ASSIGNMENT NO 4

Name : Shashank Kapadnis

Batch: E11

Roll No: 23310

Title :

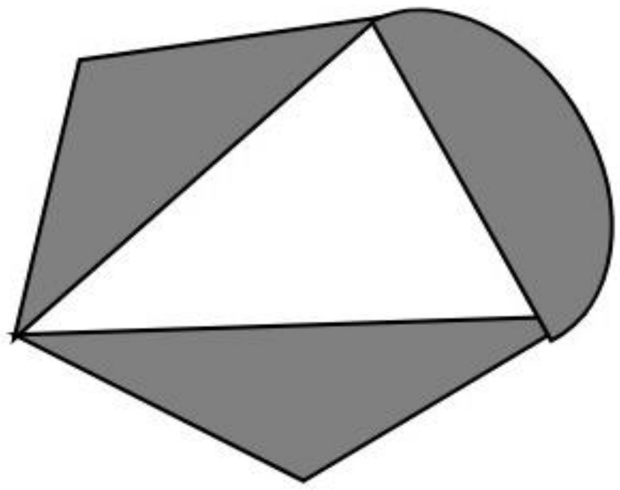
Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming

Theory :

Flood Fill / Seed Fill Algorithm

In this method, a point or seed which is inside region is selected. This point is called a seed point. Then four connected approaches or eight connected approaches is used to fill with specified color.

The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color we use this algorithm.



In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

Algorithm:

```
1. Procedure floodfill (x, y,fill_ color, old_color: integer)
2. If (getpixel (x, y)=old_color)
3. {
4. setpixel (x, y, fill_color);
5. fill (x+1, y, fill_color, old_color);
6. fill (x-1, y, fill_color, old_color);
7. fill (x, y+1, fill_color, old_color);
8. fill (x, y-1, fill_color, old_color);
9. }
10. }
```

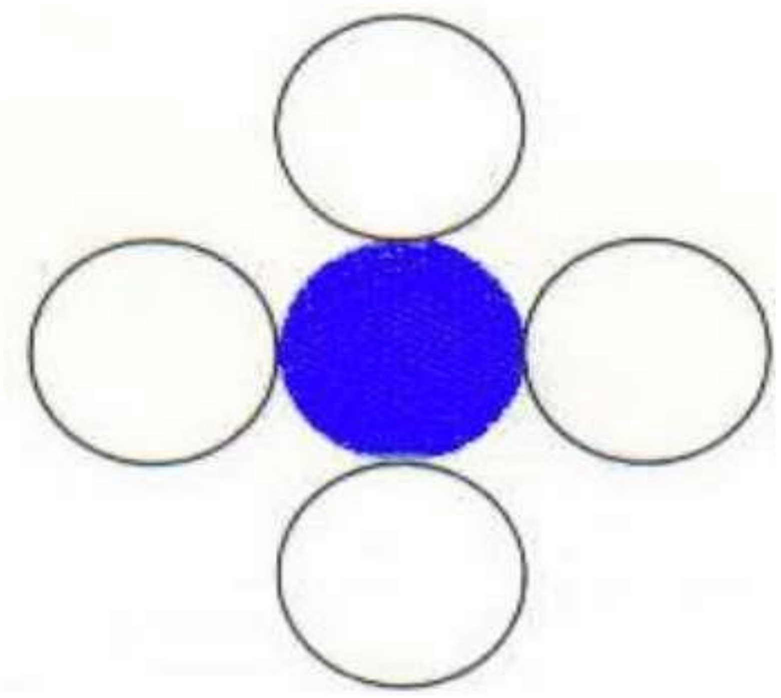
Boundary Fill Algorithm

The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work.

In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-Connected Polygon

In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.



Algorithm:

Step 1 − Initialize the value of seed point seedx, seedyseedx, seedy, fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

If getpixel(x, y) = dcol then repeat step 4 and 5

Step 4 − Change the default color with the fill color at the seed point.

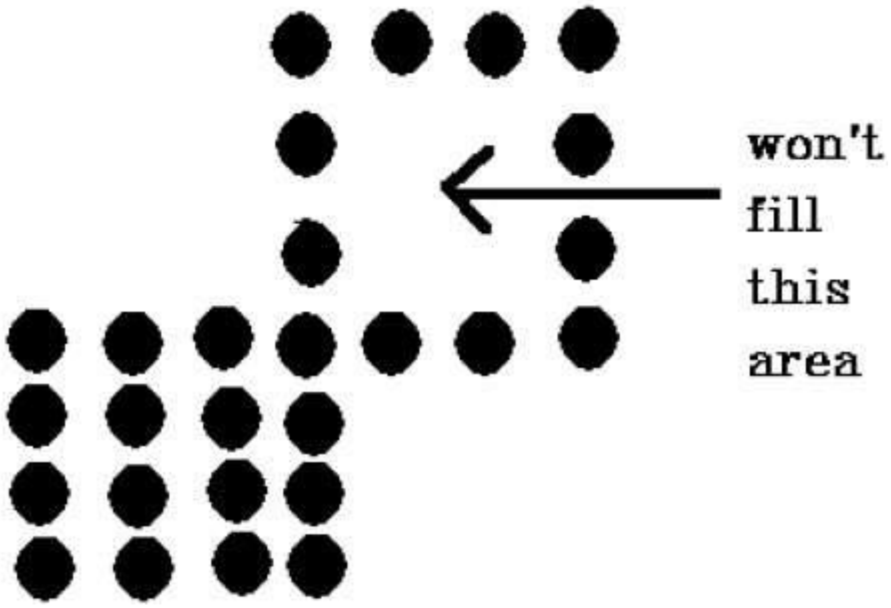
```
setPixel(seedx, seedy, fcol)
```

Step 5 − Recursively follow the procedure with four neighborhood points.

```
FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
```

