|  |  |
| --- | --- |
| 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»**  **Дисциплина Компьютерная графика**  **Студент Кузин Антон**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.**Цель:**

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.

**Теоретические сведения:**

Построчному алгоритму заполнения с затравкой изначально кроме ограничивающей области необходимо задать одну затравку – точку, лежащую внутри области. Затем закрашиваются пиксели, лежащие в пределах непрерывного интервала и ордината которых равна ординате затравки, следующим шагом производится поиск новых затравочных пикселей на соседних строках.

К преимуществам алгоритма можно отнести относительно небольшой размер стека, за счёт хранения одного затравочного пикселя для каждого непрерывного интервала, анализ только тех пикселей, которые лежат в пределах заполняемой области.

**Практическая часть:**

bool isClear(int x, int y, canvas\_t &canvas)

{

return (canvas.image->pixel(x, y) != canvas.boundColor->rgb()) &&

(canvas.image->pixel(x, y) != canvas.fillColor->rgb());

}

void findNewSeed(QStack<QPoint> \*stack, int xl, int xr, int y, canvas\_t &canvas)

{

int x = xl;

bool flag;

int xent;

while(x <= xr)

{

flag = false;

while ((x <= xr) && isClear(x, y, canvas))

{

if (!flag)

flag = true;

x++;

}

if (flag)

{

if ((x == xr) && isClear(x, y, canvas))

stack->push(QPoint(x, y));

else

stack->push(QPoint(x - 1, y));

flag = false;

}

xent = x;

while ((x < xr) && !isClear(x, y, canvas))

x++;

if (x == xent)

x++;

}

}

double MainWindow::fill(const QPoint &start\_seed, canvas\_t &canvas, float delay)

{

QStack<QPoint> stack;

QPoint start;

stack.push(start\_seed);

int xr, xl;

while(stack.length() > 0)

{

QueryPerformanceCounter(&t1);

start = stack.pop();

canvas.image->setPixel(start, canvas.fillColor->rgb());

int tx = start.x();

int x = start.x() + 1;

int y = start.y();

while (canvas.image->pixel(x, y) != canvas.boundColor->rgb())

{

Q\_ASSERT(x < 800);

canvas.image->setPixel(x, y, canvas.fillColor->rgb());

x++;

}

xr = x - 1;

x = tx - 1;

while (canvas.image->pixel(x, y) != canvas.boundColor->rgb())

{

Q\_ASSERT(x > 0);

canvas.image->setPixel(x, y, canvas.fillColor->rgb());

x--;

}

xl = x + 1;

findNewSeed(&stack, xl, xr, y - 1, canvas);

findNewSeed(&stack, xl, xr, y + 1, canvas);

if (ui->delayCheckBox->isChecked())

{

imageView();

repaint();

QThread::msleep(delay);

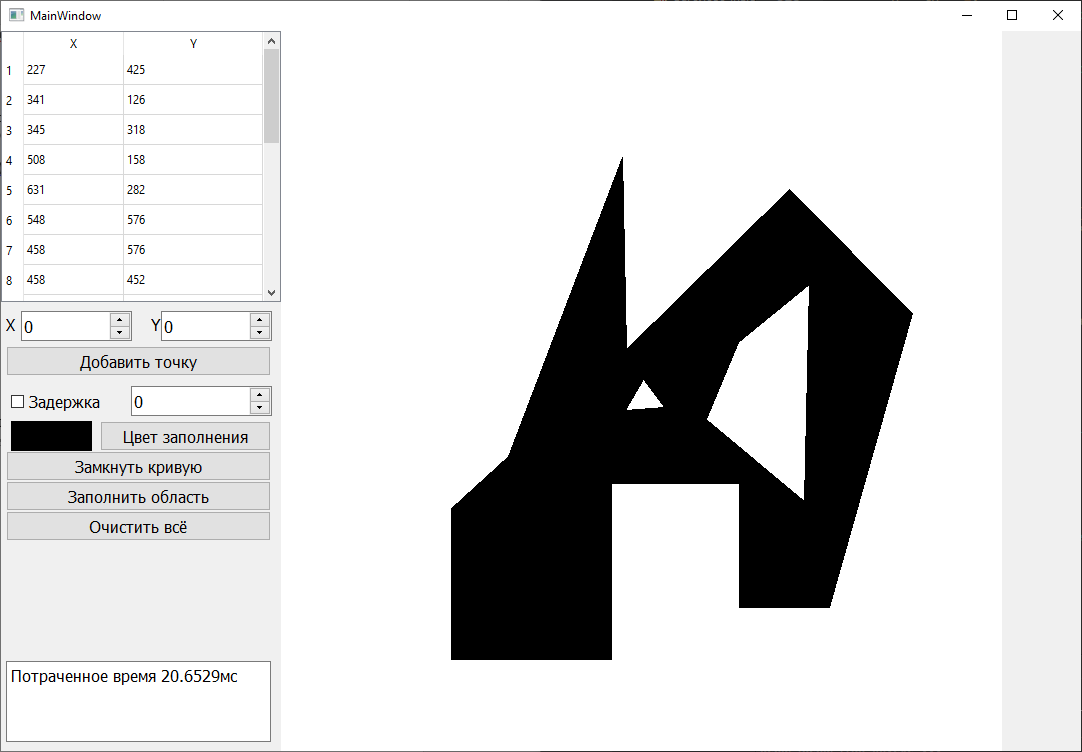
}

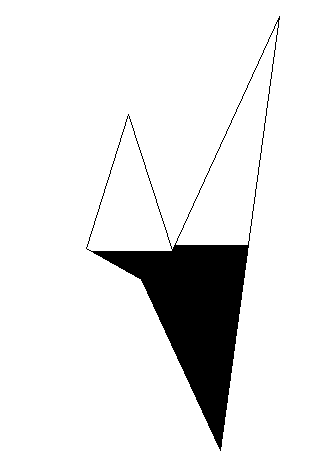
}

return elapsedTime;

}

**Пример работы:**





**Вывод**

Был реализован построчный алгоритм заполнения с затравкой, исследовано его быстродейсвтие.