|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 6**

|  |  |
| --- | --- |
| **Тема**  Реализация и исследование алгоритма построчного затравочного заполнения сплошных областей.  **Студент** Козлов М. А.  **Группа** ИУ7-45Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_**Куров А. В.**\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:** Реализация и исследование алгоритма построчного затравочного заполнения.

**Техническое задание.**

Необходимо обеспечить ввод произвольной многоугольной области, содержащей произвольное количество отверстий. Ввод (вершин многоугольника) производить с помощью мыши, при этом для удобства пользователя должны отображаться ребра, соединяющие вводимые вершины. Предусмотреть ввод горизонтальных и вертикальных ребер. Должен быть предусмотрен ввод затравочной точки.

Пользователь должен иметь возможность задания цвета заполнения.

Работа программы должна предусматривать два режима – с задержкой и без задержки.

Режим с задержкой должен позволить проследить выполняемую последовательность действий.

(Задержку целесообразно выполнять после обработки очередной строки).

Обеспечить замер времени выполнения алгоритма (без задержки, с выводом на экран только окончательного результата).

Продемонстрировать возможность заполнения с помощью затравочного алгоритма произвольной области, ограниченной замкнутой кривой линией.

**Теоретический материал.**

Алгоритмы заполнения с затравкой используют для заполнения области затравочный пиксель, он расположен внутри заполняемой области. Области заполнения могут быть внутренне-определёнными или гранично-определёнными, 4х связанными или 8ми связанными. Построчный алгоритм затравочного заполнения применим к гранично-определённым 4х связанным областям.

Работу алгоритма можно разбить на 4ые этапа.

Затравочный пиксель извлекается из стека, содержащий затравочные пиксели.

Интервал с затравочным пикселем заполняется влево и вправо от затравки вдоль сканирующей строки до тех пор, пока не будет найдена граница. В xLeft и xRight запоминаются крайний левый и крайний правый пиксели интервала.

В диапазоне xLeft <= x <= xRight проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется, есть ли на них ещё не заполненные пиксели. Если такие есть, то в указанном диапазоне крайний правый пиксель в каждом интервале отмечается как затравочный и помещается в стек.

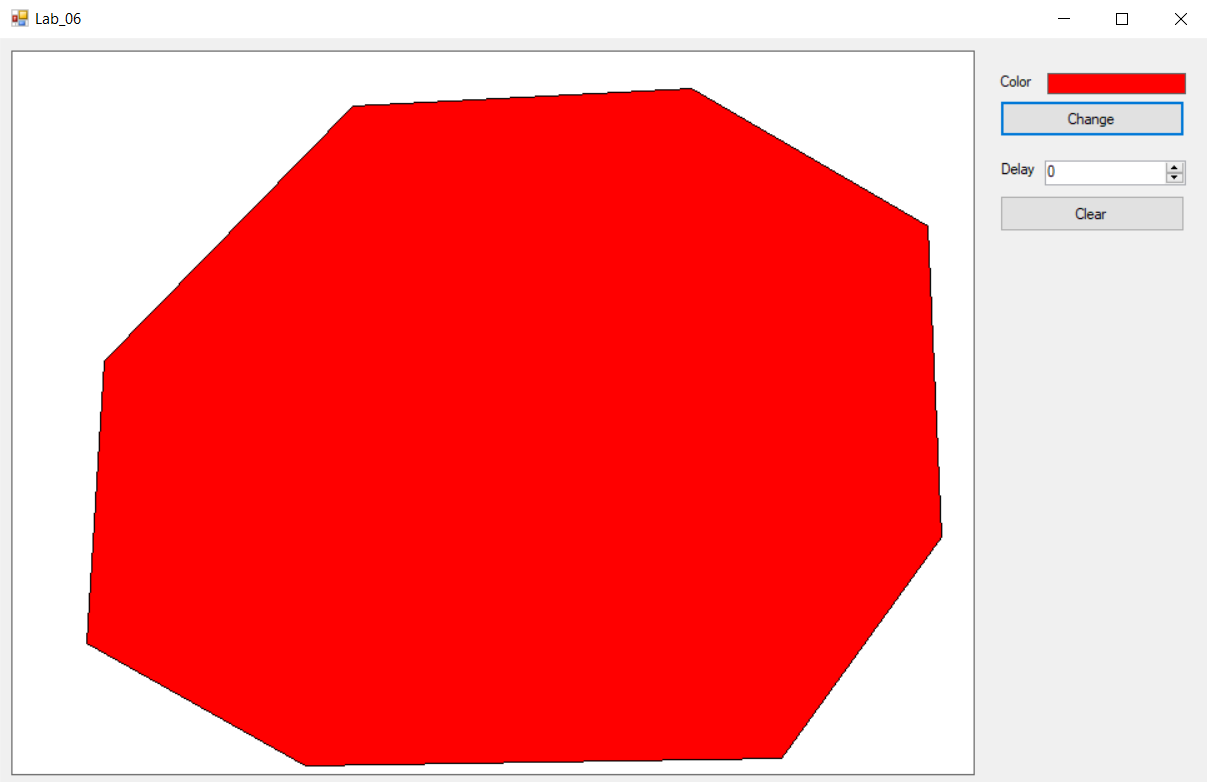
При инициализации алгоритма в стек помещается затравка, работа завершается при опустошении стека.