Step 6 – Exit

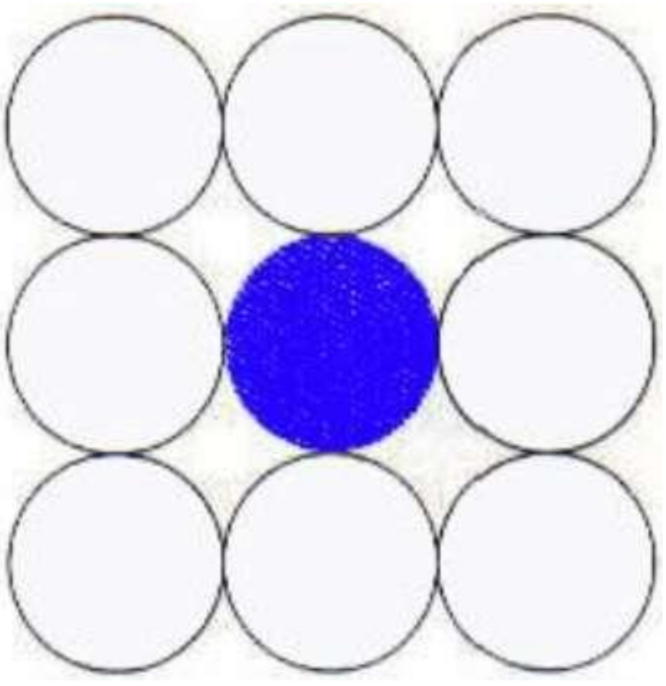
There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.



8-Connected Polygon

In this technique 8-connected pixels are used as shown in the figure. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique.

In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.



Algorithm:

Step 1 − Initialize the value of seed point seedx, seedyseedx, seedy, fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color then repeat the steps 4 and 5 till the boundary pixels reached

If getpixel(x,y) = dcol then repeat step 4 and 5

Step 4 − Change the default color with the fill color at the seed point.

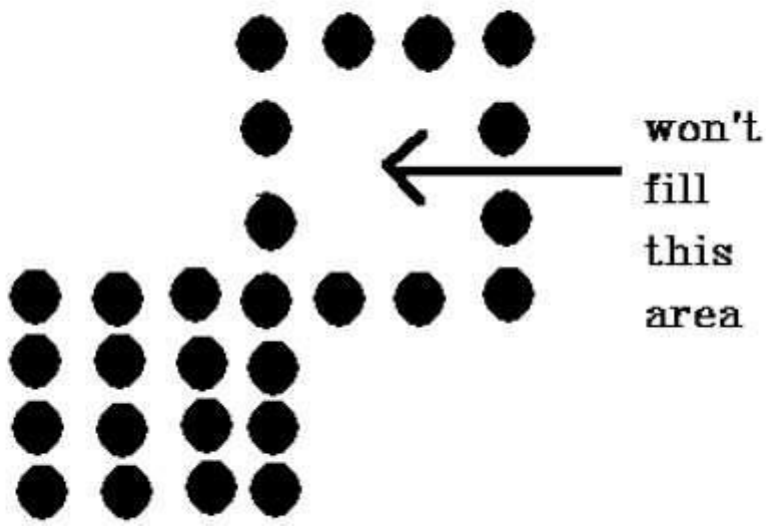
setPixel(seedx, seedy, fcol)

Step 5 − Recursively follow the procedure with four neighbourhood points

FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx, seedy + 1, fcol, dcol)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy - 1, fcol, dcol)
FloodFill (seedx – 1, seedy - 1, fcol, dcol)

Step 6 − Exit

The 4-connected pixel technique failed to fill the area as marked in the following figure which won't happen with the 8-connected technique.

