4 АЛГОРИТМЫ ЗАКРАШИВАНИЯ ЗАМКНУТЫХ ОБЛАСТЕЙ

В большинстве графических приложений используется одно из существенных достоинств растровых устройств - возможность заполнения областей экрана.

Существует две разновидности заполнения:

- заполнение внутренней части многоугольника, заданного координатами его вершин.
- заполнение (заливка) области, которая либо очерчена границей с кодом (цветом) пикселя, отличающимся от кодов любых пикселов внутри области (т.е. цвет границы области отличен от цвета внутренности области), либо закрашена пикселями с заданным кодом.

Рассмотрим алгоритмы заполнения многоугольника и алгоритмы заливки области.

4.1 АЛГОРИТМЫ ЗАПОЛНЕНИЯ МНОГОУГОЛЬНИКА

Итак, имеем многоугольник, заданный координатами своих вершин. Необходимо заполнить (закрасить) или перекрасить его внутренность.

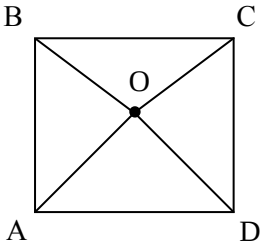
4.1.1 Алгоритм заполнения многоугольника на базе тестов принадлежности пикселя многоугольнику

Данный алгоритм является одним из самых простых алгоритмов. Он заключается в определении, принадлежит ли текущий пиксель внутренней части многоугольника. Если принадлежит, то пиксель перекрашиваем.

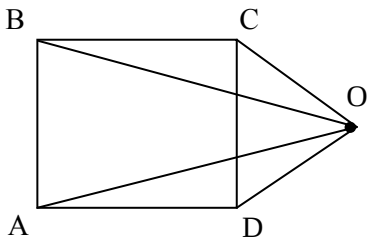
Определить принадлежность пикселя многоугольнику можно с помощью 2-х тестов.

I тест принадлежности пикселя многоугольнику

Подсчитать суммарный угол с вершинами при обходе контура многоугольника по часовой стрелке. Если пиксель внутри, то суммарный угол будет равен 360°, если вне - 0° (рис. 4.1).



Суммарный угол равен
$\angle AOB + \angle BOC + \angle COD + \angle AOD = 360^\circ$
– пиксель О внутри многоугольника



В данном случае $\angle COD = \angle AOB + \angle BOC + \angle AOD$
Значит, суммарный угол равен 0°
– пиксель О вне многоугольника

Рисунок 4.1 – Определение принадлежности пикселя многоугольнику

II тест принадлежности пикселя многоугольнику

Обозначим вершины выпуклого многоугольника P_i, i = 1, 2, ..., n (при этом вершина с номером n + 1 совпадает с вершиной номер 1).

Пусть $A_i(x, y)$ – некоторая точка плоскости, не принадлежащая границе многоугольника (рис.4.2). Нужно определить, находится ли она внутри многоугольника.

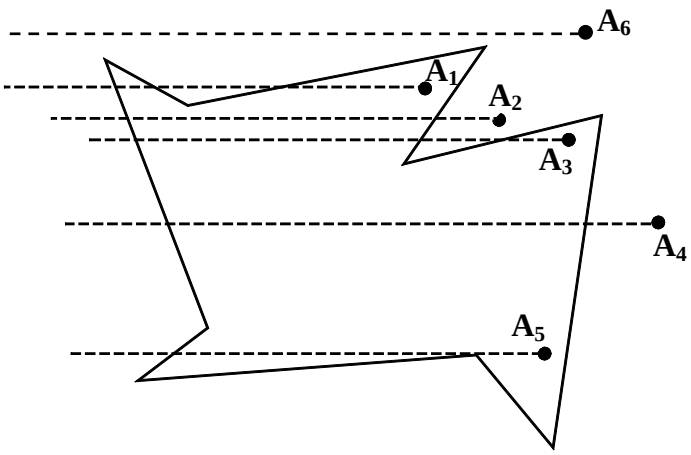


Рисунок 4.2 – Тест принадлежности точки многоугольнику

Из точки A_i проведем горизонтальную полупрямую влево, т.е. точка A_i – правый конец полупрямой.

