**Open-Ended Program**

**Name: Kuldeep Dwivedi**

**Enrollment No.: A2305218477**

**Subject: Computer Graphics**

**Section:** **6CSE7-X/Y**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Category**  **of Assignment** | **Code** | **Name of Experiment** | **Date of Allotment of experiment** | **Date of Evaluation** | **Max.**  **Marks** | **Marks obtained** | **Sign.**  **of Faculty** |
| **Design Based Open Ended experiment** | **PR (10)** | Polygon is an ordered list of vertices. For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon. Flood fill is an algorithm mainly used to determine a bounded area connected to a given node in a multi-dimensional array. It is a close resemblance to the bucket tool in paint programs. Boundary fill is the algorithm used where we have to do an interactive painting in computer graphics, where interior points are easily selected. If we have a specified boundary in a single color, then the fill algorithm proceeds pixel by pixel until the boundary color is encountered. Draw Geometric primitives using basic scan conversion algorithms to fill polygon using a. flood fill algorithm b. boundary algorithm. You may take polygon as per your choice. |  |  | **10** |  |  |

**OPEN ENDED EXPERIMENT**

**Date: -**

**Objective: -**. Polygon is an ordered list of vertices. For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon. Flood fill is an algorithm mainly used to determine a bounded area connected to a given node in a multi-dimensional array. It is a close resemblance to the bucket tool in paint programs. Boundary fill is the algorithm used where we have to do an interactive painting in computer graphics, where interior points are easily selected. If we have a specified boundary in a single color, then the fill algorithm proceeds pixel by pixel until the boundary color is encountered. Draw Geometric primitives using basic scan conversion algorithms to fill polygon using a. flood fill algorithm b. boundary algorithm. You may take polygon as per your choice.

**Software Used: -**  Code Blocks.

**Theory: -**

**Variable Description:-**

* (x1,y1) is the co-ordinate of the center of the circle.
* 'r' is the radius.
* 'x' is the abcissa of the starting point from where plotting is to begin.
* 'y' is the ordinate of the starting point from where plotting is to begin.
* 'd' is the decision parameter.
* 'speed' is basically the time(in milliseconds) for which a delay will be applied.
* 'color' stores the color of the current pixel.
* 'boundCol' is the color of the boundary of the circle.
* 'fillCol' is the color with which the circle is to be painted (different for both algorithms).
* 'oriCol' is the color with which the circle is to be painted ('fillColor' of boundary fill algorithm).

Note: Variables 'x1','y1','r' and 'speed' are initialized from before in this piece of code. To avoid any loss of generality, take user-inputs.

**Functions used:-**

**Pre-defined functions:**

**initwindow()** - to create the displaying screen.

**setcolor()-** to color everything drawn after that that segment till the same function has been invoked with a different color.

**delay()** - to hold the current state for a particular time

**putpixel()-** to color the pixel (x,y) with a given color, say C (passed as an integer).

**getpixel()-** to get the color of a pixel (x,y).

**User-defined functions:**

**drawQuadrants()** - to draw four quadrants of the window.

**circleMidPointApp()** - o draw a circle through Mid-Point Approach.

**paint()** - plot 8 different points of a circle at once at a time

**boundaryFill()-** to color the circle applying 'boundary fill' technique

**floodFill()-** to color the circle applying 'flood fill' technique

**Logic Description:-**

For the first quadrant start form (0,r), increments 'x' by '1' and find corresponding 'y' and plot the (x,y) pixel.

Repeat the process till 'x' and 'y' are unequal. For each quadrant find the reflecting points of the 8 octants and plot them too.

**Algorithm (Mid Point):**

di+1=di+2(xi+1)+1, if di<0, E(xi+1,yi) (Eastern pixel)

di+1=di+2(xi+1)+1-2(yi-1), if di>=0, SE(xi+1,yi-1) (South-Eastern pixel)

Initially:

d0=5/4-r = 1-r (appx).

