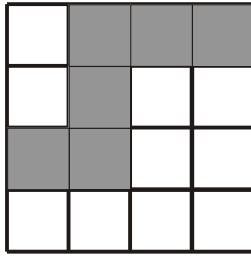
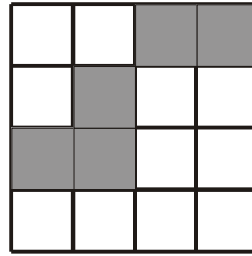


Лекция 1.3. Площадные примитивы



А)



Б)

Разновидности заполнения (для растровых устройств):

- заполнение внутренней части многоугольника, заданного координатами его вершин.
- заливка области, которая либо очерчена границей с кодом пиксела, отличающимся от кодов любых пикселей внутри области, либо закрашена пикселями с заданным кодом.

Двумерные области разделяются на 4-х связные и 8-и связные.

- 4-х связные – две точки считаются принадлежащими одной области, если их координаты отличаются только в одной компоненте, причем не более чем на 1 (рис.А).
- 8-и связные – две точки считаются принадлежащими одной области, если их координаты отличаются не более чем на 1 (возможно в обеих компонентах) (рис.Б).

Простейший способ перекраски

4-х связной области

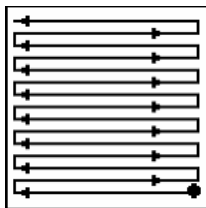
```
void Flood_Fill_4( int x, int y, int oldcolor, int newcolor )
{
    if ( GetPixel ( x, y ) == oldcolor )
    {
        SetPixel ( x, y, newcolor );
        Flood_Fill_4( x-1, y, oldcolor, newcolor );
        Flood_Fill_4( x, y-1, oldcolor, newcolor );
        Flood_Fill_4( x+1, y, oldcolor, newcolor );
        Flood_Fill_4( x, y+1, oldcolor, newcolor );
    }
}
```

```

    }
}

```

- Очень прост
- Очень долгий и ресурсоемкий.
- Некоторые языки не позволяют использовать рекурсию



Видно, что глубина стека должна быть $100 \times 100 = 10\,000$. Если вместо стека использовать очередь, то максимальная расходуемая память сокращается до 200

Методы с затравкой

Предполагается, что известна определенная точка внутри области и необходимо определить точки, соседние с затравочной и расположенные внутри области.

Если соседняя точка расположена не внутри, то значит, что обнаружена граница области, если внутри - то она становится новой затравочной точкой и поиск продолжается рекурсивно.

Задаются:

- заливаемая (перекрашиваемая) область,
- код пиксела, которым будет выполняться заливка,
- начальная точка в области, начиная с которой начнется заливка.

По способу задания области делятся на 2 типа:

- гранично-определенные, задаваемые своей (замкнутой) границей, нарисованной определенным кодом пиксела.
- внутренне-определенные, нарисованные одним определенным кодом пиксела.

Попиксельный алгоритм с затравкой

1. Определяется затравочный пиксель:

```

pix( x, y ) := затравка( x, y );

```

push pix(x, y)

2. Проверка на наличие пикселей в стеке

while (стек не пуст) do

2.1 pop pix(x, y);

if pix(x, y) \neq треб.значение then

pix(x, y) := треб.значение

2.2 для каждого из 4-х соседних пикселей проверяются условия:

а) не является ли он граничным;

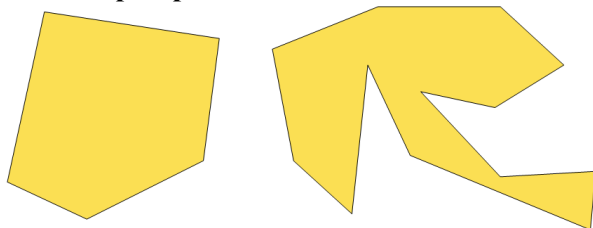
б) не присвоено ли ему требуемое значение.

