**Лабораторная работа N 3**  “ Программная реализация алгоритмов заполнения с затравкой и исследование их временных характеристик”.

**Цель работы**: *реализовать построчный алгоритм заполнения гранично-определенной области с затравкой. Область должна быть с внутренними дырами и зубцами на границе. Предусмотреть возможность ввода координат произвольной области и любой затравочной точки. Оценить время реализации алгоритма для областей различной сложности.*

В алгоритмах затравочного заполнения сплошных областей предполагается, что известна определенная точка (затравка) внутри области и необходимо определить точки, соседние с затравочной и расположенные внутри области. Если соседняя точка расположена не внутри, значит обнаружена граница области, если же точка находится внутри, то она становится новой затравочной точкой и поиск продолжается рекурсивно.

Известны два алгоритма заполнения с затравкой :

- простой алгоритм заполнения с затравкой;

- построчный алгоритм заполнения с затравкой.

Данные алгоритмы применимы к гранично-определенным областям, то есть таким, что все пикселы на границе данных областей имеют выделенное значение или цвет , но не один пиксел из внутренней части таких областей не может иметь это выделенное значение (рис. ). Гранично-определенные области могут быть 4- или 8-связными (рис. ). Любой пиксел в 4-связной области становится доступен с помощью движений только в четырех направлениях: вверх, вниз, направо, налево; для 8-связной же области к любому пикселу можно подойти. используя комбинацию движений в 2-х горизонтальных, в 2-х вертикальных и 4-х диагональных направлениях. Алгоритм заполнения 8-связной области заполнит и 4-связную область (обратное неверно).

Простой алгоритм заполнения с затравкой ( базовый алгоритм разработан Смитом, см. [ 5 ] ) легко реализовать, используя понятие стека с порядком обслуживания “ первым пришел, последним обслужен”, то есть, когда новое значение помещается в стек, то все остальные опускаются вниз на один уровень, а когда значение извлекается из стека, все остальные поднимаются на один уровень. Такой стек еще называется стеком прямого действия. Процесс реализации простого алгоритма с затравкой для 4-связанной области состоит в следующем:

1. Затравка выдает затравочный пиксел (x,y), который затем помещается в стек. Стек инициилизируется.

2. Осуществляется проверка на наличие пикселов в стеке. Если стек не пуст, то:

2.1. Пиксел извлекается из стека и ему присваивается требуемое значение:

если пиксел (x,y) < > требуемое значение, тогда

пиксел (x,y) = требуемое значение.

2.2. Проводится анализ: надо ли помещать соседние пикселы в стек. Для этого, каждый из соседних к данному 4-связанных пиксела проверяется на два условия:

не является ли он граничным или не присвоено ли уже ему требуемое значение:

если (пиксел (x+1,y) < > граничное зн-е и пиксел(x+1,y) < > требуемое зн-е),

тогда пиксел (x+1,y) помещается в стек;

если (пиксел (x,y+1) < > граничное зн-е и пиксел(x,y+1) < > требуемое зн-е),

тогда пиксел (x,y+1) помещается в стек;

если (пиксел (x-1,y) < > граничное зн-е и пиксел(x-1,y) < > требуемое зн-е),

тогда пиксел (x-1,y) помещается в стек;

если (пиксел (x,y-1) < > граничное зн-е и пиксел(x,y-1) < > требуемое зн-е),

тогда пиксел (x,y-1) помещается в стек.

То есть, если проверка какого-либо из двух случаев дает положительный результат, то пиксел игнорируется, в противном случае этот пиксел помещается в стек.

В приведенном алгоритме 4-связанные пикселы проверяются, начиная с правого от текущего, в направлении против часовой стрелки. Данный алгоритм может заполнять и области, содержащие дыры.

На лабораторную работу выносится реализация построчного алгоритма с затравкой, поэтому расмотрим подробнее построчный алгоритм заполнения с затравкой для 4-связной области (для самостоятельной работы предлагается переделать его для 8-связанной области и заполнение проводить не в 4-х , а в 8-ми направлениях). Гранично-определенная 4-связанная область может быть выпуклой, не выпуклой, а также содержать внутри себя дыры; но во внешней, примыкающей к данной гранично-определенной области, не должно быть пикселов с цветом заполнения.

