4. Растровые алгоритмы

4.1. Алгоритмы заливки области

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

Существует две разновидности заполнения:

- для заполнения внутренней части многоугольника, заданного координатами его вершин.
- для заливки области, которая либо очерчена границей с кодом пиксела, отличающимся от кодов любых пикселов внутри области, либо закрашена пикселами с заданным кодом;

Для приложений, связанных в основном с интерактивной работой, используются алгоритмы заполнения области с затравкой. При этом тем или иным образом задается заливаемая (перекрашиваемая) область, атрибут (например, цвет), которым будет выполняться заливка и точка в области, начиная с которой начнется заливка.

В дальнейшем, когда будем говорить «цвет пиксела», будем подразумевать, что на месте цвета может быть любой атрибут пиксела.

По способу задания области делятся на два типа:

- гранично-определенные, задаваемые своей (замкнутой) границей такой, что цвет пикселов границы отличен от цвета внутренней, перекрашиваемой части области. На цвет пикселов внутренней части области налагаются два условия: они должны быть отличны от цвета пикселов границы и цвета заливки. Если внутри гранично-определенной области имеется еще одна граница, нарисованная пикселами с тем же цветом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;
- внутренне-определенные, нарисованные одним определенным цветом пиксела. При заливке этот цвет заменяется на новый цвет.

В этом состоит основное отличие заливки области с затравкой от заполнения многоугольника. В последнем случае мы сразу имеем всю информацию о предельных размерах части экрана, занятой многоугольником. Поэтому определение принадлежности пиксела многоугольнику базируется на быстро работающих алгоритмах, использующих когерентность строк и ребер. В алгоритмах же заливки области с затравкой нам

вначале надо прочитать пиксел, затем определить принадлежит ли он области и если принадлежит, то перекрасить.

Заливаемая область или ее граница — некоторое связное множество пикселов. По способам доступа к соседним пикселам области делятся на 4-х и 8-ми связные. В 4-х связных областях доступ к соседним пикселам осуществляется по четырем направлениям — горизонтально влево и вправо и в вертикально вверх и вниз. В 8-ми связных областях к этим направлениям добавляются еще 4 диагональных. Используя связность мы можем, двигаясь от точки затравки по направлениям связности, достичь и закрасить все пикселы области.

Важно отметить, что для 4-х связной области достаточно 8-ми связной границы (рис. 4.1, а), а для 8-ми связной области граница должна быть 4-х связна (см.

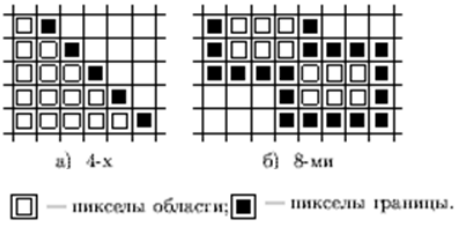

Рис. 4.1. 4-х и 8-ми связные области.

рис. 4.1, б). Поэтому заполнение 4-х связной области 8-ми связным алгоритмом может привести к «просачиванию» через границу и заливке пикселов в примыкающей области.

4.1.1. Простой алгоритм заливки

Рассмотрим простой алгоритм заливки гранично-определенной 4-х связной области (алгоритм 1).

Заливка выполняется следующим образом:

- определяется является ли пиксел граничным или уже закрашенным,
- если нет, то пиксел перекрашивается, затем проверяются и если надо перекрашиваются 4 соседних пиксела.

На рис. 4.2 показан выбранный порядок перебора соседних пикселов, а на рис. 4.3

Алгоритм 1: Простой алгоритм заливки

Вход: P — закрашиваемая область, C_{gr} — цвет границы области, C_{zal} — цвет заливки области.

начало алгоритма
- Пусть $S = \varnothing$ — пустой стек, в котором будем сохранять точки;
- Определить координаты (x_i, y_i) внутренней точки области;
- Закрасить точку (x_i, y_i) цветом C_{zal};
- Положить точку (x_i, y_i) в стек S;
- **цикл пока** $S \neq \varnothing$ **выполнять**
  - Взять (забрать) точку (x_j, y_j) из стека S;
  - **цикл для** $(x, y) \in \{(x_j + 1, y_j), (x_j, y_j + 1), (x_j - 1, y_j), (x_j, y_j - 1)\}$ **выполнять**
    - Определить C_{xy} — цвет точки (x, y);
    - **если** $C_{xy} \neq C_{gr}$ *и* $C_{xy} \neq C_{zal}$ **то**
      - Закрасить точку (x, y);
      - Положить точку (x, y) в стек S;

конец алгоритма

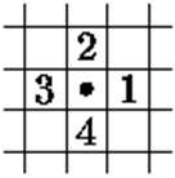

Рис. 4.2. Порядок перебора соседних точек.

соответствующий ему порядок закраски простой гранично-определенной области.

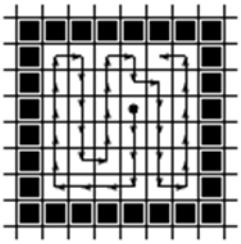

Рис. 4.3. Порядок закраски области.

