Lab IV

Course title: Computer Graphics Lab

Course code: CSE-304

3rd Year 1st Semester Examination 2022

Date of Submission: 16-07-2023

Submitted to-

Prof. Dr. Mohammad Shorif Uddin
Professor

Dr. Morium Akther
Associate Professor

Department of Computer Science and Engineering
Jahangirnagar University

Sl	Class Roll	Exam Roll	Name
01	388	202200	Md. Tanvir Hossain Saon

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka, Bangladesh

Experiment Name: Bounding and flood filling of an object.

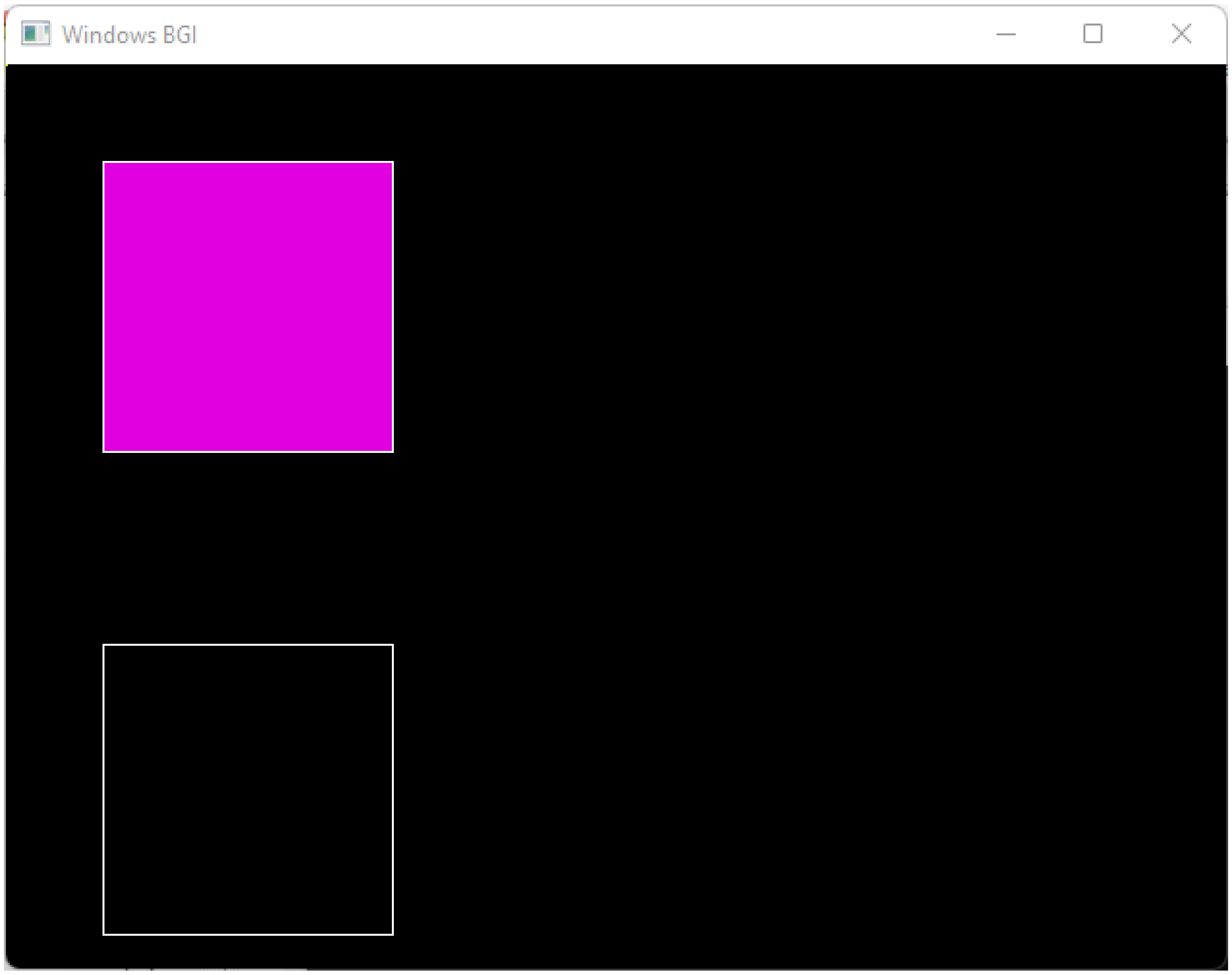
Introduction: This lab focuses on boundary and flood filling algorithms for object detection and filling in computer graphics. These algorithms are essential for identifying object boundaries and efficiently filling their interiors. The report presents the implementation, evaluation, and optimization strategies of these algorithms, aiming to enhance our understanding of computer graphics principles and their practical applications.