При инициализации алгоритма в стек прямого действия помещаются координаты введенного затравочного пиксела (x,y).

При реализации алгоритм можно разделить на следующие этапы:

Пока стек не пуст:

1. Затравочный пиксел на интервале извлекается из стека, содержащего затравочные пикселы (в построчном алгоритме с затравкой размер стека минимизируется за счет хранения только одного затравочного пиксела для любого непрерывного интервала на сканирующей строке) и ему присваивается требуемое значение:

пиксел (x,y) = требуемое зн-е.

2. Интервал с затравочным пикселом заполняется влево и вправо от затравочной точки вдоль сканирующей строки до тех пор, пока не будет найдена граница:

2.1. Сохраняем координаты абсциссы затравочного пиксела: xt=x.

2.2. Заполняем интервал вправо от затравки: x=x+1,

пока (пиксел (x,y) < > граничное зн-е)

делать:( пиксел (x,y)=требуемое зн-е;

x=x+1);

2.3. В переменной xr запоминаем крайний правый пиксел: xr=x-1.

2.4. Восстанавливаем координату абсцисс затравки: x =xt.

2.5. Заполняем интервал слева от затравки: x=x-1,

пока (пиксел (x,y) < > граничное зн-е)

делать:( пиксел (x,y)=требуемое зн-е;

x=x-1);

2.6. В переменной xl запоминаем крайний левый пиксел: xl= x+1.

2.7. Восстанавливаем координату абсциссы затравки: x=xt.

3. В диапазоне xl <= x <= xr проверяются строки , расположенные непосредственно над и под текущей строкой и на них определяются еще не заполненные пикселы. При обнаружении таких пикселов ( если не все пикселы уже заполнены или граничны), крайний правый пиксел в каждом интервале данного диапазона заносится как затравочный в стек. То есть:

3.1. Осуществляем проверку строки, расположенной выше текущей:

x=xl.

y=y+1.

Пока (x<=xr) ищем затравочный пиксел на строке , выше текущей:

f=0 ( промежуточная переменная);

пока (пиксел (x,y)< > граничное зн-е и пиксел(x,y) < > требуемое зн-е и x<xr)

делать: (если f=0 , тогда f=1; и абсцисса увеличивается на 1: x=x+1);

помещаем в стек крайний справа пиксел : если f=1, тогда

(если (пиксел(x,y)< >граничное зн-е и пиксел(x,y)< >требуемое зн-е и x=xr),

тогда пиксел (x,y) помещается в стек);

в противном случае в стек заносится пиксел с координатами (x-1,y);

затем промежуточной переменной присваивается нулевое значение: f=0.

Далее проверка не является ли строка выше текущей границей многоу-

ника, или уже полностью заполненной продолжается в том случае, если

интервал был прерван:

xn=x;

пока (пиксел (x,y)=граничное зн-е или пиксел(x,y)=требуемое зн-е и x<xr)

делать : x=x+1;

убедимся, что координата абсциссы пиксела увеличилась:

если x=xn, тогда х=х+1.

3.2. Аналогично пункту 3.1. осуществляем проверку строки ниже текущей, изменения лишь касаются координаты y, ей присваивается значение : y=y-1.

И так до тех пор, пока стек не становится пустым, область заполненной , а работа алгоритма завершенной.

Если гранично-определенная область примыкает к краю экрана, то при реализации алгоритмов заполнения с затравкой возможен выход за пределы области, чтобы этого не случилось рекомендуется:

- либо на каждом шаге производить проверку, не превзойдены ли пределы памяти;

- либо сразу же увеличивать пределы памяти на краях, для исключения возможности появления ошибки.

В следущих лабораторных работах рассматриваются алгоритмы заполнения сплошных областей в порядке сканирования ( растровой развертки ) дисплея. В этих алгоритмах в порядке сканирования строк определяется, лежит ли точка внутри области, ограниченной ребрами многоугольника или нет , поэтому процесс реализации таких алгоритмов начинается от верхней вершины верхнего ребра многоугольника и идет к нижнему пикселу нижнего ребра. Информация о нахождении текущей точки внутри или снаружи заполняемой точки легко получается с помощью “ контроля четности” , то есть подсчета числа пересечений ребер многоугольника, ограничивающих закрашиваемую область, со строкой сканирования, на которой находится эта точка ( рис. 3.3.), тт. V1, V2 - лежат внутри области, так как число пересечений нечетное, а тт. V3,V4 - расположены вне закрашиваемой области, ибо число пересечений четное.