**Примеры работы программы.**

Ввод горизонтальных и вертикальных рёбер осуществляется при зажатии клавиши SHIFT.

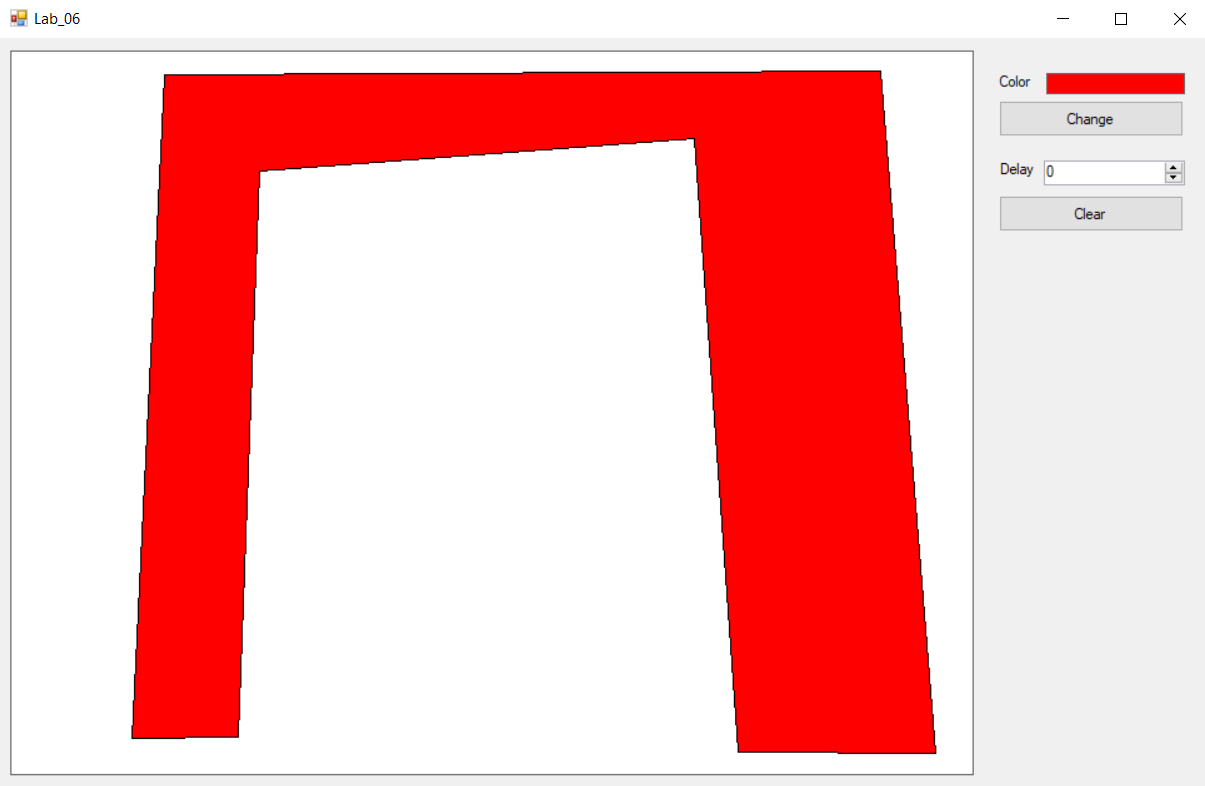
Ввод затравочного пикселя осуществляется левой кнопкой мыши при зажатой клавиши CTRL.

