Experiment – 6

Aim:

Write a program to show Working and implementation of boundary fill and flood-fill show or illustrate:

Boundary fill and flood fill are two commonly used algorithms in computer graphics for filling closed areas with a specified color.

Boundary Fill Algorithm:

The boundary fill algorithm is a recursive algorithm that is used to fill an enclosed area with a specified color. It works by starting at a given point within the enclosed area and filling each pixel that is not a boundary color with the specified fill color. The algorithm stops filling when it reaches the boundary of the enclosed area. The boundary fill algorithm assumes that the boundary of the area to be filled is already drawn with a unique color.

Flood Fill Algorithm:

The flood fill algorithm is also a recursive algorithm used to fill an enclosed area with a specified color. Unlike the boundary fill algorithm, the flood fill algorithm starts at a point outside the enclosed area and fills each pixel that is not a boundary color with the specified fill color. The algorithm stops filling when it reaches the boundary of the enclosed area. The flood fill algorithm does not require the boundary of the area to be drawn with a unique color.

Both algorithms are commonly used in computer graphics applications such as paint programs, image editing software, and game development. However, boundary fill is more commonly used in applications where the boundary of the area to be filled is already known and can be easily defined. On the other hand, flood fill is more commonly used in applications where the boundary of the area to be filled is not well defined or difficult to determine.
The boundary fill algorithm is a recursive algorithm that is used to fill an enclosed area with a specified color. Here is the algorithm for boundary fill:

Boundary fill algorithm:

```
boundaryFill(x, y, fill_color, boundary_color) {
   if(current_color != boundary_color && current_color != fill_color) {
Set pixel (x, y) to 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);
   }
}
```

where:
- `(x, y)` is the starting point for the fill.
- `fill_color` is the color to be filled.
- `boundary_color` is the color of the boundary.

- `current_color` is the color of the current pixel being processed.

The algorithm starts at a given point `(x, y)`. If the current pixel is neither the boundary color nor the fill color, it is filled with the fill color. The algorithm then recursively calls itself with the four neighboring pixels. This process continues until all the pixels in the enclosed area have been filled.

The algorithm assumes that the boundary of the area to be filled is already drawn with a unique color. If this is not the case, a separate algorithm must be used to draw the boundary.

Implementation:

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

// Define the parameters for the rectangle
int x1 = 100; int y1 = 100; int x2 = 300; int
y2 = 300;

// Define the fill color
float fill_color[] = {1.0, 0.0, 0.0};  // Red

// Define the boundary color
float boundary_color[] = {0.0, 0.0, 1.0};  // Blue

// Define the recursive boundary fill function
void boundaryFill(int x, int y, float* fill_color, float* boundary_color) {
float current_color[3];
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, current_color);

   if(current_color[0] != boundary_color[0] || current_color[1] != boundary_color[1] || current_color[2] !=
boundary_color[2]) {
      glColor3fv(fill_color);
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();        glFlush();

      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);
   }
}

// Define the display function
void display() {
   glClear(GL_COLOR_BUFFER_BIT);
glColor3fv(boundary_color);
glBegin(GL_LINE_LOOP);
glVertex2i(x1, y1);     glVertex2i(x1,
y2);
```

```
   glVertex2i(x2, y2);
glVertex2i(x2, y1);
glEnd();    glFlush();

   int x = (x1+x2)/2;
int y = (y1+y2)/2;
   boundaryFill(x, y, fill_color, boundary_color);
}

// Define the reshape function
void reshape(int w, int h) {
   glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();    gluOrtho2D(0,
w, 0, h); }

// Main function int main(int
argc, char **argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutCreateWindow("Boundary Fill Example");
   glClearColor(0.5, 0.6, 1.0, 1.0);
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();    return 0;
}
```

Flood fill algorithm:

```
floodFill(x, y, fill_color, background_color) {
if(getPixelColor(x, y) != background_color) {
      return; // exit if the starting pixel is already filled
   }
   setPixelColor(x, y, fill_color); // fill the starting pixel with the fill color
   stack.push(x, y); // add the starting pixel to the stack
while(stack is not empty) {
      current_pixel = stack.pop(); // pop the top pixel from the stack
```

```
      x = current_pixel.x;
y = current_pixel.y;
      if(getPixelColor(x+1, y) == background_color) {
setPixelColor(x+1, y, fill_color); // fill the pixel to the right
stack.push(x+1, y); // add the pixel to the right to the stack
      }
      if(getPixelColor(x-1, y) == background_color) {
setPixelColor(x-1, y, fill_color); // fill the pixel to the left
stack.push(x-1, y); // add the pixel to the left to the stack
      }
      if(getPixelColor(x, y+1) == background_color) {
setPixelColor(x, y+1, fill_color); // fill the pixel above
         stack.push(x, y+1); // add the pixel above to the stack
      }
      if(getPixelColor(x, y-1) == background_color) {
setPixelColor(x, y-1, fill_color); // fill the pixel below
         stack.push(x, y-1); // add the pixel below to the stack
      }
   }
}
```

Implementation:

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

// Define the parameters for the flood fill int
x_seed = 250;   // X-coordinate of seed point int
y_seed = 250;   // Y-coordinate of seed point
float fill_color[] = { 0.0, 1.0, 0.0 };   // Fill color

// Define the display function
void display() {
   glClear(GL_COLOR_BUFFER_BIT);
glPointSize(1.0);     glColor3f(1.0,
0.0, 0.0);
   glBegin(GL_POINTS);
   for (int i = 0; i < 500; i++) {
for (int j = 0; j < 500; j++) {
glVertex2i(i, j);
      }
   }
   glEnd();     glFlush();
glPointSize(1.0);
glColor3fv(fill_color);
   glRecti(x_seed, y_seed, x_seed+1, y_seed+1);
glFlush(); }

// Define the flood fill function
void flood_fill(int x, int y) {
float pixel_color[3];
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, pixel_color);
```

```
   if (pixel_color[0] == 1.0 && pixel_color[1] == 0.0 && pixel_color[2] == 0.0) {          glColor3fv(fill_color);
glRecti(x, y, x+1, y+1);       flood_fill(x+1, y);       flood_fill(x-1, y);       flood_fill(x, y+1);
      flood_fill(x, y-1);
   }
}

// Define the mouse function void mouse(int button, int state,
int x, int y) {     if (button == GLUT_LEFT_BUTTON && state ==
GLUT_DOWN) {       x_seed = x;       y_seed = 500 - y;
flood_fill(x_seed, y_seed);
      glutPostRedisplay();
   }
}

// Define the reshape function
void reshape(int w, int h) {
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
   gluOrtho2D(0.0, 500.0, 0.0, 500.0);
}

// Main function int main(int
argc, char **argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);    glutInitWindowSize(500, 500);
glutCreateWindow("Flood Fill Example");
   glutDisplayFunc(display);    glutMouseFunc(mouse);    glutReshapeFunc(reshape);    glutMainLoop();
return 0;
}
```