Построчный алгоритм с затравкой

- Затравочный пиксель извлекается из стека.
- Интервал с затравочным пикселем заполняется влево и вправо от него вдоль сканирующей строки, пока не будет достигнута граница.
- В переменные Хлев и Хправ запоминаются соответственно крайние левый и правый пиксели интервала. В диапазоне $X_{\text{лев}} \leq X \leq X_{\text{прав}}$ проверяются строки расположенные непосредственно над и под текущей строкой. Определяется, есть ли на них незаполненные пиксели. Если такие есть, то крайний левый пиксель помещается в стек.

Размер стека минимизируется за счет хранения только одного затравочного пикселя для любого непрерывного интервала на сканирующей строке.

Списки ребер



Многоугольник - фигура, ограниченная на плоскости простой (непересекающейся) замкнутой ломаной.

- Ломаная задается координатами вершин:

$$Ai(xi, yi), i=1..n,$$

- при этом соседние точки в этом списке являются вершинами ломаной. Т.о. многоугольник задается списком вершин:

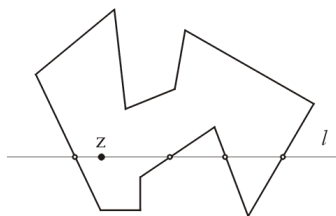
$$A1, A2 \dots An$$

- и списком ребер:

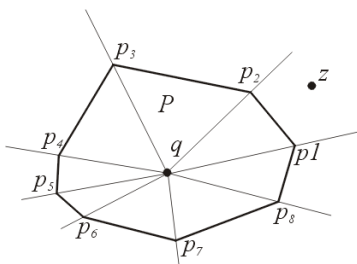
$$(A1, A2), (A2, A3) \dots (An, A1).$$

Задача «закраски» многоугольника заключается в инициализации всех его внутренних точек.

Принадлежность точки многоугольнику

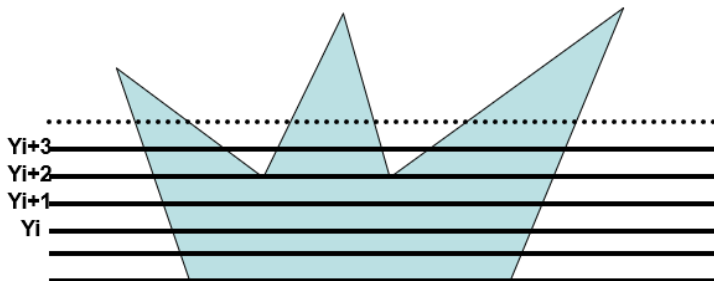


Подсчет количества пересечений



Суммирование углов

Сканирующие линии (Scanline)



- Начинается от верхней вершины верхнего ребра многоугольника и идет к нижней вершине нижнего ребра.
- Пересечение ребра многоугольника со сканирующей строкой делит ее на области.

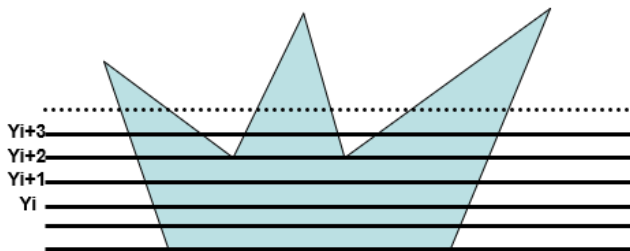
Простой алгоритм со списком ребер

Два этапа:

- Сортировка данных;
- Преобразования отсортированных данных в растровую форму.

Сортировка данных

- Определение точек пересечения рёбер многоугольника со строками сканирования (горизонтальные рёбра не могут пересекать сканирующую строку и, следовательно, игнорируются).
- Каждое пересечение заносится в список;
- Список сортируется по строкам и по возрастанию абсциссы в строке.



Преобразование в растровую форму

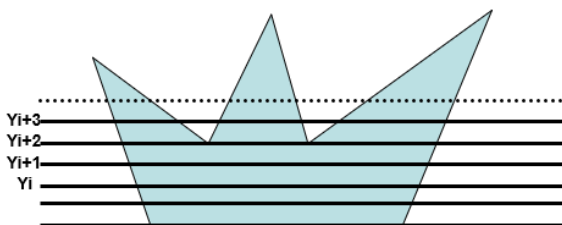
Преобразование в растровую форму:

- Из отсортированного списка выделяется пара точек $(X1, Y1)$ и $(X2, Y1)$;
- На сканирующей строке активизируются точки с целыми значениями X , удовлетворяющими неравенству $X1 \leq X+1/2 \leq X2$.