Выпуклый многоугольник.



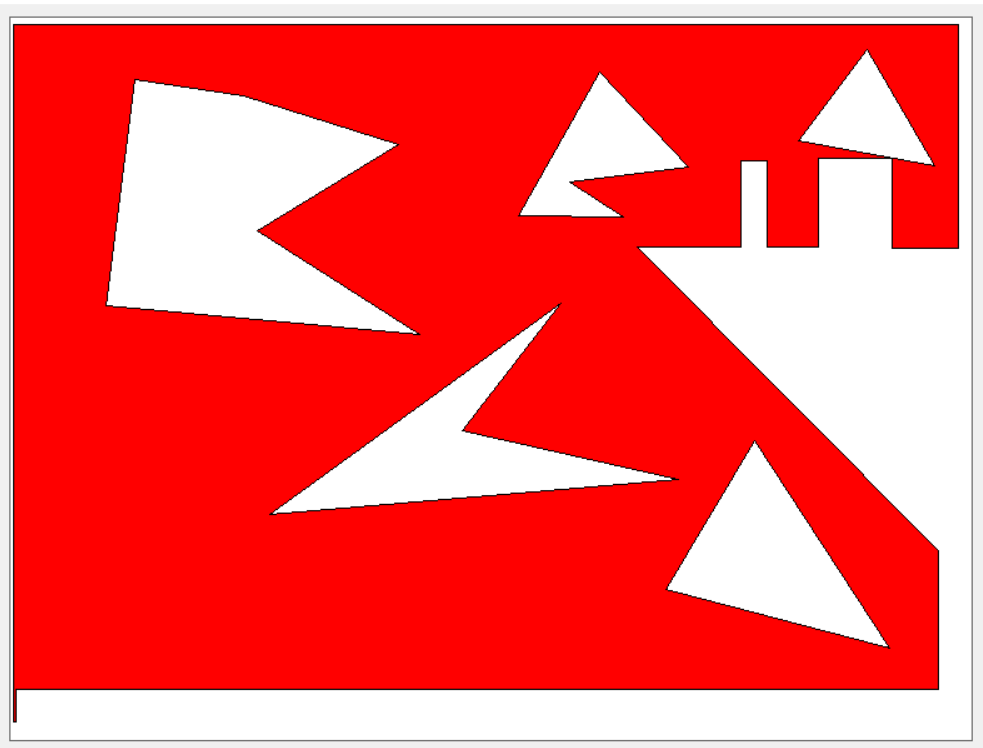
Время выполнения 1033ms.

Невыпуклый многоугольник.



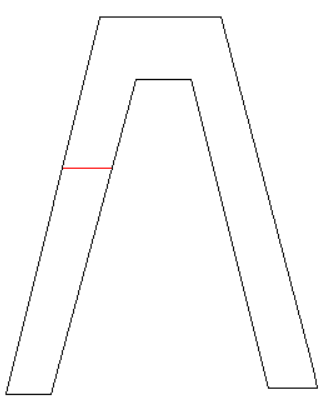
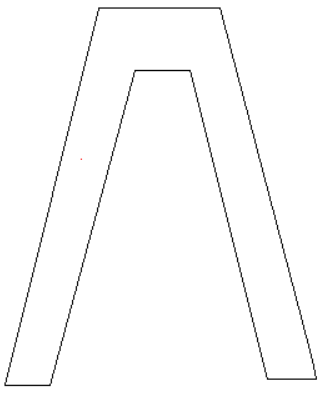
Время выполнения 544ms.