Рассмотрим возможные варианты существенных пересечений данной полупрямой с границами многоугольника. Пересечение называется ***существенным***, если полупрямая пересекает ломаную, задающую границы многоугольника, а не касается одной из вершин.

При достаточно большом удалении от точки A_i возможны следующие варианты:

1. нет пересечений полупрямой с границей многоугольника, тогда A_i – внешняя точка (на рис.4.2 это точка A_6);
2. четное число пересечений полупрямой с границами многоугольника, тогда A_i – внешняя для этого многоугольника точка (на рис.4.2 это точки A_2, A_4);
3. нечетное число пересечений, A_i - внутренняя точка (на рис.4.2 это точки A_1, A_3).

Особые правила для данного теста:

1. пересечения отрезка с горизонтальными ребрами игнорируются;
2. пересечение игнорируется, если точкой пересечения является вершина ребра, и засчитывается в любом другом случае.

Если просто перебирать все точки растра на плоскости и перекрашивать те, что в соответствии с тестом оказываются внутри, то можно в конце концов закрасить многоугольник и выполнить задачу; это долго и не эффективно, особенно в случаях, когда размер закрашиваемой области много меньше, чем размеры рабочего поля, все пиксели которого должны проверяться на принадлежность к этой области. В связи с этим в любом реальном алгоритме используются те или иные способы ускорения его работы.

1 способ. Многоугольник помещается в некоторый минимально объемлющий прямоугольник со сторонами, параллельными осям, и анализируются точки внутри этого прямоугольника (рис.4.3). Тем самым во многих случаях существенно уменьшается

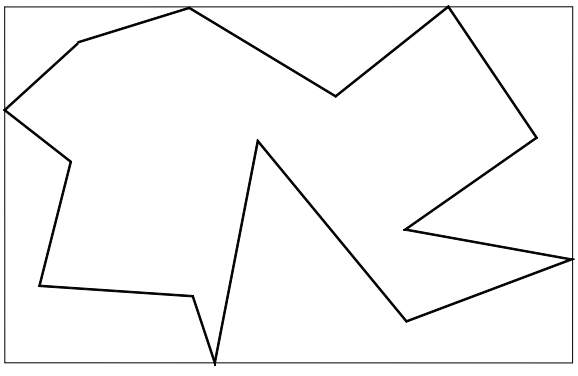


Рисунок 4.3

объем работы: из рассмотрения сразу исключаются все точки вне этого прямоугольника.

2 способ. Всякая горизонтальная прямая разбивается ломаной, образованной границами многоугольника на чередующиеся интервалы, лежащие внутри или снаружи многоугольника. Данный принцип лежит в основе алгоритмов построчного сканирования или XY-алгоритма .

4.1.2 Построчное заполнение многоугольника – XY-алгоритм

Рассмотрим один из наиболее популярных алгоритмов заполнения многоугольников типа построчного сканирования – XY-алгоритм.

Его основная идея – закрашивание фигуры отрезками горизонтальных прямых. Алгоритм представляет собой цикл вдоль оси y, в ходе которого выполняется поиск точек пересечения линии контура с соответствующими горизонталями.

1) найти $\min\{y_i\}$ и $\max\{y_i\}$ среди всех вершин P_i многоугольника.

2) Выполнить цикл по y от $y = \min\{y_i\}$ до $y = \max\{y_i\}$.

{

2.1) Нахождение точек пересечения всех отрезков контура многоугольника с горизонталью y. Координаты x_j точек сечения записать в массив.

2.2) Сортировка массива $\{ x_j\}$ по возрастанию x.

2.3) Вывод горизонтальных отрезков с координатами

$\{ x_0, y\} - \{ x_1, y\}$

$\{ x_2, y\} - \{ x_3, y\}$

. . . . . .

