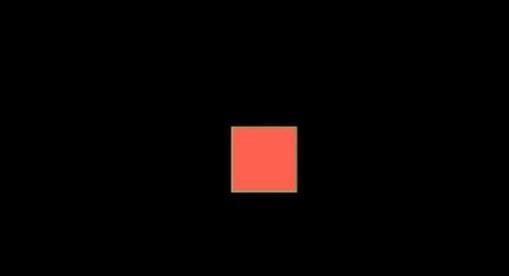
|  |  |
| --- | --- |
| **Name:** | Yash Ravindra Kerkar |
| **Roll No:** |  |
| **Class/Sem:** | SE/III |
| **Experiment No.:** | 5 |
| **Title:** | Boundary Fill and Flood Fill Polygon filling Algorithm |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Marks:** |  |
| **Sign of Faculty:** |  |

# Experiment No. 5

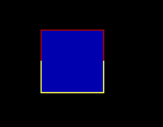
|  |  |
| --- | --- |
| **Aim** | Write a program to implement Boundary Fill and Flood Fill Polygon filling Algorithm in C. |
| **Objective** | To implement boundary fill and flood fill algorithm |
| **Theory** | **Boundary Fill Algorithm:**  This algorithm uses the recursive method. First of all, a starting pixel called the seed is considered. The algorithm checks whether boundary pixels or adjacent pixels are colored or not. If the adjacent pixel is already filled or colored then leave it, otherwise fill it. The filling is done using four connected or eight connected approaches.     1. **Four connected approaches:** In this approach, left, right, above, below pixels are tested. 2. **Eight connected approaches:** In this approach, left, right, above, below and four diagonals are selected.   Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom. The algorithm takes time and memory because some recursive calls are needed.  **Flood 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** | **Boundary Fill**  boundary\_fill\_8(x, y, fill\_color, boundary\_color) int current;  current=getpixel (x, y);  if ( ( current != fill\_color ) && ( current != boundary\_color) )  {  setpixel (x, y, fill\_color )  boundary\_fill\_8(x+1, y, fill\_color , boundary\_color); boundary\_fill\_8(x-1, y, fill\_color , boundary\_color); boundary\_fill\_8(x, y+1, fill\_color , boundary\_color); boundary\_fill\_8(x, y-1, fill\_color , boundary\_color); boundary\_fill\_8(x+1, y+1, fill\_color , boundary\_color); boundary\_fill\_8(x+1, y-1, fill\_color , boundary\_color); boundary\_fill\_8(x-1, y+1, fill\_color , boundary\_color); boundary\_fill\_8(x-1, y-1, fill\_color , boundary\_color);  }  **Flood Fill**  flood\_fill\_8(x, y, fill\_color, default\_color)  {  int current; current=getpixel (x, y);  if ( current == default\_color)  {  setpixel (x, y, fill\_color)  flood\_fill\_8(x+1, y, fill\_color , default\_color); flood\_fill\_8(x-1, y, fill\_color , default\_color); flood\_fill\_8(x, y+1, fill\_color , default\_color); flood\_fill\_8(x, y-1, fill\_color , default\_color); flood\_fill\_8(x+1, y+1, fill\_color , default\_color); flood\_fill\_8(x-1, y+1, fill\_color , default\_color); flood\_fill\_8(x+1, y-1, fill\_color , default\_color); flood\_fill\_8(x-1, y-1, fill\_color , default\_color);  }  }  **Code:**  #include <graphics.h>  #include <conio.h>  void boundaryFill(int x, int y, int fill\_color, int boundary\_color) {  if (getpixel(x, y) != boundary\_color && getpixel(x, y) != fill\_color) {  putpixel(x, y, fill\_color);  boundaryFill(x + 1, y, fill\_color, boundary\_color);  boundaryFill(x - 1, y, fill\_color, boundary\_color);  boundaryFill(x, y + 1, fill\_color, boundary\_color);  boundaryFill(x, y - 1, fill\_color, boundary\_color);  }  }  int main() {  int gdriver = DETECT, gmode;  initgraph(&gdriver, &gmode, "C:\\Turboc3\\BGI");  rectangle(50, 50, 200, 200);  boundaryFill(100, 100, 4, 15); // (100, 100) is the starting point, 4 is fill color, 15 is boundary color  getch(); // Wait for a key press  closegraph();  return 0;  } |

## Output Boundary Fill



**Flood Fill**



## Conclusion: In this program, we have learned about boundary fill algorithm, a popular algorithm used in computer graphics for filling a closed area with a specific color. Let's conclude the program step by step:

## Include Header Files: We start by including the necessary header files for graphics and input/output functions.

## Define the boundaryFill Function: This function is the heart of the program. It recursively fills a closed area with the fill\_color while respecting the boundaries of the shape defined by the boundary\_color. It checks if the current pixel is not the boundary\_color and not already filled with the fill\_color, and if so, it fills the pixel with the fill\_color and recursively calls itself for neighboring pixels to continue the filling process.

## Draw a Rectangle: We draw a rectangle on the graphics screen using the rectangle function. This rectangle serves as the boundary within which we want to perform the boundary fill.

## Call boundaryFill Function: We call the boundaryFill function, specifying the starting point (100, 100), the fill color 4, and the boundary color 15. The filling process will start from the starting point and continue until it reaches the boundaries defined by the boundary color.