Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П.Г. Демидова
Кафедра информационных и сетевых технологий

ОСНОВНЫЕ АЛГОРИТМЫ РАСТРОВОЙ ГРАФИКИ

Ярославль 2004

ŋqdqɏeɎ bɆ dedɆbɎɜe d ɕɔɑbɔɏ Ɉqddɏqɏè

ББК В 185я73
О 75
УДК 002:372.8

Составитель: **Н.В. Легков**

Основные алгоритмы растровой графики: Метод. указания / Сост. Н.В. Легков; Яросл. гос. ун-т. Ярославль, 2004. 19 с.

Приводятся описания алгоритмов машинной графики. Рассмотрены алгоритмы закраски областей. Приведены алгоритмы Брезенхема и ЦДА для изображения отрезков. Рассмотрен алгоритм Брезенхема генерации окружности.

Пособие предназначено для студентов, обучающихся по специальности 351500 Математическое обеспечение и администрирование информационных систем (дисц. “Компьютерная графика”, блок ОПД), очной формы обучения.

Рецензент – кафедра информационных и сетевых технологий ЯрГУ.

Введение

В пособии будут описаны простейшие алгоритмы растровой графики, такие как генерация отрезка прямой, окружности или эллипса, алгоритма закраски многоугольника или, в более общем случае, области плоскости.

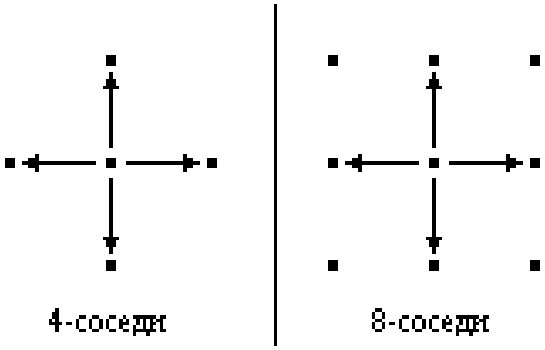
Практически любой современный язык программирования имеет стандартные процедуры, которые решают эти задачи (не говоря о том, что для Windows набор таких алгоритмов является просто «системными» функциями). В связи с этим возникает вопрос для чего вообще необходимо изучать эти алгоритмы? Но в любом деле следует начинать с чего-то, и если Вы собираетесь в дальнейшем решать задачи связанные с графикой, то знать, как реализуются простейшие из алгоритмов, просто необходимо.

Мы будем изучать алгоритмы в общем виде, поэтому определимся с некоторыми условиями нашей работы. Будем считать, что устройством, на которое происходит вывод графики, является массив *Screen: array [0.. ScW-1,0..ScH-1] of TColor*. Здесь *TColor* означает, что элементы массива задают цвет соответствующей точки устройства. Для задания цвета мы будем использовать функцию *RGB(bR,bG,bB:byte)*, параметры которой задают соотношения красного, зеленого и синего цвета, например, такая запись: *Screen[12,15]:=RGB(255,255,255)* будет означать, что мы "закрашиваем" точку с координатами *(12,15)* белым цветом. Будем считать, что, если нам необходимо, мы можем не только присваивать значение элементу массива, но и считывать цвет точки с произвольными координатами. В случае некоторого "реального" устройства будем считать, что точке с координатами *(0,0)* соответствует левый верхний угол; при этом увеличение первой координаты означает движение точки по горизонтали, а второй - по вертикали.

Прежде чем перейти непосредственно к алгоритмам генерации растровых изображений, определим еще несколько понятий, которые в дальнейшем будут нам полезны.

Назовем точки 4-соседями (или "непосредственными" соседями), если у них отличается только одна из координат, и притом только на 1. Назовем точки 8-соседями (или "косвенными" соседями), если у них отличается горизонтальная или вертикальная координата, но не более чем на 1. Нетрудно заметить, что всякий непосредственный со-

сед является также и косвенным соседом. Всякая точка имеет 4 непосредственных и 8 косвенных соседей.



Итак, вводная часть закончена, и мы можем перейти к рассмотрению алгоритмов. Начнем мы, естественно, с главного, без чего нельзя обойтись при работе с графикой, а именно - с построения отрезка, заданного своими концами.

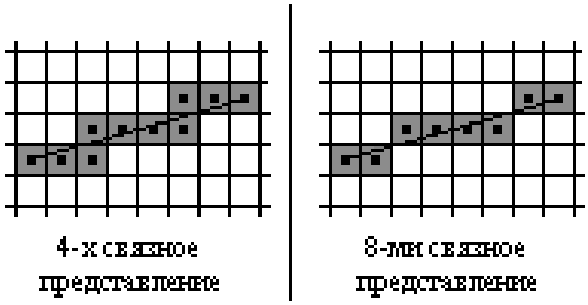
1. Алгоритмы генерации растрового представления отрезка

Для начала выскажем общие соображение относительно построения отрезка на экране.