При реализации данного алгоритма дополнительные трудности возникают при пересечении сканирующей строкой вершины многоугольника. Правильный результат получается, если учитывать точку пересечения вершины два раза в том случае, если она является точкой локального максимума или минимума, и один раз в других случаях.

Алгоритм со списком активных ребер (САР)

- Для каждой строки рассматриваются только те ребра, которые пересекают строку.
- Они задаются списком активных ребер (САР).
- При переходе к следующей строке переисчисляются X -координаты пересечений.
- При появлении в строке сканирования вершин производится перестройка САР.



- Смена интервалов закрашки происходит только тогда, когда в строке сканирования появляется вершина.
- Зная X -координату пересечения с i -ой строкой легко вычислить пересечение с $i+1$ -ой:

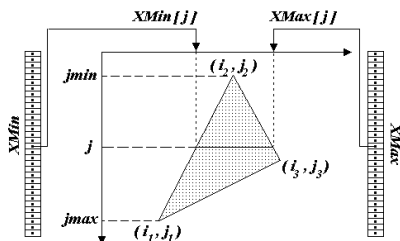
$$X_{i+1} = X_i + 1 / k$$

- где $k = dy/dx$ - тангенс угла наклона ребра.

Алгоритмы заполнения областей

Для каждой сканирующей строки определяются точки пересечения с ребрами многогранника, которые затем упорядочиваются по координате x . Определение того, какой интервал между парами пересечений есть внутренний для многогранника, а какой нет, является достаточно простой логической задачей. При этом если сканирующая строка проходит через вершину многогранника, то это пересечение должно быть учтено дважды в случае, когда вершина является точкой локального минимума или максимума. Для поиска пересечений сканирующей строки с ребрами можно использовать алгоритм Брезенхема построения растрового образа отрезка.

В заключение в качестве примера приведем алгоритм закраски внутренней области треугольника, основанный на составлении полного упорядоченного списка всех отрезков, составляющих этот треугольник. Для записи горизонтальных координат концов этих отрезков будем использовать два массива $Xmin$ и $Xmax$ размерностью, равной числу пикселей растра по вертикали.



Построение начинается с инициализации массивов $Xmin$ и $Xmax$: массив $Xmax$ заполняется нулями, а массив $Xmin$ - числом N , равным числу пикселей растра по горизонтали. Затем определяем значения $Jmin$, $Jmax$, ограничивающие треугольник в вертикальном направлении. Теперь, используя модифицированный алгоритм Брезенхема, занесем границы отрезков в массивы $Xmin$ и $Xmax$. Для этого всякий раз при переходе к очередному пикселю при формировании отрезка вместо его инициализации будем сравнивать его координату i с содержимым j -й ячейки массивов. Если $Xmin[j] > i$,

то записываем координату i в массив $Xmin$. Аналогично при условии $Xmax[j] < i$ координату i записываем в массив $Xmax$.

Если теперь последовательно применить алгоритм Брезенхема ко всем трем сторонам треугольника, то мы получим нужным образом заполненные массивы границ. Остается только проинициализировать пиксели внутри отрезков $\{(Xmin[j], j), (Xmax[j], j)\}$.

Этот алгоритм можно легко распространить на случай произвольного выпуклого многоугольника.

Простой алгоритм с упорядоченным списком ребер

Используя описанные выше методы, можно разработать эффективные алгоритмы растровой развертки сплошных областей, называемые алгоритмами с упорядоченным списком ребер. Они зависят от сортировки в порядке сканирования точек пересечений ребер многоугольника со сканирующими строками. Эффективность этих алгоритмов зависит от эффективности сортировки. Рассмотрим очень простой алгоритм.