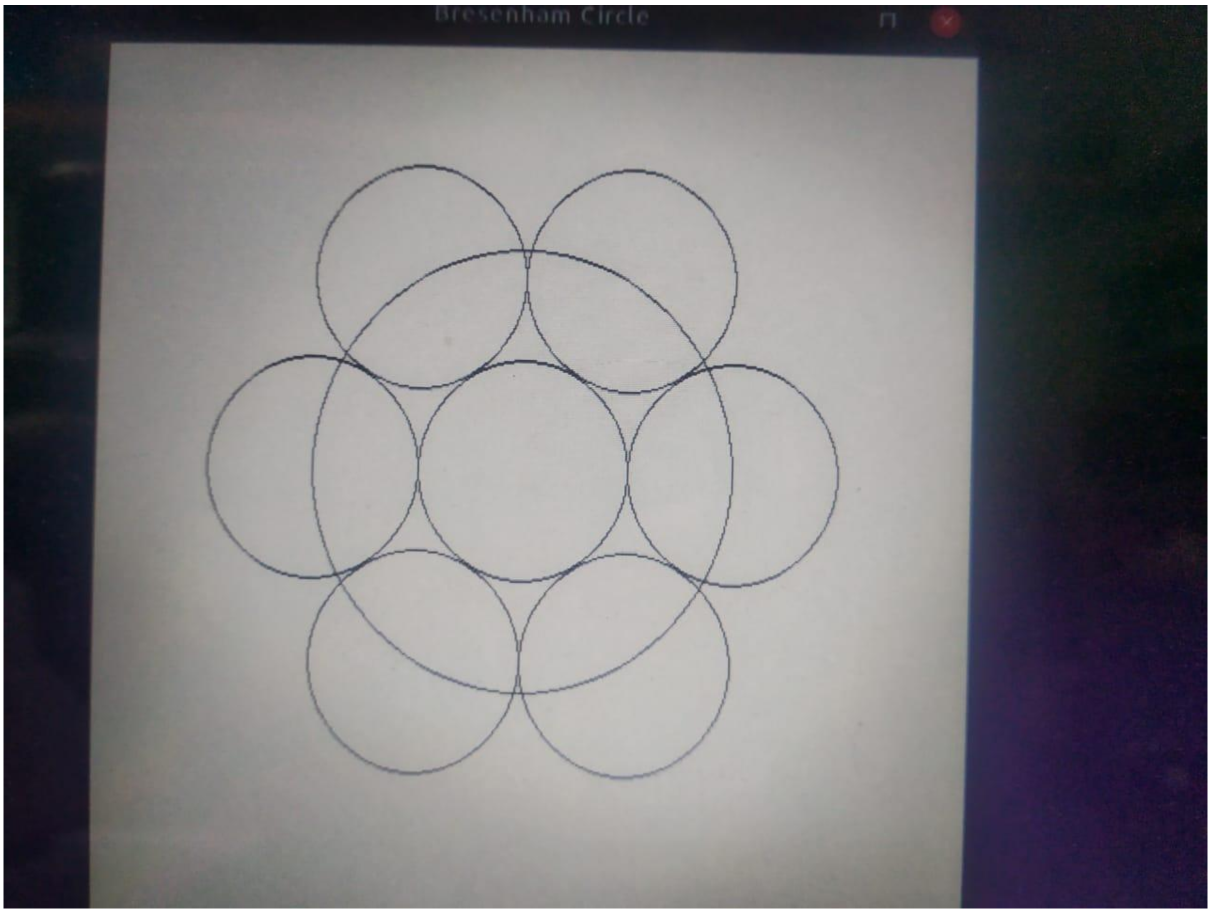


Code:

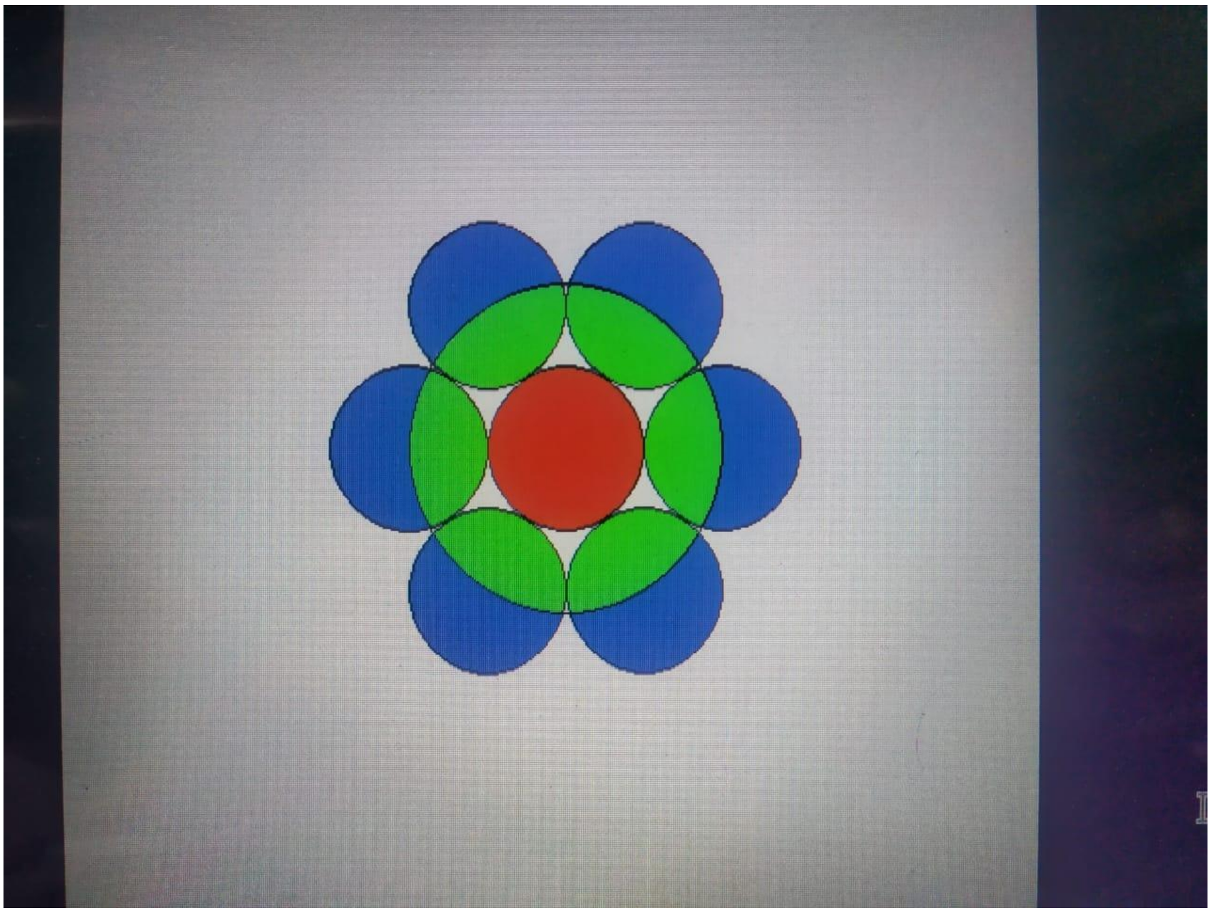
```
#include <GL/glut.h>

#include <iostream>

#include <vector>

#include <math.h>

using namespace std;
```

```
struct Color
{
 float r;
 float g;
 float b;
};
struct Circle
{
 int xc, yc, r;
};
struct Pattern
{ int patternType, x0, y0, r0;
 vector<Circle> circs;
};
//
bool drawMode = false;
bool colorMode = false;
int patternNumber = 2;
int patternXc = 250;
int patternYc = 250;
int radius = 25;
Color bkg = {1.0, 1.0, 1.0};
Color bd = {0.0, 0.0, 0.0};
Color red = {1.0, 0.0, 0.0};
Color green = {0.0, 1.0, 0.0};
Color blue = {0.0, 0.0, 1.0};
Color fillColor = red;
char algo = 'f'; // f or b are two options
//void init()
{
 glClearColor(1.0, 1.0, 1.0, 0.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glMatrixMode(GL_PROJECTION);
 gluOrtho2D(0.0, 500.0, 0.0, 500.0);
}
Color getPixel(int x, int y)
{
 Color color;
 glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT,
&color);
 return color;
}
void setPixel(int x, int y)
{
 glColor3f(fillColor.r, fillColor.g, fillColor.b);
 glBegin(GL_POINTS);
 glVertex2i(x, y); glEnd();
 glFlush();
}
void boundaryFill(int x, int y)
{
 float r = getPixel(x, y).r;
 float g = getPixel(x, y).g;
 float b = getPixel(x, y).b;
 if ((r != bd.r || g != bd.g || b != bd.b) && (r !=
fillColor.r || g != fillColor.g || b != fillColor.b))
 {
 setPixel(x, y);
 boundaryFill(x, y + 1);
 boundaryFill(x - 1, y);
 boundaryFill(x + 1, y);
 boundaryFill(x, y - 1);
 }
}
void floodFill(int x, int y)
{ float r = getPixel(x, y).r;
 float g = getPixel(x, y).g;
 float b = getPixel(x, y).b;
 if (r == bkg.r && g == bkg.g && b == bkg.b)
 {
 setPixel(x, y);
 floodFill(x, y + 1);
 floodFill(x - 1, y);
 floodFill(x + 1, y);
 floodFill(x, y - 1);
 }
}
void drawPoints(int x, int y, int xc, int yc)
{
 x += xc;
 y += yc;
 glColor3f(0.0, 0.0, 0.0);
 glBegin(GL_POINTS);
 glVertex2i(x, y);
 glEnd();
 glFlush();}
void drawCircle(Circle c)
{
 int x = 0;
 int y = c.r;
 int d = 3 - (2 * c.r);
 while (x <= y)
 {
 drawPoints(x, y, c.xc, c.yc);
 drawPoints(-x, y, c.xc, c.yc);
 drawPoints(x, -y, c.xc, c.yc);
 drawPoints(-x, -y, c.xc, c.yc);
 drawPoints(y, x, c.xc, c.yc);
 drawPoints(-y, x, c.xc, c.yc);
 drawPoints(y, -x, c.xc, c.yc);
 drawPoints(-y, -x, c.xc, c.yc);
 x++;
 if (d <= 0)
 { d += (4 * x) + 6;
 }
 else
 {
 d += 4 * (x - y) + 10;
 y--;
 }
 }
}
void generatePattern()
{
 Pattern temp;
 if (patternNumber == 1)
 {
 cout << "The chosen type is of patterType 1" <<
endl;
 drawCircle({patternXc, patternYc, radius});
 drawCircle({(patternXc + radius),
(int)round(patternYc + 1.73 * radius), radius});
 drawCircle({patternXc + 2 * radius, patternYc,
radius});
 drawCircle({patternXc + radius,
(int)round(patternYc - 1.73 * radius), radius}); drawCircle({patternXc - radius,
(int)round(patternYc
- 1.73 * radius), radius});
 drawCircle({patternXc - 2 * radius, patternYc,
radius});
 drawCircle({patternXc - radius, (int)round(patternYc
+ 1.73 * radius), radius});
 drawCircle({patternXc, patternYc, 2 * radius});
 }
 else if (patternNumber == 3)
 {
 cout << "The chosen type is of patterType 2" <<
endl;
 drawCircle({patternXc, patternYc, radius});
 drawCircle({(patternXc + radius),
(int)round(patternYc + 1.73 * radius), radius});
 drawCircle({patternXc + 2 * radius, patternYc,
radius});
 drawCircle({patternXc + radius,
(int)round(patternYc - 1.73 * radius), radius});
 drawCircle({patternXc - radius, (int)round(patternYc
- 1.73 * radius), radius});
 drawCircle({patternXc - 2 * radius, patternYc,
radius}); drawCircle({patternXc - radius, (int)round(patternYc
+ 1.73 * radius), radius});
 drawCircle({patternXc, patternYc, 2 * radius});
 drawCircle({patternXc, patternYc, 3 * radius});
 }
 else if (patternNumber == 2)
 {
 cout << "The chosen type is of patternType 3" <<
endl;
 drawCircle({patternXc, patternYc, radius});