Source Code:

```
#include<graphics.h>

#include<dos.h>

#include<conio.h>

void boundaryFill8(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);

        boundaryFill8(x + 1, y, fill_color, boundary_color);

        boundaryFill8(x, y + 1, fill_color, boundary_color);

        boundaryFill8(x - 1, y, fill_color, boundary_color);

        boundaryFill8(x, y - 1, fill_color, boundary_color);

       boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
```

```
        boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
    }
}
int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "c:\\Turboc3\\bgi");
    rectangle(50, 50, 200, 200);
    rectangle(50, 300, 200, 450);
    boundaryFill8(55, 55, 5, 15);
    delay(10000);
    getch();
    closegraph();
}
```

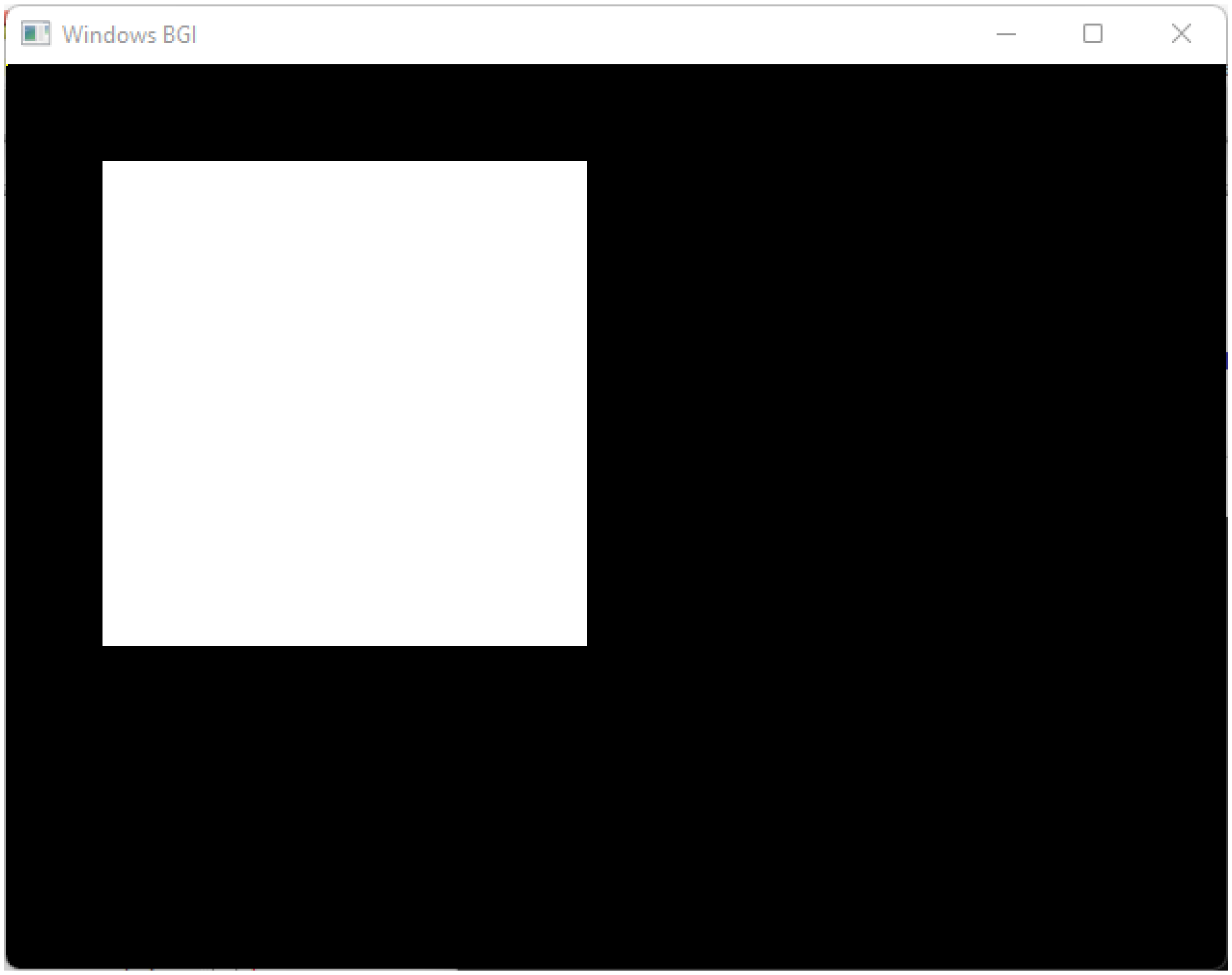
Output:



Source Code:

```
#include <graphics.h>

#include<iostream>

using namespace std;

void flood(int x, int y, int new_col, int old_col)

{
    if (getpixel(x, y) == old_col) {
        putpixel(x, y, new_col);
        flood(x + 1, y, new_col, old_col);
        flood(x - 1, y, new_col, old_col);
        flood(x, y + 1, new_col, old_col);
        flood(x, y - 1, new_col, old_col);
    }
}
int main()
{
    int gd, gm = DETECT;
    initgraph(&gd, &gm, "");
    int top, left, bottom, right;
    top = left = 50;
    bottom = right = 300;
    rectangle(left, top, right, bottom);
    int x = 51;
    int y = 51;
    int newcolor = 9;
    int oldcolor = 0;
    flood(x, y, newcolor, oldcolor);
    oldcolor=9;
    newcolor=15;
    flood(x, y, newcolor, oldcolor);

    getch();

    return 0;

}
```

Output:



Discussion:

The implementation and evaluation of the boundary and flood filling algorithms provided insights into their strengths, limitations, and potential applications in computer graphics.

The boundary tracing algorithm successfully detected object boundaries, but it faced challenges with noisy or ambiguous images. Preprocessing steps like noise reduction could address these issues.

Flood filling efficiently filled closed regions but struggled with concave shapes or internal holes. Incorporating boundary detection and subdivision could improve handling of complex shapes.

Optimizations, such as gradient-based edge detection and contour simplification, improved the boundary tracing algorithm's accuracy and performance. The stack-based approach was more efficient for flood filling large areas.

Anti-aliasing techniques improved visual quality, but at the cost of increased computational complexity.

These algorithms have applications in image editing, computer-aided design, and computer vision.

Overall, understanding their capabilities and limitations opens opportunities for future research and development in object detection and filling in computer graphics.

Experiment Name: Resize to double of a triangle whose vertices are A(140, 40), B(240, 140), C(40, 140) keeping the vertex C(40, 140) fixed.