Подготовить данные:

Определить для каждого ребра многоугольника точки пересечений со сканирующими строками, проведенными через середины интервалов, для чего можно использовать алгоритм Брезенхема или ЦДА. Горизонтальные ребра игнорируются. Занести каждое пересечение $(x, y + 1/2)$ в список. Отсортировать список по строкам и по возрастанию x в строке; т. е. (x_1, y_1) предшествует (x_2, y_2) , если $y_1 > y_2$ или $y_1 = y_2$ и $x_1 < x_2$.

Преобразовать эти данные в растровую форму:

Выделить из отсортированного списка пары элементов (x_1, y_1) и (x_2, y_2) . Структура списка гарантирует, что $y = y_1 = y_2$ и $x_1 < x_2$. Активировать на сканирующей строке y пиксели для целых значений x , таких, что $x_1 < x + 1/2 < x_2$.

Рассмотрим многоугольник, изображенный на рис. Его вершины: $P_1, (1,1)$, $P_2(8,1)$, $P_3(8,6)$, $P_4(5,3)$ и $P_5(1,7)$. Пересечения с серединами сканирующих строк, следующие:

скан. строка 1.5 $(8, 1.5), (1, 1.5)$

скан. строка 2.5 (8. 2.5), (1, 2.5)

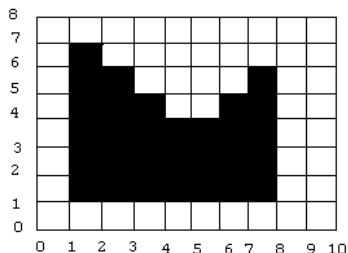
скан. строка 3.5 (8, 3.5), (5.5, 3.5), (4.5, 3.51), (1, 3.5)

скан. строка 4.5 (8, 4.5), (6.5, 4.5), (3.5, 4.5). (1, 4.5)

скан. строка 5.5 (8, 5.5), (7.5, 5.5), (2.5, 5.5), (1, 5.5)

скан. строка 6.5 (1.5, 6.5), (1, 6.5)

скан. строка 7.5 нет



Весь список сортируется в порядке сканирования сначала сверху вниз и затем - слева направо

(1, 6.5), (1.5, 6.5), (1, 5.5), (2.5, 5.5), (7.5, 5.5), (8, 5.5), (1, 4.5),

(3.5, 4.5), (6.5, 4.5).

(8, 4.5), (1, 3.5), (4.5, 3.5), (5.5, 3.5), (8, 3.5), (1, 2.5), (8, 2.5), (4, 1.5),

(8, 1.5)

Выделяя из списка пары пересечений и применяя алгоритм, описанный выше, получим список пикселей, которые должны быть активированы:

(1,6)

(1, 5), (2, 5), (7, 5)

(1, 4), (2, 4), (3, 4), (6, 4), (7, 4)

(1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3)

(1,1), (2, 2), (3, 2), (4, 2), (5, 2), (6, 2), (7, 2), (1, 1), (2, 1), (3,1), (4,1),

(5, 1), (6,1), (7, 1)



Алгоритм XOR для граней

Метод XOR для граней описывается следующим простым алгоритмом: Для каждого ребра в многоугольнике инвертируются цвета всех пикселей, расположенных правее этого ребра. При этом порядок обхода рёбер не имеет значения. В таблице приведены шаги этого алгоритма (движение по часовой стрелке):

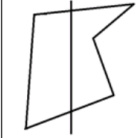




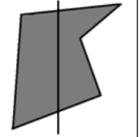
1	2	3	4	5	6

Недостаток этого алгоритма – высокие временные затраты, так как некоторые пиксели обрабатываются более одного раза. Кроме того, чем больше расстояние от изображения до правой границы области экрана, тем больше будет совершено лишних операций.

Алгоритм XOR для граней с перегородкой

От некоторых из недостатков свободен модифицированный вариант алгоритма XOR для граней. Он называется алгоритм XOR для граней с перегородкой. Его идея заключается в том, чтобы инвертировать область не между ребром и границей экрана, а между ребром и специальной вертикальной линией (т.н. перегородкой). Чаще всего

перегородка проводится так, чтобы она пересекала многоугольник. Шаги работы алгоритма приведены в таблице.

1	2	3	4	5	6
					

Метод использующий четность количества граничных точек, отбрасывающих тень на данную точку.

