

LAB - 7-1

7. Generate fractal patterns using i) Bezier ii) Koch Curve

i) Bezier

Source Code :

```
#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<SDL2/SDL.h>

SDL_Window* window = NULL;

SDL_Renderer* renderer = NULL;

int mousePosX , mousePosY ;

int xnew , ynew ;

void drawCircle(int xc, int yc, int x, int y)
{
    SDL_RenderDrawPoint(renderer,xc+x,yc+y) ;
    SDL_RenderDrawPoint(renderer,xc-x,yc+y);
    SDL_RenderDrawPoint(renderer,xc+x,yc-y);
    SDL_RenderDrawPoint(renderer,xc-x,yc-y);
    SDL_RenderDrawPoint(renderer,xc+y,yc+x);
    SDL_RenderDrawPoint(renderer,xc-y,yc+x);
    SDL_RenderDrawPoint(renderer,xc+y,yc-x);
    SDL_RenderDrawPoint(renderer,xc-y,yc-x);
}
```

```

void circleBres(int xc, int yc, int r)
{
    int x = 0, y = r;
    int d = 3 - 2 * r;
    while (y >= x)
    {

        drawCircle(xc, yc, x, y);
        x++;

        if (d > 0)
        {
            y--;
            d = d + 4 * (x - y) + 10;
        }
        else
            d = d + 4 * x + 6;
        drawCircle(xc, yc, x, y);
    }
}

```

```

void bezierCurve(int x[] , int y[])
{
    double xu = 0.0 , yu = 0.0 , u = 0.0 ;
    int i = 0 ;
    for(u = 0.0 ; u <= 1.0 ; u += 0.0001)
    {
        xu = pow(1-u,3)*x[0]+3*u*pow(1-u,2)*x[1]+3*pow(u,2)*(1-u)*x[2]
            +pow(u,3)*x[3];
    }
}

```

```

        yu = pow(1-u,3)*y[0]+3*u*pow(1-u,2)*y[1]+3*pow(u,2)*(1-u)*y[2]
            +pow(u,3)*y[3];
        SDL_RenderDrawPoint(renderer , (int)xu , (int)yu) ;
    }
}

int main(int argc, char* argv[])
{

    if (SDL_Init(SDL_INIT_EVERYTHING) == 0)
    {

        if(SDL_CreateWindowAndRenderer(640, 480, 0, &window, &renderer) == 0)
        {

            SDL_bool done = SDL_FALSE;

            int i = 0 ;
            int x[4] , y[4] , flagDrawn = 0 ;

            while (!done)
            {

                SDL_Event event;

                SDL_SetRenderDrawColor(renderer, 0, 0, 0,
SDL_ALPHA_OPAQUE);

                SDL_RenderClear(renderer);

                SDL_SetRenderDrawColor(renderer, 255, 255, 255,
SDL_ALPHA_OPAQUE);

```

```
if(i==4)
{
    bezierCurve(x , y) ;
    flagDrawn = 1 ;
}
```

```
SDL_SetRenderDrawColor(renderer, 128, 128, 128,
SDL_ALPHA_OPAQUE);
circleBres(x[0] , y[0] , 8) ;
SDL_SetRenderDrawColor(renderer, 255, 0, 0,
SDL_ALPHA_OPAQUE);
SDL_RenderDrawLine(renderer , x[0] , y[0] , x[1] , y[1]) ;
```

```
SDL_SetRenderDrawColor(renderer, 128, 128, 128,
SDL_ALPHA_OPAQUE);
circleBres(x[1] , y[1] , 8) ;
```

```
SDL_SetRenderDrawColor(renderer, 255, 0, 0,
SDL_ALPHA_OPAQUE);
SDL_RenderDrawLine(renderer , x[1] , y[1] , x[2] , y[2]) ;
```

```
SDL_SetRenderDrawColor(renderer, 128, 128, 128,
SDL_ALPHA_OPAQUE);
circleBres(x[2] , y[2] , 8) ;
```

```

SDL_SetRenderDrawColor(renderer, 255, 0, 0,
SDL_ALPHA_OPAQUE);

SDL_RenderDrawLine(renderer , x[2] , y[2] , x[3] , y[3]) ;


SDL_SetRenderDrawColor(renderer, 128, 128, 128,
SDL_ALPHA_OPAQUE);

circleBres(x[3] , y[3] , 8) ;


if (SDL_PollEvent(&event))
{

    if (event.type == SDL_QUIT)
    {
        done = SDL_TRUE;
    }

    if(event.type == SDL_MOUSEBUTTONDOWN)
    {
        /*If left mouse button down then store
        that point as control point*/
        if(event.button.button ==
SDL_BUTTON_LEFT)

        {

            if(i < 4)
            {
                printf("Control
Point(P%d):(%d,%d)\n"
                ,i,mousePosX,mousePosY) ;