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

Очевидный недостаток алгоритмов непосредственно использующих связность за-

крашиваемой области — большие затраты памяти на стек. Проверка цвета некоторых пикселов может проводиться многократно. Это приводит к потере быстродействия. Несколько более экономен далее рассматриваемый построчный алгоритм заливки.

4.1.2. Построчный алгоритм заливки с затравкой

Использует пространственную когерентность:

- пикселы в строке меняются только на границах;
- при перемещении к следующей строке размер заливаемой строки скорее всего или неизменен или меняется на 1 пиксел.

Таким образом, на каждый закрашиваемый фрагмент строки в стеке хранятся координаты только одного начального пиксела, 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 вправо и влево по строке с затравкой, т.е. пока не попадется граничный пиксел. Пусть это x_l и x_r, соответственно.
3. Анализируется строка ниже закрашиваемой в пределах от x_l до x_r и в ней находятся крайние правые пикселы всех незакрашенных фрагментов. Их координаты заносятся в стек.
4. То же самое проделывается для строки выше закрашиваемой.

Более детально метод излагается в алгоримтме 2

Алгоритм 2: Построчный алгоритм заливки области

Вход: P — закрашиваемая область, C_{gr} — цвет границы области, C_{zal} — цвет заливки области.

начало алгоритма
- Пусть $S = \varnothing$ — пустой стек, в котором будем сохранять точки;
- Определить координаты (x_i, y_i) внутренней точки области;
- Положить точку (x_i, y_i) в стек S;
- **цикл пока** $S \neq \varnothing$ **выполнять**
  - Взять (забрать) точку (x_j, y_j) из стека S;
  - Определить C_{xy} — цвет точки (x_j, y_j). **если** $C_{xy} \neq C_{zal}$ **то**
    - $x_{\min} = x_j$; $x_{\max} = x_j$;
    - **повторять вычисления**
      - $x_{\min} = x_{\min} - 1$;
      - Определить $C_{\min}$ — цвет точки $(x_{\min}, y_j)$;
    - **пока** $C_{\min} \neq C_{gr}$;
    - **повторять вычисления**
      - $x_{\max} = x_{\max} + 1$;
      - Определить $C_{\max}$ — цвет точки $(x_{\max}, y_j)$;
    - **пока** $C_{max} \neq C_{gr}$;
    - Начертить линию от $(x_{\min} + 1, y_j)$ до $(x_{\max} - 1, y_j)$ цветом C_{zal};
    - **цикл для каждого** x *от* $x_{\min} + 1$ *до* $x_{\max} - 1$ **выполнить**
      - Определить C_{up} — цвет точки $(x, y_j + 1)$;
      - **если** $C_{up} \neq C_{gr}$ *и* $C_{up} \neq C_{zal}$ **то**
        - Положить точку $(x, y_j + 1)$ в стек S;
      - Определить C_{dn} — цвет точки $(x, y_j - 1)$;
      - **если** $C_{dn} \neq C_{gr}$ *и* $C_{dn} \neq C_{zal}$ **то**
        - Положить точку $(x, y_j - 1)$ в стек S;

конец алгоритма

4.1.3. Заполнение многоугольника

В данном разделе рассмотрим алгоритм заполнения многоугольника.

Простейший способ заполнения многоугольника, заданного координатами вершин, заключается в определении принадлежит ли текущий пиксел внутренней части многоугольника. Если принадлежит, то пиксел заносится. Определить принадлежность пиксела многоугольнику можно, например, подсчетом суммарного угла с вершиной на пикселе при обходе контура многоугольника. Если пиксел внутри, то угол будет равен 360 , если вне - 0.

Вычисление принадлежности должно производиться для всех пикселов экрана и так как большинство пикселов скорее всего вне многоугольников, то данный способ слишком расточителен. Объем лишних вычислений в некоторых случаях можно сократить использованием прямоугольной оболочки — минимального прямоугольника, объемлющего интересующий объект, но все равно вычислений будет много.

4.1.4. Построчное заполнение многоугольника

Реально используются алгоритмы построчного заполнения, основанные на том, что соседние пикселы в строке скорее всего одинаковы и меняются только там где строка пересекается с ребром многоугольника. Это свойство называется когерентностью растровых строк (строки сканирования y_i, y_{i+1}, y_{i+2} на рис. 4.4). При этом достаточно определить x-координаты пересечений строк сканирования с ребрами. Пары отсортированных точек пересечения задают интервалы заливки.

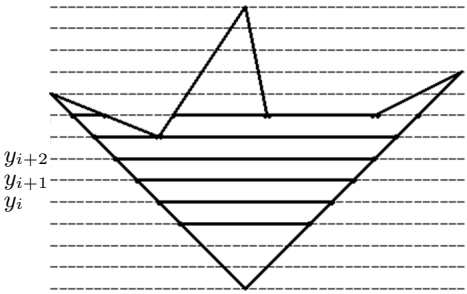

Рис. 4.4. Построчное заполнение многоугольника.