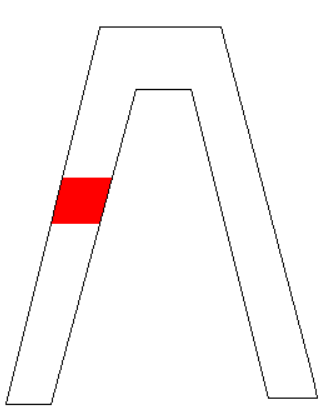
Невыпуклый многоугольник с отверстиями.

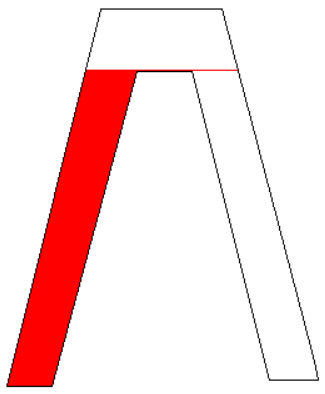
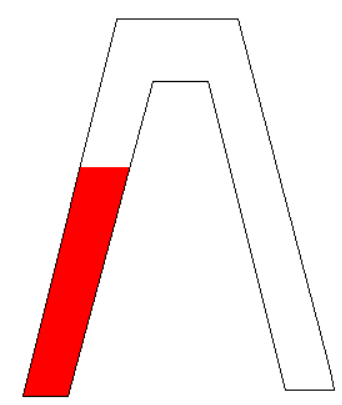


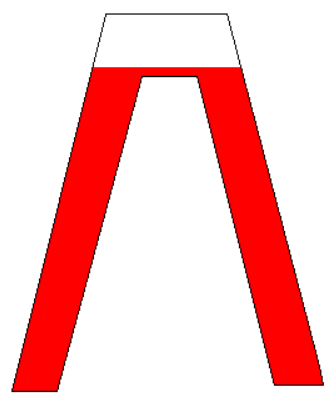
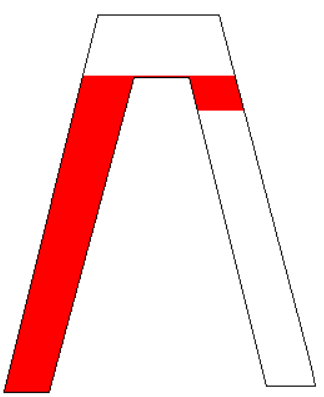
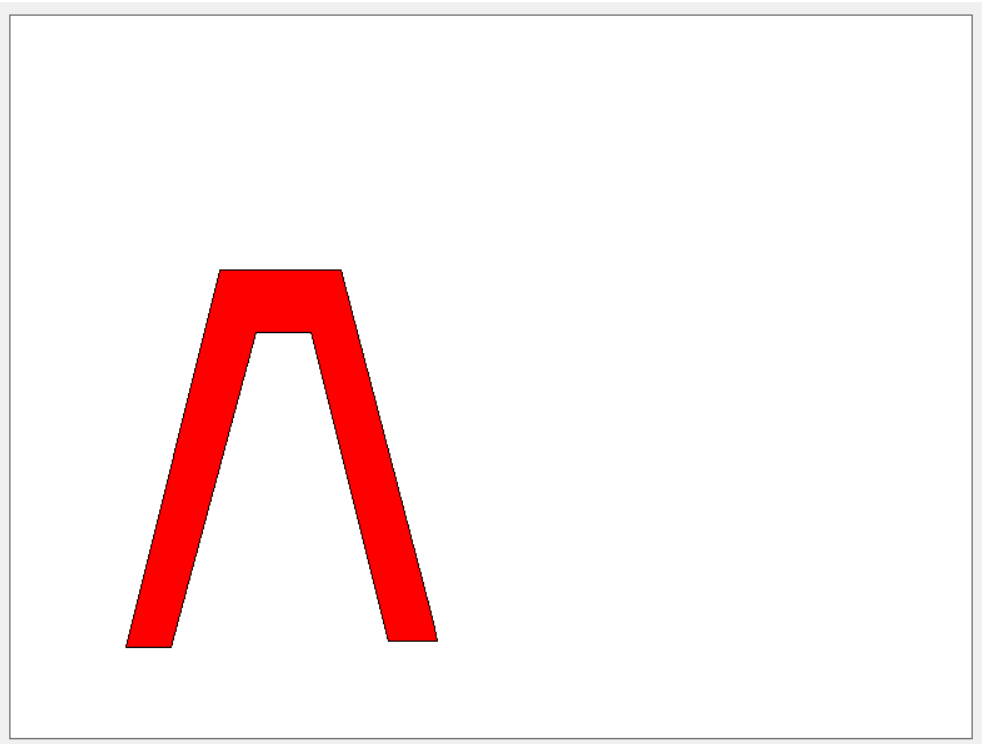
Время выполнения 1006ms.