$\{ x_{2k}, y\} - \{ x_{2k+1}, y\}$

Каждый отрезок выводится цветом заполнения

}

В данном алгоритме использовано свойство ***топологии контура фигуры***. Оно состоит в том, что любая прямая линия пересекает любой замкнутый контур четное количество раз (рис.4.4). Для выпуклых фигур точек пересечения с любой прямой всегда две. Таким образом на шаге 2.1 алгоритма в массив $\{x_j\}$ всегда должно записываться парное число точек сечения.

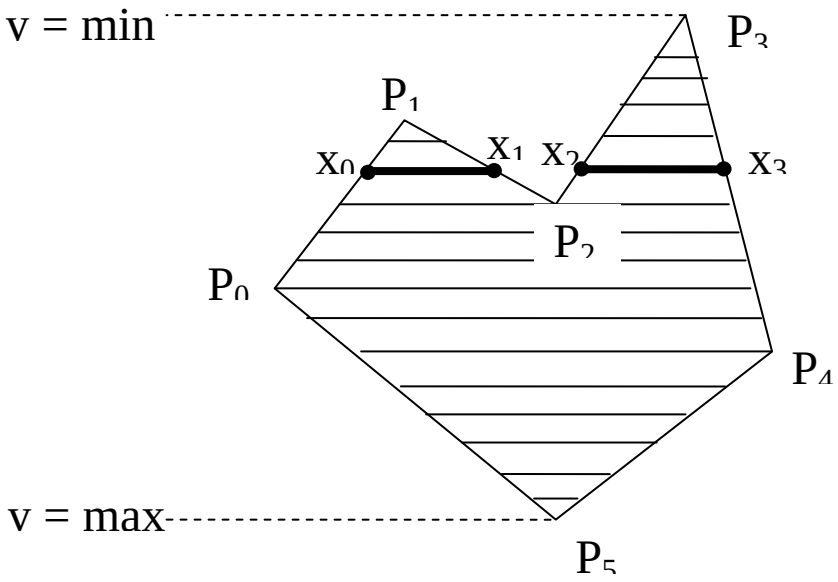


Рисунок 4.4 - Построчная закраска многоугольника

При нахождении точек пересечения горизонтали с контуром необходимо принимать во внимание особые точки. Если горизонталь имеет координату (y), совпадающую с y-координатой вершины P_i, тогда необходимо анализировать то, как горизонталь проходит через вершину. Если горизонталь при этом *пересекает* контур, как, например, в вершинах P_0 или P_4, то в массив записывается одна точка сечения. Если горизонталь *касается* вершины (в этом случае вершина соответствует локальному минимуму или максимуму как, например, в вершинах P_1, P_2, P_3 или P_5), тогда координата точки касания или не записывается, или записывается в массив дважды. Это условие четности количества точек пересечения, хранящихся в массиве $\{ x_j\}$.

Для определения координат (x) точек пересечения для каждой горизонтали необходимо перебирать все n отрезков контура. Среди них определить те, с которыми есть пересечение (для этого можно, например,

рассмотреть координаты у концов отрезка; если они лежат по разную сторону от рассматриваемой горизонтали, то пересечение возможно). Координата x пересечения ребра P_iP_k с горизонталью y равна:

$$x = x_i + (y - y_i)(x_k - x_i)/(y_k - y_i)$$

4.2 ЗАЛИВКА ОБЛАСТИ С ЗАТРАВКОЙ

Нижеследующие алгоритмы используются для закрашивания областей, которые не могут быть заданы координатами своих вершин. В графических приложениях для закраски таких замкнутых областей используются алгоритмы заполнения области с затравкой. ***Затравка*** – это любая точка, лежащая внутри закрашиваемой области.

Основное отличие заливки области с затравкой от заполнения многоугольника: в последнем случае сразу имеется вся информация о предельных размерах части экрана, занятой многоугольником. В алгоритмах же заливки области с затравкой вначале надо прочитать пиксель, затем определить принадлежит ли он области и если принадлежит, то перекрасить.