**Лабораторная работа N 4** “ Программная реализация алгоритмов заполнения сплошной области с упорядоченным списком ребер”

**Цель работы**: *заполнить произвольно введенную область, используя алгоритм с упорядоченным списком ребер. Определить время заполнения области и сравнить с построчным затравочным алгоритмом.*

Это быстродействующий алгоритм, суть которого состоит в сортировке в порядке сканирования точек пересечений ребер многоугольника, ограничивающего заданную область, со сканирующими строками. Процесс реализации алгоритма можно разбить на два этапа: подготовка (сортировка) данных и преобразование отсортированных данных в растровую форму.

При реализации данного алгоритма большое время уходит на подготовку данных, для этого необходимо определить для каждого ребра многоугольника точки пересечений со строками сканирования, проведенными через середины интервалов, используя уже реализованные алгоритм Брезенхема или цифровой дифференциальный анализатор. Горизонтальные ребра не могут пересекать сканирующую строку и , следовательно, игнорируются. Но при синтезе изображения они, естественно, присутствуют , ибо формируются верхней и нижней строками пикселов. Каждое пересечение ( x, y+1/2 ) заносится в список. Затем список отсортировывается по строкам и по возрастанию абсциссы в строке ( то есть точка с координатами ( x1,y1 ) будет предшествовать точке с координатами ( x2,y2 ) в том случае, если y1> y2 или при равных ординатах точек - x1 <=x 2.

После подготовки данных, они преобразуются в растровую форму, для этого из отсортированного списка выделяются пары точек ( x1,y1 ) и ( x2,y2 ) ( они соответствуют условию предыдущего абзаца) и на сканирующей строке инициализируются точки с целыми значениями x, которые соответствуют неравенству: x1 <= x+1/2 <= x2.

При реализации данного алгоритма дополнительные трудности возникают при пересечении сканирующей строки и многоугольника точно по вершине (рис.4.1). При использовании соглашений о середине интервала между строками сканирования возможен случай, когда получится нечетное количество пересечений. и, следовательно, разбиение пикселов на пары даст неверный результат. Правильный результат в этом случае получается, если учитывать точку пересечения в вершине два раза в том случае, если она является точкой локального минимума или максимума, и один раз в противном случае. Локальный максимум или минимум многоугольника в иссследуемой вершине определяется с помошью проверки концевых точек ребер, соединенных в данной вершине. Если у обоих конечных точек ординаты y больше, чем у вершины, значит вершина является точкой локального минимума. Если одна больше, а другая меньше, значит вершина - точка локального максимума. Если же одна ордината больше, а другая меньше, значит вершина не является ни точкой локального минимума, ни точкой локального максимума ( на рис. 4.1. т. V1 - локальный максимум, V3 - локальный минимум, тт. V2,V4 не являются ни локальным минимумом , ни локальным максимумом; то есть при пересечении в тт. V1 , V3 учитывается два пересечения со строками сканирования, а в тт. V2 и V4 - одно).

Вышеописанный алгоритм с упорядоченным списком ребер требует формирования большого списка точек пересечений, который должен быть полностью отсортирован. Поэтому разработан более эффективный алгоритм с упорядоченным списком ребер, в котором процесс сортировки разделяется по строкам в направлении y и в строке в направлении x с помощью групповой сортировки по y . При подготовке данных аналогичным образом определяются точки пересечений ребер многоугольника со строками сканирования, затем координата x точки пересечения помещается в группу, соответствующую y. Для каждой такой y-группы отсортировываются точки пересечений в порядке возрастания координаты абсцисс, то есть x1 предшествует в отсортированном списке x2 , если x1 <= x2. Затем отсортированные данные преобразовываются в растровую форму, то есть для каждой строки сканирования выделяются из списка абсцисс точек пересечений пары точек пересечений и на соответствующей строке сканирования - y активизируются пикселы для их целых значений ( как и в предыдущем случае ).