 drawCircle({patternXc, patternYc + radius, radius});
 drawCircle({(int)round(patternXc + 0.7 * radius),
(int)round(patternYc + 0.7 * radius), radius});
 drawCircle({patternXc + radius, patternYc, radius});
 drawCircle({(int)round(patternXc + 0.7 * radius),
(int)round(patternYc - 0.7 * radius), radius});
 drawCircle({patternXc, patternYc - radius, radius});
 drawCircle({(int)round(patternXc - 0.7 * radius),
(int)round(patternYc - 0.7 * radius), radius});
 drawCircle({patternXc - radius, patternYc, radius});
 drawCircle({(int)round(patternXc - 0.7 * radius),
(int)round(patternYc + 0.7 * radius), radius});
 drawCircle({patternXc, patternYc, 2 * radius});
 }}
void display()
{
}
void myMouseFunc(int button, int state, int x, int y)
{
 if (button == GLUT_LEFT_BUTTON && state ==
GLUT_DOWN)
 {
 if (drawMode)
 {
 patternXc = x;
 patternYc = 500 - y;
 generatePattern();
 }
 else if (colorMode)
 {
 if (algo == 'f')
 {
 floodFill(x, 500 - y);
 } else
 {
 boundaryFill(x, 500 - y);
 }
 }
 }
}
void myMenu(int value)
{
 glutIdleFunc(display);
 if (drawMode)
 {
 if (value == 1)
 {
 patternNumber = 1;
 cout << "Pattern 1 is selected" << endl;
 cout << "now enter the radius(a=5, b=10, c=15
and so on)" << endl;
 }
 else if (value == 2)
 { patternNumber = 2;
 cout << "Pattern 2 is selected" << endl;
 cout << "now enter the radius(a=5, b=10, c=15
and so on)" << endl;
 }
 else if (value == 3)
 {
 patternNumber = 3;
 cout << "Pattern 3 is selected" << endl;
 cout << "now enter the radius(a=5, b=10, c=15
and so on)" << endl;
 }
 }
 else if (colorMode)
 {
 if (value == 4)
 {
 fillColor = red;
 cout << "Red color is selected, select algo if not
selected yet" << endl;
 }
 else if (value == 5)
 {
 fillColor = green; cout << "Green color is selected, select algo if not
selected yet" << endl;
 }
 else if (value == 6)
 {
 fillColor = blue;
 cout << "Blue color is selected, select algo if not
selected yet" << endl;
 }
 else if (value == 7)
 {
 algo = 'f';
 cout << "flood fill algorithm selected, select color
if not selected yet" << endl;
 }
 else if (value == 8)
 {
 algo = 'b';
 cout << "boundary fill algorithm selected, select
color if not selecte yet" << endl;
 }
 }
}void myKeyboard(unsigned char key, int x, int y)
{
 if ((int)key == 48)
 {
 drawMode = true;
 colorMode = false;
 cout << "Draw-mode selected.. " << endl;
 cout << "Right click and select the pattern type by
again right clicking." << endl;
 }
 else if ((int)key == 49)
 {
 drawMode = false;
 colorMode = true;
 cout << "color-Mode selected.." << endl;
 cout << "Right click and select the color and
algorithm to fill" << endl;
 }
 else
 {
 radius = ((int)key - 96) * 5;
 cout << "radius is equal to " << radius << endl;
 cout << "click the screen to select the central point
for the pattern: " << endl; }
}
int main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitWindowPosition(500, 500);
 glutInitWindowSize(500, 500);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutCreateWindow("Bresenham Circle");
 cout << "1. Draw mode (click 0)" << endl;
 cout << "2. Color mode (click 1)" << endl;
 init();
 glutMouseFunc(myMouseFunc);
 int subMenuId01 = glutCreateMenu(myMenu);
 glutAddMenuEntry("Pattern 1", 1);
 glutAddMenuEntry("Pattern 2", 2);
 glutAddMenuEntry("Pattern 3", 3);
 int subMenuId02 = glutCreateMenu(myMenu);
 glutAddMenuEntry("Red", 4);
 glutAddMenuEntry("Green", 5); glutAddMenuEntry("Blue", 6);
 int subMenuId03 = glutCreateMenu(myMenu);
 glutAddMenuEntry("Flood-Fill", 7);
 glutAddMenuEntry("Boundary-Fill", 8);
 glutCreateMenu(myMenu);
 glutAddSubMenu("Pattern", subMenuId01);
 glutAddSubMenu("Color", subMenuId02);
 glutAddSubMenu("Algorithm", subMenuId03);
 glutAttachMenu(GLUT_RIGHT_BUTTON);
 glutKeyboardFunc(myKeyboard);
 glutDisplayFunc(display);
 glutMainLoop();
}
```

