dt +2*yfix);
}

//simulation
void animate(circle c, int maxiter)
{
  initwindow(length,breadth);
  double dt= 0.25;
  for(int i=0; i<maxiter; i++)
  {
    ~~updatepos~~
    fillcircle(c.x, c.y, c.rad, Color.RED); //displaying circle
    updatepos(&c, dt); //updating position
    sleep(200); //pause
    clearwindow(); // clear for next frame
  }
}

int main() //demo
{
 length=1000;
 breadth=500;
```

6.
```
Circle c;
c.x = 200;
c.y = 250;
c.vx = 2.2;
c.vy = -1.3;
c.rad = 5;
animate(c, 5000);
}
```