Кроме того, если какие-либо ребра пересекались i-й строкой, то они скорее всего будут пересекаться также и строкой $i+1$. (строки сканирования y_i и y_{i+1} на рис. 4.4). Это называется когерентностью ребер. При переходе к новой строке легко вычислить новую x-координату точки пересечения ребра, используя x-координату старой точки пересечения и тангенс угла наклона ребра:

$$x_{i+1} = x_i + \frac{1}{k},$$

где $k = \dfrac{\Delta y}{\Delta x}$ — тангенс угла наклона ребра, так как $y_{i+1} = y_i + 1$.

Смена же количества интервалов заливки происходит только тогда, когда в строке сканирования появляется вершина.

Учет когерентности строк и ребер позволяет построить для заполнения многоугольников различные высокоэффективные алгоритмы построчного сканирования. Для каждой строки сканирования рассматриваются только те ребра, которые пересекают строку. Они задаются списком активных ребер (*AEL*). При переходе к следующей

строке для пересекаемых ребер перевычисляются x-координаты пересечений. При появлении в строке сканирования вершин производится перестройка AEL. Ребра, которые перестали пересекаться, удаляются из AEL, а все новые ребра, пересекаемые строкой заносятся в него.

Общая схема алгоритма, динамически формирующего список активных ребер и заполняющего многоугольник снизу-вверх, следующая:

1. Подготовить служебные целочисленные массивы y-координат вершин и номеров вершин.
2. Совместно отсортировать y-координаты по возрастанию и массив номеров вершин для того, чтобы можно было определить исходный номер вершины.
3. Определить пределы заполнения по оси Oy — y_{min} и y_{max}. Стартуя с текущим значением $y_t = y_{min}$, исполнять пункты 4-9 до завершения раскраски.
4. Определить число вершин, расположенных на строке y_t — текущей строке сканирования.
5. Если вершины есть, то для каждой из вершин дополнить список активных ребер, используя информацию о соседних вершинах.

   Для каждого ребра в список активных ребер заносятся:

   - начальное значение x-координаты,
   - максимальное значение y-координаты ребра,
   - тангенс угла наклона ребра.

   Если обнаруживаются горизонтальные ребра, то они просто закрашиваются и информация о них в список активных ребер не заносится.

   Если после этого обнаруживается, что список активных ребер пуст, то заполнение закончено.
6. По списку активных ребер определяется y_{next} — y-координата ближайшей вершины. (Вплоть до y_{next} можно не заботиться о модификации AEL а только менять x-координаты пересечений строки сканирования с активными ребрами).
7. В цикле от y_t до y_{next}:

   а) выбрать из списка активных ребер и отсортировать x-координаты пересечений активных ребер со строкой сканирования;

б) определить интервалы и выполнить закраску;

в) перевычислить координаты пересечений для следующей строки сканирования.

8. Проверить не достигли ли максимальной y-координаты. Если достигли, то заливка закончена, иначе выполнить пункт 9.
9. Очистить список активных ребер от ребер, закончившихся на строке y_{next} и перейти к пункту 4.

Более детально метод изложен в алгоритме 3.

Алгоритм 3: Алгоритм построчной заливки многоугольника

Вход: P — список вершин многоугольника (без самопересечений), C_{zal} — цвет закраски.
начало алгоритма
- Сформировать список S ребер многоугольника, где для каждого ребра $[(x_1, y_1), (x_2, y_2)]$ выполняется $y_1 \leqslant y_2$;
- Упорядочить список S по возрастанию значения y_1;
- Найти y_{min} и y_{max} — минимальное и максимальное значение координаты y точек вершин многоугольника;
- $AEL = \varnothing$; $y_t = y_{min}$; $y_{S\,next} = y_{min}$;

цикл пока $y_t \leqslant y_{max}$ **выполнять**
- **если** $y_t = y_{S\,next}$ **то**
  - Добавить в AEL все тройки $\left(x_1, y_2, \dfrac{x_2 - x_1}{y_2 - y_1}\right)$, составленные для каждого отрезка из S, у которого $y_1 = y_t$ и $y_1 \neq y_2$;
  - Начертить все ребра в S, у которых $y_1 = y_t$ и $y_1 = y_2$;
  - Удалить все ребра в S, у которых $y_1 = y_t$;
  - Для отрезков в S найти $y_{S\,next}$ — минимальное значение y_1 у отрезков в S;
  - Отсортировать AEL по возрастанию первого элемента и по возрастанию третьего элемента;
  - Найти $y_{AEL\,next}$ — минимальное значение второго элемента, отличное от y_t;
- **цикл для** i *от* 1 *до* $|AEL|$ *с шагом* 2 **выполнять**
  - Начертить отрезок $[(x_i, y_t), (x_{i+1}, y_t)]$ цветом C_{zal}, где $(x_i, y_i, \Delta_i x)$ обозначает i-й элемент списка AEL;
- В каждой тройке $(x_j, y_j, \Delta_j x)$ в AEL заменить x_j на $x_j + \Delta_j x$;
- $y_t = y_t + 1$;
- **если** $y_t \geqslant y_{AEL\,next}$ **то**
  - Удалить из AEL все тройки, второй элемент которых меньше или равен y_t;
  - Найти $y_{AEL\,next}$ — минимальное значение второго элемента, отличное от y_t;

конец алгоритма