Таким образом, модифицируя простой алгоритм с упорядоченным списком ребер мы сначала с помощью групповой сортировки по оси ординат проводим сортировку в порядке сканирования строк, а затем сортируем в строке по оси абсцисс, то есть начинаем процесс развертки до окончания сортировки. Как видно из алгоритма, в нем легче добавлять или удалять информацию из сортировочного списка, а следовательно, и дисплейного файла: необходимо только добавить или удалить информацию из соответствующей y-группы и пересортировать только измененные строки.

Несмотря на то, что при использовании y-групп задача сортировки упрощается, при реализации модифицированного алгоритма с упорядоченным списком ребер требуется резервировать большое количество памяти ( значительная часть которой не будет использована ) или ограничивать число пересечений с данной сканирующей строкой. Эти недостатки легко преодолеваются благодаря введению добавочной структуры данных - связанного списка активных ребер (САР). В этом случае значительно сокращается потребность в памяти, а точки пересечения со строками сканирования вычисляются в пошаговом режиме.

В процессе подготовки данных в алгоритме с упорядоченным списком ребер, использующем список активных ребер необходимо определить для каждого ребра многоугольника наивысшую сканирующую строку, пересекаемую данным ребром. Это удобно сделать анализируя строки сканирования, проведенные через середины отрезков ( через y+1/2). Затем ребро многоугольника сохраняется в y-группе, соответствующей этой сканирующей строке и формируется связанный список, в который заносятся следующие значения:

- начальное значение абсцисс точек пересечения -x;

- число сканирующих строк, пересекаемых ребром многоугольника - y;

- шаг приращения по оси абсцисс при переходе от одной строки сканирования к другой - x.

В процесе преобразования подготовленных данных в растровую форму для каждой строки сканирования осуществляется проверка соответствующей y-группы на наличие новых ребер, в случае их обнаружения, соответствующие значения заносятся в САР. После чего координаты абсцисс точек пересечения из связанного списка отсортировываются в порядке возрастания ( то есть x1 <= x2 ) и из него выделяются пары точек, с помощью которых активизируются пикселы на строке сканирования для целых значений x аналогично двум предыдущим случаям. Затем для каждого ребра из САР число сканирующих строк, пересекаемых данным многоугольником - y, уменьшается на 1. Если в результате, y становится меньше 0, то данное ребро исключается из САР и вычисляется новое начальное значение координаты абсцисс точек пересечения xn = x + x.

Затем переходят к новой строке сканирования и этапы реализации алгоритма повторяются.

Так как в данном алгоритме минимизированы операции ввода/вывода, то при реализации можно сделать его независящим от устройств.

**Лабораторная работа N 5** “ Программная реализация алгоритмов заполнения сплошных областей по ребрам и оценка их временных характеристик”

**Цель работы:** *Заполнить произвольно введенную область , используя алгоритм заполнения по ребрам. Определить время заполнения области и сравнить с временем реализации алгоритма с упорядоченным списком ребер и построчным затравочным алгоритмом.*

В алгоритме заполнения по ребрам в отличие от предыдущего не треуется создавать и сортировать списки различных данных. Он очень прост: при его реализации необходимо для каждой строки сканирования, пересекающей ребро многоугольника, ограничивающего закрашиваемую область, в точке с координатами (x1,y1), активизировать все пикселы, которые лежат справа от (x1,y1) и для которых справедливо неравенство : x+1/2 > x1.

К каждому ребру многоугольника алгоритм применяется индивидуально. Пересечения строк сканирования осуществляется аналогично алгоритму с упорядоченным списком ребер, то есть определяются точки пересечений со сканирующими строками, проведенными через середины интервалов (y+1/2).

Рекомендуется использовать этот алгоритм совместно с дисплейным файлом, что позволяет выбирать и обрабатывать ребра в любом порядке. Ибо тогда при обработке ребра многоугольника обрабатываются пикселы из дисплейного файла, соответствующие точкам пересечения ребра со строкой сканирования. После обработки всех ребер многоугольника, ограничивающего закрашиваемую область, дисплейный файл выводится в порядке сканирования на экран графического дисплея.