**Algorithm (Boundary Fill):**

Start traversing from centre (x1,y1) horizontally and go on plotting pixels with the 'fillColor' untill a

pixel is reached which is already painted with the 'boundCol'. Do it recursively using either 4-nbd or 8-nbd of (x1,y1).

**Algorithm (Flood Fill):**

Start traversing from centre (x1,y1) horizontally and go on plotting pixels with the 'fillColor' untill a

pixel is reached which is already painted with the 'boundCol'. Do it recursively using either 4-nbd or 8-nbd of (x1,y1).

**Code:**

**#include<iostream>**

**#include<graphics.h>**

**#include<conio.h>**

// pre-processor directories to calculate co-ordinates for the displaying device

**#define fx(x) (getmaxx()/2+x)**

**#define fy(y) (getmaxy()/2-y)**

// required constants

**#define X 640**

**#define Y 480**

// colors

**#define LWHITE 7** // boundary color of the circle

**#define LCYAN 3** // color of the Co-ordinate axe

**#define IMAGENTA 13** // boundary fill color

**#define YELLOW 14** // flood fill color

**using namespace std;**

**void drawQuadrants();** // function prototype to create quadrants

**void circleMidPointApp(int,int,int);** // function prototype to draw acircle through Mid-Point Approach

**void paint(int,int,int,int);** // function prototype to plot 8 different points of a circle at once at a time

**void boundaryFill(int,int,int,int);** // function prototype to color the circle applying 'boundary fill' technique

**void floodFill(int,int,int,int);** // function prototype to color the circle applying 'flood fill' technique

**int main(void)**

**{**

**// declaring and initializing variables**

**int x1=56,y1=-67,r=100;**

**initwindow(X,Y);** // creating window

**drawQuadrants();** // calling function to draw quadrants

**circleMidPointApp(x1,y1,r);** // function call to draw a circle through Mid-Point Approach

**boundaryFill(x1,y1,LWHITE,IMAGENTA);** // function call to color the circle using Boundary-Fill Technique

**delay(100);** // holding the present state for '100' milliseconds

**floodFill(x1,y1,IMAGENTA,YELLOW**); // function call to color the circle Flood-Fill Technique

**getch();**

**return 0;**

**}** // end of main

// function definition to create quadrants