```

```

        x[i] = mousePosX ;
        y[i] = mousePosY ;
        i++ ;
    }
}

if(event.type == SDL_MOUSEMOTION)
{

    xnew = event.motion.x ;
    ynew = event.motion.y ;

    int j ;

    if(flagDrawn == 1)
    {
        for(j = 0 ; j < i ; j++)
        {

            if(((float)sqrt(abs(xnew-x[j]) *
abs(xnew-x[j])
abs(ynew-y[j])) < 8.0)
+ abs(ynew-y[j]) *
abs(ynew-y[j])) < 8.0)
            {

                x[j] = xnew ;
                y[j] = ynew ;
            }
        }
    }
}

```

```

Point(P%d):(%d,%d)\n"

printf("Changed Control

,j,xnew,ynew) ;

}

}

}

mousePosX = xnew ;
mousePosY = ynew ;

}

}

SDL_RenderPresent(renderer);

}

}

if (renderer)
{
    SDL_DestroyRenderer(renderer);
}

if (window)
{
    SDL_DestroyWindow(window);
}

}

SDL_Quit();
return 0;

}

```

Output :

The screenshot shows a Linux desktop environment. The top panel displays the date and time as 'Apr 1 11:09'. The main workspace contains two windows. The left window, titled 'cg71.c', is a code editor showing C code for drawing a circle using Bresenham's algorithm. The code includes headers for `stdlib.h`, `math.h`, and `SDL2/SDL.h`. It defines variables for an SDL window, renderer, and mouse position. The `drawCircle` function uses `SDL_RenderDrawPoint` to draw pixels. The `circleBres` function implements Bresenham's algorithm to generate circle points. The right window, titled 'C04.c', is a terminal window showing the output of the program, which displays a circle drawn on a black background. The bottom panel shows the system tray with icons for various applications, including a web browser, file manager, and terminal.

```
9 #include<stdlib.h>
10 #include<math.h>
11 #include<SDL2/SDL.h>
12
13 SDL_Window* window = NULL;
14 SDL_Renderer* renderer = NULL;
15 int mouseX , mouseY ;
16 int xnew , ynew ;
17
18 /*Function to draw all other 7 pixels present at symmetric position*/
19 void drawCircle(int xc, int yc, int x, int y)
20 {
21     SDL_RenderDrawPoint(renderer,xc+x,yc+y) ;
22     SDL_RenderDrawPoint(renderer,xc-x,yc+y);
23     SDL_RenderDrawPoint(renderer,xc+x,yc-y);
24     SDL_RenderDrawPoint(renderer,xc-x,yc-y);
25     SDL_RenderDrawPoint(renderer,xc+y,yc+x);
26     SDL_RenderDrawPoint(renderer,xc-y,yc+x);
27     SDL_RenderDrawPoint(renderer,xc+y,yc-x);
28     SDL_RenderDrawPoint(renderer,xc-y,yc-x);
29 }
30
31 /*Function for circle-generation using Bresenham's algorithm */
32 void circleBres(int xc, int yc, int r)
33 {
34     int x = 0, y = r;
35     int d = 3 - 2 * r;
36     while (y >= x)
37     {
38         /*for each pixel we will draw all eight pixels */
39         drawCircle(xc, yc, x, y);
```