Для реализации алгоритма заливки с затравкой необходимо каким-нибудь образом задать:

- заливаемую (перекрашиваемую) область,
- код (цвет) пикселя, которым будет выполняться заливка,
- и начальную точку в области, начиная с которой выполняется заливка.

По способу задания области делятся на два типа:

- ***гранично-определенные***, задаваемые своей (замкнутой) границей такой, что коды пикселов границы отличны от кодов внутренней, перекрашиваемой части области. На коды пикселов внутренней части области налагаются два условия - они должны быть отличны от кода пикселов границы и кода пикселя перекраски. Если внутри гранично-

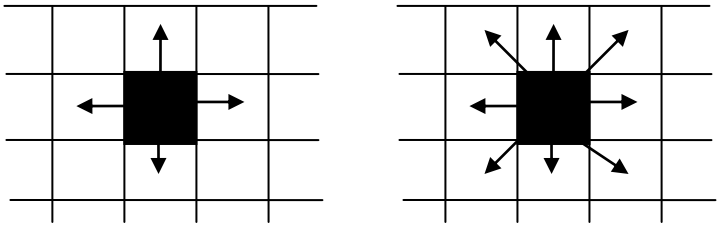
определенной области имеется еще одна граница, нарисованная пикселями с тем же кодом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться (рис.4.5, а);

– ***внутренне-определенные***, нарисованные одним определенным кодом пикселя (рис.4.5, б). При заливке этот код заменяется на новый код закраски.

а) б)

Рисунок 4.5 – Замкнутые области: гранично-определенная (а) и внутренне-определенная (б)

Заливаемая область или ее граница - некоторое связное множество пикселов. По способам доступа к соседним пикселям области делятся на *4-х и 8-ми связные*. В ***4-х связных областях*** доступ к соседним пикселям осуществляется по четырем направлениям - горизонтально влево и вправо и вертикально вверх и вниз (рис.4.6, а). В ***8-ми связных областях*** к этим направлениям добавляются еще 4 диагональных (рис.4.6, б).



а) б)

Рисунок 4.6 – Связность областей

Используя связность, можно, двигаясь от точки затравки, достичь и закрасить все пиксели области.

Важно отметить, что для 4-х связной прямоугольной области граница 8-ми связна (рис. 4.7, а) и, наоборот у 8-ми связной области граница 4-х связна (см. рис. 4.7, б). Поэтому заполнение 4-х связной области 8-ми связным алгоритмом может привести к "просачиванию" через границу и заливке пикселов в примыкающей области.

В общем, 4-х связную область можно заполнить как 4-х, так и 8-ми связным алгоритмом. Обратное же неверно. Так область на рис. 4.7,а можно заполнить любым алгоритмом, а область на рис. 4.7,б, состоящую из двух примыкающих 4-х связных областей, можно заполнить только 8-ми связным алгоритмом.

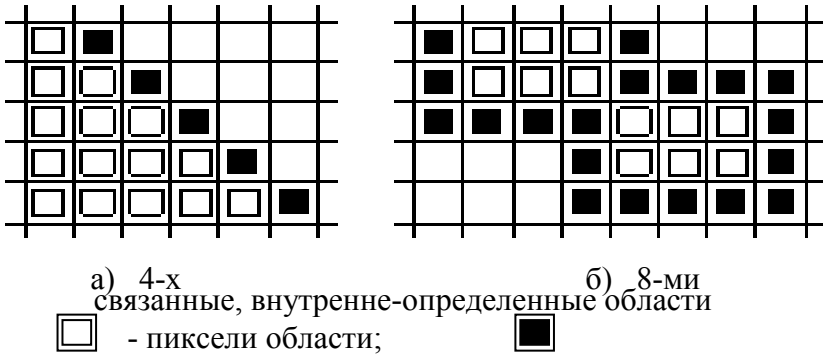


Рисунок 4.7 - Связность областей и их границ

С использованием связности областей и стека можно построить простые алгоритмы закраски как внутренне, так и гранично-определенной области.