**void drawQuadrants()**

**{**

**setcolor(LCYAN);** // setting a color for the co-ordinate axes

**line(0,fy(0),X,fy(0));** // drawing the X-axis; fy(0)=480/2=240;

**line(fx(0),0,fx(0),Y);** // drawing the Y-axis; fx(0)=640/2=320;

**}**

// function definition to draw a circle through Mid-Point Approach

**void circleMidPointApp(int x1,int y1,int r)**

**{**

// declaring and initializing variables

**int x=0,y,speed=50,d;**

**y=r;**

**d=1-r;**

// applying Mid-point algorithm

**while(x<=y)**

**{**

**paint(x1,y1,x,y);** // function call to plot 8 different points of a circle at once at a time

**if(d<0)** // if decision variable is negative, E corner

**d=d+(2\*x)+3;**

**else**  // if decision variable is non-negative, SE corner

**{**

**d=d+(2\*(x-y))+5;**

**y=y-1;**

**}**

**x=x+1;**

**delay(speed);**  // holding the present state for 'speed' milliseconds

**}**

**}**

// function definition to plot 8 different points of a circle at once at a time

**void paint(int x1,int y1,int x,int y)**

**{**

// first quadrant

**putpixel(fx(x1+x),fy(y1-y),LWHITE);**  // starting from (0,r) and drawing the half semi-circle, left to right

**putpixel(fx(x1+y),fy(y1-x),LWHITE);** // starting from (r,0) and drawing the half semi-circle, right to left

**// second quadrant**

**putpixel(fx(x1-x),fy(y1-y),LWHITE);** // starting from (0,r) and drawing the half semi-circle, right to left

**putpixel(fx(x1-y),fy(y1-x),LWHITE);** // starting from (-r,0) and drawing the half semi-circle, left to right

// third quadrant

**putpixel(fx(x1-x),fy(y1+y),LWHITE);** // starting from (0,-r) and drawing the half semi-circle, right to left

**putpixel(fx(x1-y),fy(y1+x),LWHITE);** // starting from (-r,0) and drawing the half semi-circle, left to right

// fourth quadrant

**putpixel(fx(x1+y),fy(y1+x),LWHITE);** // starting from (r,0) and drawing the half semi-circle, right to left

**putpixel(fx(x1+x),fy(y1+y),LWHITE);** // starting from (0,-r) and drawing the half semi-circle, left to right

**}**

// function prototype to color the circle applying 'boundary fill' technique