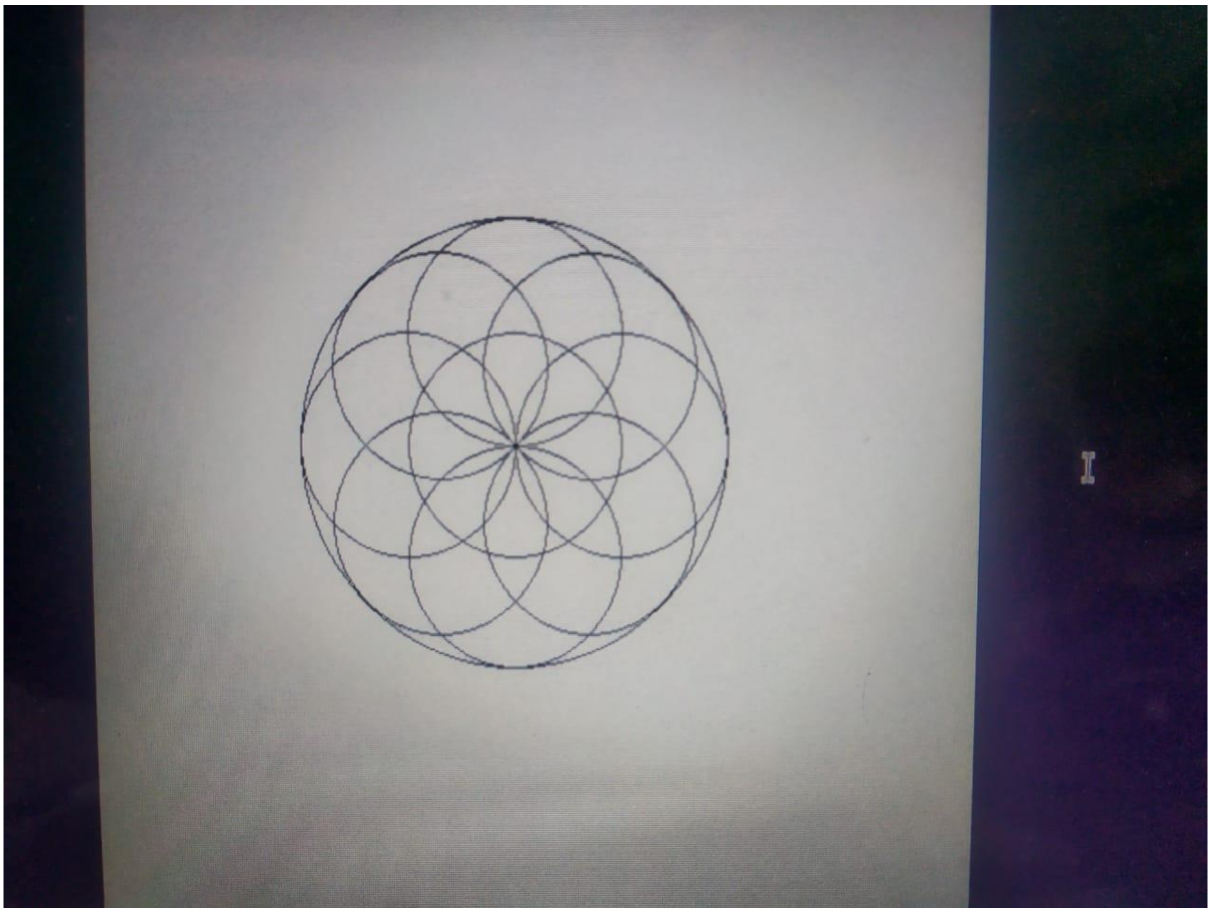
Output:



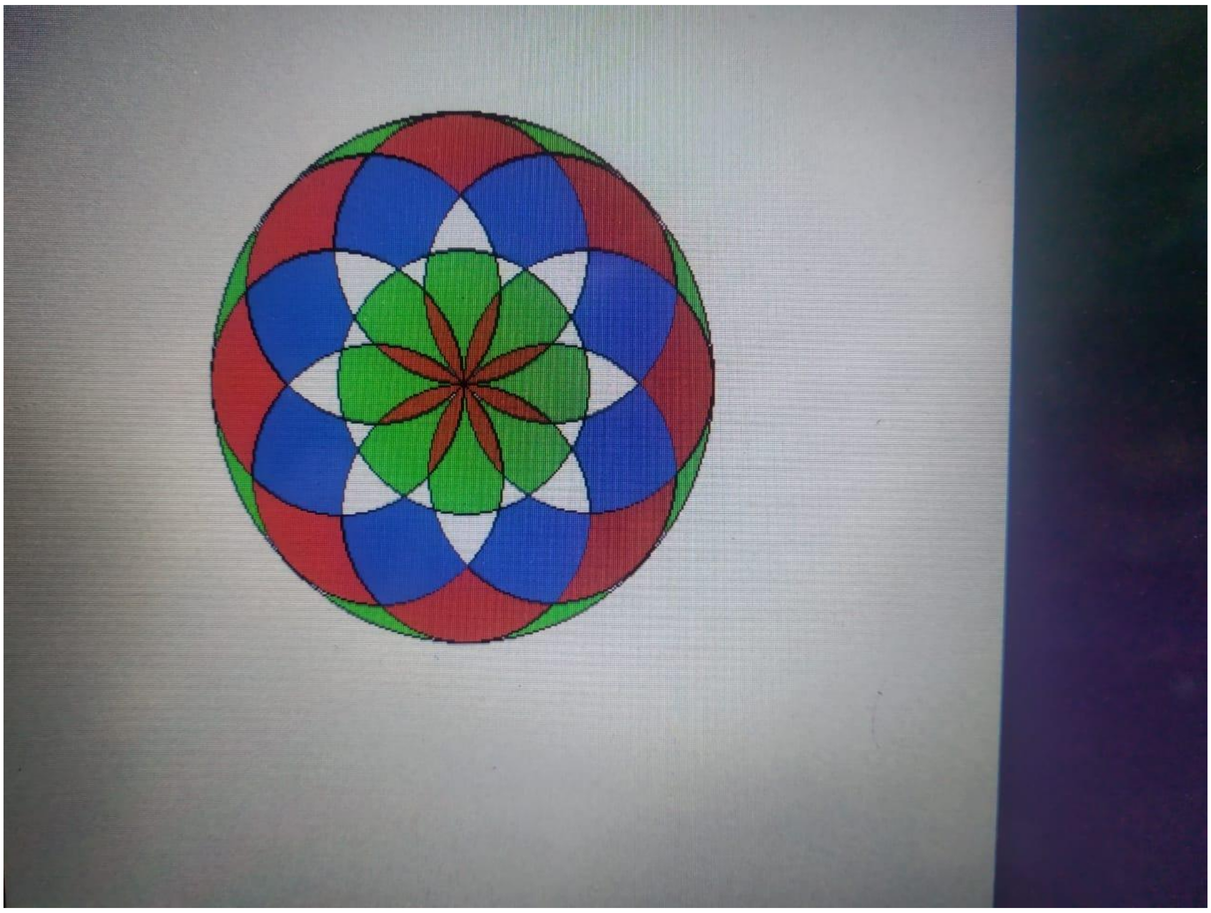
Pattern 1 uncoloured



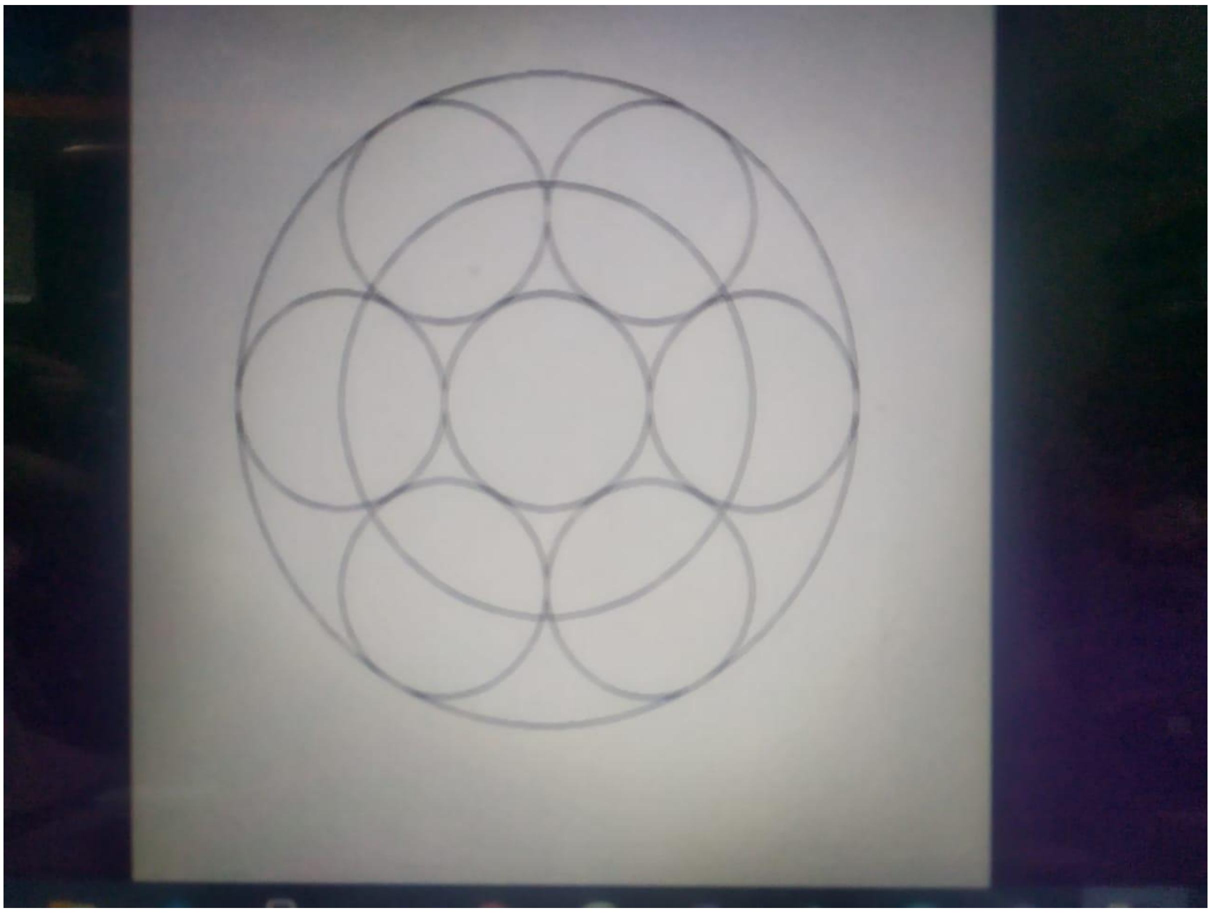
Pattern 1 coloured by boundary fill algorithm