Пример выполнения программы с задержкой.







Время выполнения 88ms.

**Исходный код.**

private bool IsEqualColors(Color colorA, Color colorB)

{

return colorA.ToArgb() == colorB.ToArgb();

}

// Возвращает true, если пиксель граничный

private bool PixelIsBorder(Bitmap canvas, int x, int y, Color borderColor)

{

if ((x >= 0) && (x < canvas.Width) && (y >= 0) && (y < canvas.Height))

{

Color pixelColor = canvas.GetPixel(x, y);

return IsEqualColors(pixelColor, borderColor);

}

return false;

}

// Возвращает true, если пиксель уже закрашен или граничный

private bool PixelIsFillOrBorder(Bitmap canvas, int x, int y, Color fillColor, Color borderColor)

{

if ((x >= 0) && (x < canvas.Width) && (y >= 0) && (y < canvas.Height))

{

Color pixelColor = canvas.GetPixel(x, y);

return IsEqualColors(pixelColor, fillColor) || IsEqualColors(pixelColor, borderColor);

}

return false;

}

private void FindSeed(Stack<Point> stack, Bitmap canvas, int xLeft, int xRight, int y, Color fillColor, Color borderColor)

{

// поиск слева на право новых затравочных пикселей

int x = xLeft;

bool flag = false; // найден не закрашенный пиксель

while (x <= xRight)

{

// Проходим до конца незакрашенного интервала

while (!PixelIsFillOrBorder(canvas, x, y, fillColor, borderColor) && x <= xRight)

{

flag = true;

x++;

}

if (flag)

{

// В стек помещаем крайний справа пиксель

if (x == xRight && !PixelIsFillOrBorder(canvas, x, y, fillColor, borderColor))

stack.Push(new Point(x, y));

else

stack.Push(new Point(x - 1, y));

flag = false;

}

// Продолжим проверку, если интервал был прерван

do

{

x++;

}

while (PixelIsFillOrBorder(canvas, x, y, fillColor, borderColor) && x <= xRight);

}

}

public float FillArea(Bitmap canvas, Color fillColor, Color borderColor, Point seed)

{

var sw = new Stopwatch();

sw.Start();

Stack<Point> stack = new Stack<Point>();

Point curPoint;

int x, y, xRight, xLeft;

stack.Push(seed);

while (stack.Count > 0)

{

curPoint = stack.Pop();

x = curPoint.X;

y = curPoint.Y;

while(!PixelIsBorder(canvas, x, y, borderColor) && x < canvas.Width)

{

canvas.SetPixel(x, y, fillColor);

x++;

}

xRight = x - 1;

x = curPoint.X - 1;

while (!PixelIsBorder(canvas, x, y, borderColor) && x >= 0)

{

canvas.SetPixel(x, y, fillColor);

x--;

}

xLeft = x + 1;

if (y > 0)

FindSeed(stack, canvas, xLeft, xRight, y - 1, fillColor, borderColor);

if (y < canvas.Height - 1)

FindSeed(stack, canvas, xLeft, xRight, y + 1, fillColor, borderColor);

}

sw.Stop();

return sw.ElapsedMilliseconds;

}