**void boundaryFill(int x,int y,int boundCol,int fillCol)**

**{**

**int color;** // declaring variable

**color=getpixel(fx(x),fy(y));** // get the color if the current pixel

**if(color!=boundCol && color!=fillCol)**

**{**

**putpixel(fx(x),fy(y),fillCol);**

// right semi-circle

**boundaryFill(x,y+1,boundCol,fillCol);**

**boundaryFill(x,y-1,boundCol,fillCol);**

// right semi-circle

**boundaryFill(x+1,y,boundCol,fillCol);**

**boundaryFill(x-1,y,boundCol,fillCol);**

**}**

**}**

// function prototype to color the circle applying 'flood fill' technique

**void floodFill(int x,int y,int oriCol,int fillCol)**

**{**

**int color;** // declaring variable

**color=getpixel(fx(x),fy(y));** // get the color if the current pixel

**if(color==oriCol)**

**{**

**putpixel(fx(x),fy(y),fillCol);**

**/**/ right semi-circle

**floodFill(x,y+1,oriCol,fillCol);**

**floodFill(x,y-1,oriCol,fillCol);**

// left semi-circle

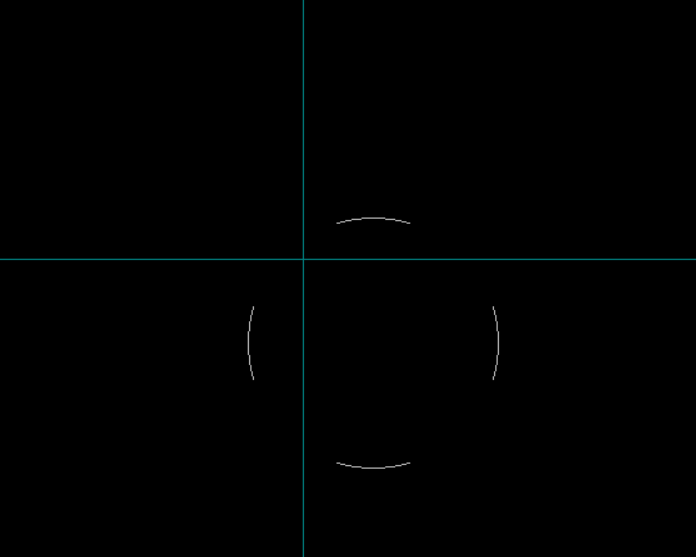
**floodFill(x+1,y,oriCol,fillCol);**

**floodFill(x-1,y,oriCol,fillCol);**

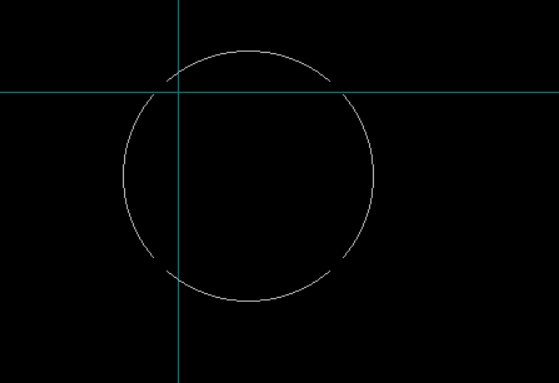
**}**

**}**

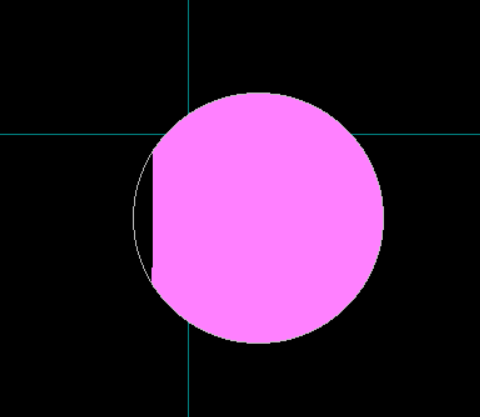
**Output :**

****

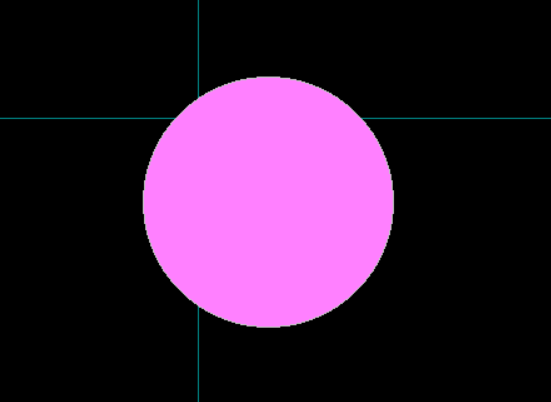
**Fig: Circle Formation**

****

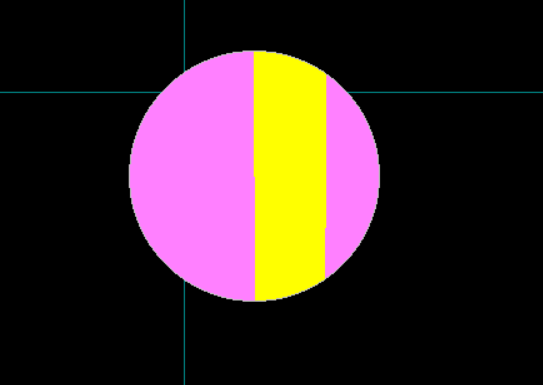
**Fig: Complete Circle**

****

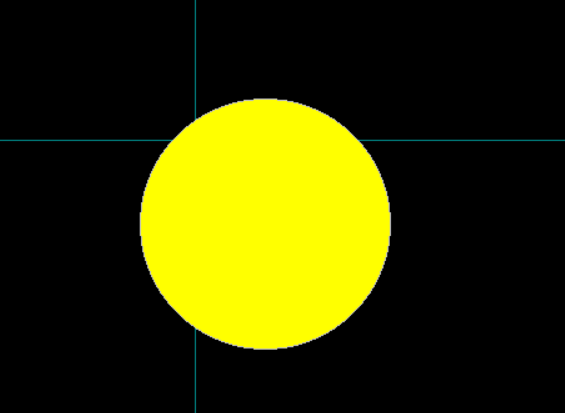
**Fig : Filling of color using Boundary Fill Algo.**

****

**Fig : Boundary Fill implemented.**

****

**Fig : Filling of (yellow) color using Flood Fill Algo.**

****

**Fig : Implementation of Flood Fill Algo.**

**Learning Outcome:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Design Based Experiment) sheet for Lab Experinemt Department of Computer Science & Engineering Amity University, Noida (UP)** | | | |
| Programme | B. Tech CSE | Course Name | Computer Graphics |
| Course Code | [CSE203] | Semester | 6 |
| Student Name | XYZ | Enrollment No. | A23000000 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Designing Concept (D) | 3 |  |  |
| Application of Knowledge (E) | 2 |  |  |
| Performance (F) | 3 |  |  |
| Result (G) | 2 |  |  |
| Total | 10 |  |  |