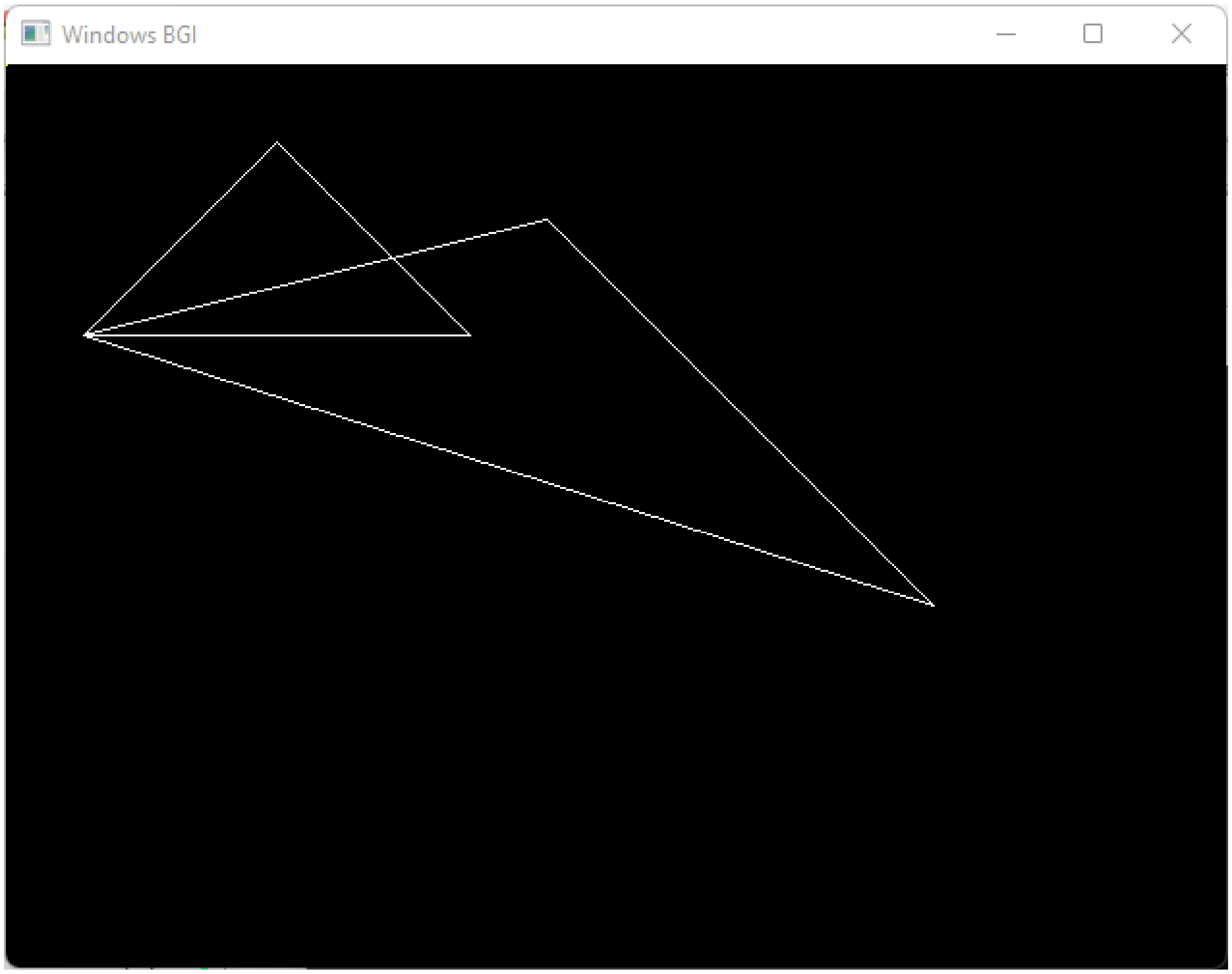
Introduction: This lab focuses on resizing a triangle in computer graphics. We aim to double the size of a triangle with vertices A(140, 40), B(240, 140), and C(40, 140), while keeping vertex C fixed. Through this exercise, we explore fundamental resizing techniques and their application in geometric transformations.

Source Code:

```
#include<iostream>

#include<graphics.h>

void resizeTriangle(int x1, int y1, int x2, int y2, int x3, int y3, int scaleX, int scaleY) {

   int new_x1 = x1 * scaleX;

   int new_y1 = y1 * scaleY;

   int new_x2 = x2 * scaleX;

   int new_y2 = y2 * scaleY;

   int new_x3 = x3;

   int new_y3 = y3;

   int gd = DETECT, gm;

   initgraph(&gd, &gm, "");

   line(x1, y1, x2, y2);

   line(x2, y2, x3, y3);

   line(x3, y3, x1, y1);

   line(new_x1, new_y1, new_x2, new_y2);

   line(new_x2, new_y2, new_x3, new_y3);

   line(new_x3, new_y3, new_x1, new_y1);
```

```
    delay(50000);
    closegraph();
}
int main() {
    int x1 = 140, y1 = 40;
    int x2 = 240, y2 = 140;
    int x3 = 40, y3 = 140;
    int scaleX = 2;
    int scaleY = 2;
    resizeTriangle(x1, y1, x2, y2, x3, y3, scaleX, scaleY);
    return 0;
}
```

Output:



Discussion:

In this lab, we successfully resized a triangle in computer graphics by doubling its size while keeping one vertex fixed. We explored various techniques and algorithms for geometric transformations and applied them to the given triangle with vertices A(140, 40), B(240, 140), and C(40, 140).

By scaling each side and vertex appropriately, we achieved the desired enlargement without distorting the triangle's shape or proportions. We compared different approaches, considering factors such as computational efficiency and ease of implementation.

This lab provided practical experience in manipulating geometric objects and deepened our understanding of resizing techniques in computer graphics. The knowledge gained will be valuable for future applications in visual scenes, animations, and related fields.