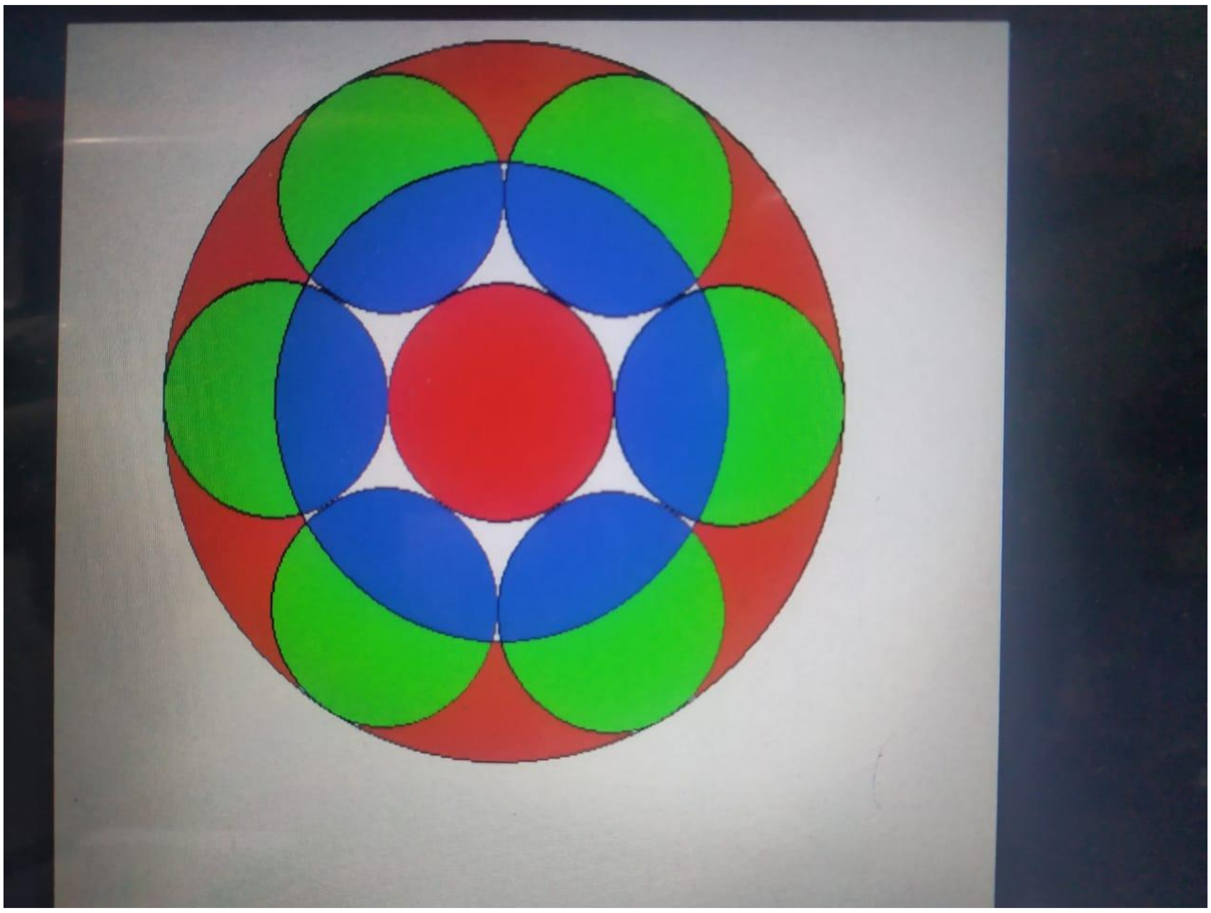
Pattern 2 uncoloured



Pattern 2 coloured by flood fill algorithm



Pattern 3 uncoloured



Pattern 3 coloured using flood fill and boundary fill algorithms

Conclusion :

Advantages Flood Fill:

- Flood fill colors an entire area in an enclosed figure through interconnected pixels using a single color.
- It is an easy way to fill color in the graphics. One just takes the shape and starts flood fill.
- The algorithm works in a manner so as to give all the pixels inside the boundary the same color leaving the boundary and the pixels outside.
- Flood Fill is also sometimes referred to as Seed Fill as you plant a seed and more and more seeds are planted by the algorithm. Each seed takes the responsibility of giving the same color to the pixel at which it is positioned.

Disadvantages of Flood Fill:

- Very slow algorithm
- May be fail for large polygons
- Initial pixel required more knowledge about surrounding pixels.

Disadvantages of Boundary-Fill over Flood-Fill:

- In boundary-fill algorithms each pixel must be compared against both the new colour and the boundary colour. In flood-fill algorithms each pixel need only be compared against the new colour. Therefore flood-fill algorithms are slightly faster.

- Boundary-fill algorithms can leak. There can be no leakage in flood-fill algorithms.

Advantages of Boundary-Fill over Flood-Fill:

- Flood-fill regions are defined by the whole of the region. All pixels in the region must be made the same colour when the region is being created. The region cannot be translated, scaled or rotated.
- 4-connected boundary-fill regions can be defined by lines and arcs. By translating the line and arc endpoints we can translate, scale and rotate the whole boundary-fill region. Therefore 4-connected boundary-fill regions are better suited to modelling.