4.2.1 Простой алгоритм заливки

Рассмотрим простой алгоритм заливки гранично-определенной 4-х связной области.

1) Рекурсивный алгоритм заливки 4-х связной гранично-определенной области:

– определяется, является ли пиксель граничным или уже закрашенным,

– если нет, то пиксель перекрашивается (выполняется функция ЗАЛИВКА), затем проверяются и если надо перекрашиваются 4 соседних пикселя.

Функцию ЗАЛИВКА определим следующим образом:

```
Функция ЗАЛИВКА (x, y)
{
    Если цвет пикселя (x, y) не равен цвету границы и цвету заполнения, то
    {
        Установить для пикселя (x, y) цвет заполнения;
        ЗАЛИВКА (x+ 1, y);
        ЗАЛИВКА (x-1, y);
        ЗАЛИВКА (x, y+1);
        ЗАЛИВКА (x, y-1);
    }
}
```

Несмотря на простоту и изящество программы, рекурсивная реализация проигрывает итеративной в том, что требуется много памяти для хранения вложенных вызовов.

Опишем итеративный алгоритм закраски 4-х связной гранично-определенной области.

2) Итеративный алгоритм заливки 4-х связной гранично-определенной области:

а) Поместить координаты затравки в стек

б) Пока стек не пуст

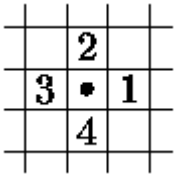
в) Извлечь координаты пикселя из стека.

г) Перекрасить пиксель.

д) Для всех четырех соседних пикселов проверить, является ли он граничным или уже перекрашен.

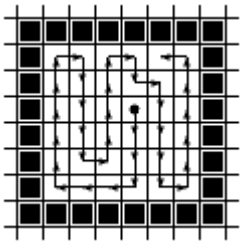
е) Если нет, то занести его координаты в стек.

На рис. 4.8, а) показан выбранный порядок перебора соседних пикселов, а на рис. 4.8, б) соответствующий ему порядок закраски простой гранично-определенной области.



а)

Порядок перебора соседних пикселов



б)

Порядок заливки области

Рисунок 4.8 - Заливка 4-х связной области итеративным алгоритмом

Рассмотренный алгоритм легко модифицировать для работы с 8-ми связными гранично-определенными областями или же для работы с внутренне-определенными.

Как уже отмечалось, очевидный недостаток алгоритмов непосредственно использующих связность закрашиваемой области - большие затраты памяти на стек, так как на каждый закрашенный пиксель в стеке по максимуму будет занесена информация о еще трех соседних. Кроме того, информация о некоторых пикселях может записываться в стек многократно. Это приведет не только к перерасходу памяти, но и потере быстродействия за счет многократной раскраски одного и того же пикселя. Значительно более экономен далее рассмотренный построчный алгоритм заливки.

4.2.2 Построчный алгоритм заливки с затравкой

В данном алгоритме используется пространственная когерентность:

- цвет пикселов в строке меняется только на границах;
- при перемещении к следующей строке размер заливаемой строки скорее всего или неизменен или меняется на 1 пиксель.

Таким образом, на каждый закрашиваемый фрагмент строки в стеке хранятся координаты только одного начального пикселя, что приводит к существенному уменьшению размера стека.

Последовательность работы алгоритма для гранично-определенной области следующая:

1. Координата затравки помещается в стек, затем до исчерпания стека выполняются пункты 2-4.
2. Координата очередной затравки извлекается из стека и выполняется максимально возможное закрашивание влево и вправо по строке с затравкой, т.е. пока не попадется граничный пиксель. Пусть это Хлев и Хправ, соответственно.
3. Анализируется строка ниже закрашиваемой в пределах от Хлев до Хправ и в ней находятся крайние правые пиксели всех незакрашенных фрагментов. Их координаты заносятся в стек.
4. То же самое проделывается для строки выше закрашиваемой.