Допустим, нам даны координаты начала (x_1,y_1) и конца (x_2,y_2) отрезка, тогда уравнение прямой, содержащей отрезок, может быть, записано в виде $y=y_1+k(x-x_1)$, где $k=(y_2-y_1)/(x_2-x_1)$, а x пробегает значения от x_1 до x_2 (мы пока исключаем из рассмотрения случай вертикального отрезка). Поскольку растровое изображение отрезка содержит только точки с целочисленными координатами, то x будет пробегать только целые значения из отрезка (x_1,x_2). И в случае $|k| < 1$ все прекрасно: изменяем значение x, по уравнению прямой высчитываем значение y, "закрашиваем" точку с координатами (x,y), переходим к началу цикла. Но не трудно, заметить, что в случае, когда $|k| > 1$, наше представление отрезка будет иметь пробелы, так как изменение x на единицу приведет к изменению y более чем на единицу. Поэтому в случае $|k| > 1$ переменные x и y надо поменять ролями.

Итак, мы практически получили искомый алгоритм, но вот только x и y могут (а в общем случае и будут) принимать вещественные значения, что для нас совершенно недопустимо. Поэтому нам надо ввести некоторое соглашение о том, каким образом мы будем "округлять" вещественные значения к целым. В связи с вопросом округления вернемся к определению 4- и 8-связности.

Ниже представлены два алгоритма Брезенхема получения 4- и 8-связного представления отрезка, заданного координатами концов. Разницу между двумя этими представлениями можно увидеть на рисунке:



И тот, и другой алгоритмы используют только целочисленную арифметику, что, естественно, улучшает скорость работы. Саму структуру алгоритмов можно понять из блок-схем, которые приведены в приложениях 1 и 2. Различие в работе понятно и связано с разницей между 4- и 8-связным представлением: в случае 4-связности на каждом шаге мы изменяем на единицу только одну из координат текущей точки, а в случае 8-связности мы можем менять сразу и вертикальную и горизонтальную координаты текущей точки, но снова не более чем на единицу.

2. Алгоритм закраски многоугольников

Под многоугольником далее будем понимать фигуру, ограниченную на плоскости простой (непересекающейся) замкнутой ломаной. Сама ломаная задается координатами вершин *Ai(xi,yi), i=1..n*, при этом соседние точки в этом списке являются вершинами ломаной.

Задача "закраски" многоугольника заключается в инициализации всех его внутренних точек.

Наиболее простой, но и наиболее медленный алгоритм, решающий эту задачу, состоит в том, чтобы проверить все точки плоскости (под плоскостью здесь, естественно, понимается устройство, на которое осуществляется вывод, в нашем случае это массив *Screen*) на принадлежность многоугольнику. Несколько ускорить этот алгоритм можно, если вместо проверки всех точек плоскости проверять только точки минимального квадрата, заключающего многоугольник, найти который не составляет труда. Ясно, что и при таком усовершенствовании скорость работы алгоритма оставляет желать лучшего.

Еще один вариант решения задачи - это алгоритм сканирования строк. Последовательно проходим все горизонтальные линии плоскости и находим отрезки на прямой, внутренние для многоугольника. Для данного алгоритма достаточно искать только точки пересечения сканирующей линии с ребрами многоугольника. Для оптимизации алгоритма можно упорядочить ребра в порядке возрастания наибольшей из ординат концов. При перемещении сканирующей прямой сверху вниз проверке на пересечение подвергаются лишь те ребра, у которых значение максимальной ординаты больше ординаты сканирующей прямой. Такой алгоритм пригоден для заполнения произвольных многоугольников. Но если многоугольник выпуклый, то алгоритм можно упростить и существенно повысить его эффективность. Для этого заметим, что границу выпуклого многоугольника можно разбить на две ломаные: "левую" и "правую" и, возможно, два ребра: "верхнее" и "нижнее", так что каждая из боковых ломаных имеет ровно одно пересечение с каждой сканирующей прямой. Далее, используя алгоритм Брезенхема и одновременно генерируя растровое представление для ребер левой и правой части ломаных границ, получаем левый и правый пиксели границы многоугольника на каждой сканирующей горизонтальной прямой. Последовательно заполняя интервалы между этими пикселями для каждого значения ординаты от верхней строки развертки до нижней, получим растровое представление многоугольника. Один частный случай этого алгоритма получил широкое распространение в связи с задачами трехмерной графики: это закраска треугольника. В этом случае можно существенно упростить алгоритм.

3. Алгоритм закраски произвольной области с затравкой

Рассмотрим еще один класс алгоритмов "закраски", а именно алгоритмы заполнения области с затравкой. В этих алгоритмах предполагается, что граница области задана на растровой плоскости и указана одна из внутренних точек области, которая называется затравочной. Требуется заполнить определенным цветом связную компоненту области, содержащую затравочную точку. Под связностью будем понимать 4- или 8-связности, определенные выше (какая связность конкретно используется, зависит от формулировки задачи).