Данный алгоритм характеризуется существенными недостатками:

- многократной обработкой одного и того же пиксела в случае областей закраши-

вания сложной формы;

- зависимостью алгоритма от операций ввода/вывода.

Для сокращения числа обрабатываемых пикселов используется “перегородка”. В этом случае алгоритм модифицируется следующим образом: анализируется каждая точка пересечения каждого ребра многоугольника со строкой сканирования. Если текущая точка пересечения находится слева от перегородки, то активизируются все пикселы, центры которых лежат справа от точки пересечения ребра со строкой сканирования и слева от перегородки. Если же точка пересечения находится справа от перегородки, то активизируются пикселы, центры которых расположены слева или на пересечении строки сканирования с ребром многоугольника и справа от перегородки (рис. 5.1.)

Рекомендуется проводить перегородку через одну из вершин многоугольника и реализовывать данный алгоритм совместно с использованием дисплейного файла.

Недостатком алгоритма заполнения с перегородкой все же остается неоднократная обработка части пикселов. Для того , чтобы избавиться от этого, разработан модифицированный алгоритм заполнения сплошной области со списком ребер и флагом.

Алгоритм со списком ребер и флагом - двухшаговый алгоритм. Во-первых, обрисовывается контур, ограничивающий область, в результате чего на каждой строке сканирования определяются пары ограничивающих пикселов; а во-вторых, активизируются пикселы, расположенные между вычисленными на предыдущем шаге ограничивающими пикселами.

Процесс реализации алгоритма со списком ребер и флагом состоит из следующих этапов:

1. Обработка ребер многоугольника, ограничивающего заполняемую область. Также, как в предыдущих алгоритмах, считаем, что строки сканирования проходят через центр строк пикселов, то есть через середину интервала y+1/2. После определения пересечений вычисляем самый левый пиксел, центр которого лежит справа от точки пересечения, у которого x+1/2 больше абсциссы точки пересечения.

2. Заполнение области. Для каждой строки сканирования, имеющей точки пересечения с многоугольником, ограничивающем область выполняем следующую последовательность действий:

f=0, если пиксел находится вне области (f-промежуточная переменная);

x=xl (xl - левая граница);

пока (x<=xr ) (xr - правая граница)

делать ( если пиксел (x,y) =граничное зн-е , тогда присвоить f=1;

если f=1, тогда присвоить пикселу (x,y) цвет многоугольника,

в противном случае, присвоить пикселу (x,y) цвет фона;

x=x+1).

Иллюстрация алгоритма заполнения по ребрам и флагу приведена на рис. 5.2. Расссмотрим лишь выделенную строку сканирования. Для обрисовки контура на

этой строке инициилизируются пикселы с абсциссами 2,5,6,9. При заполнении получаем следующие результаты:

f=0;

x=0, x=1 - пикселы не граничные и f=0, следовательно, ничего не происходит;

x=2 - пиксел граничный, следовательно f=1 и пикселу присваивается интенсивность многоугольника;

x=3, x=4 - пикселы не граничные и f=1, следовательно, им присваивается цвет многоугольника;

x=5 - пиксел граничный, следовательно, f=0 и пикселу присваивается фоновый цвет;

x=6 - пиксел граничный, следовательно, f=1 и пикселу присваивается цвет многоугольника;

x=7, x=8 - пикселы не граничные и f=1, следовательно им присваивается цвет многоугольника;

x=9 - пиксел граничный, следовательно, f=0 и пикселу присваивается цвет фона.

При реализации данного алгоритма каждый пиксел обрабатывается только один раз, следовательно временные затраты на выполнение операций ввода/вывода значительно меньше, чем в простом алгоритме заполнения по ребрам или в алгоритме с перегородкой.

Существует аппаратная реализация алгоритма со списком ребер и флагом, в этом случае его скорость выполнения увеличивается на несколько порядков по сравнению с упорядоченным списком ребер ( при программной реализации скорость выполнения алгоритмов отличается незначительно, в чем предлагается убедиться при выполнении Лабораторной работы). При реализации данного алгоритма возможен даже синтез несложных изображений в режиме реального времени.

Ни один из алгоритмов, описанных в данной Лабораторной работе не требует создания и сортировки списков в том случае, если используется дисплейный файл.