Введение в машинную графику и вычислительную геометрию

Сергей Карпухин

13 июня 2010 г.

1 Машинная графика

1.1 Растеризация простейших объектов

1.1.1 Вводные понятия

Для того, чтобы описать программу построения фигуры, нам понадобится модель устройства вывода. В качестве такой модели чаще всего принимается растр.

Определение. *Растр*(двумерный) - отображение R: $I_1 \times I_2 \to \Omega$, где $I_1 \subset \mathbb{Z}$, $I_2 \subset \mathbb{Z}$, Ω — произвольное множество, называемое *множеством атрибутов*. Элемент растра называется *пикселем*.

Изображение на мониторе представляет собой таблицу точек, каждая из которых закрашена определённым цветом, поэтому оно является растром, для которого I_1, I_2 — целочисленные отрезки, а Ω — множество доступных цветов.

Далее будем использовать простые растры, у которых I_1, I_2 — отрезки, а $\Omega = \{0, 1\}$. Такие растры соответствуют одноцветным изображениям. Их можно представлять либо как наборы точек с целочисленными координатами, либо как чёрные и белые квадраты на плоскости. В каждом случае будем выбирать более удобное представление.

При написании алгоритмов будем использовать функцию закрашивания точки putpixel(i, j). После выполнения этой функции $R(i, j) = 1$, где R — используемый растр. В начале работы программы предполагается, что ни одна точка не закрашена.

Определение. *Растеризация* — поиск растра, изображающего заданный аналитически объект.

1.1.2 Растеризация отрезка

Пусть концы отрезка (x_1, y_1) и (x_2, y_2) находятся в точках растра.

Утверждение 1. *Задачу достаточно рассмотреть при* $x1 = 0$, $y1 = 0$, $x_2 > 0$, $y_2 > 0$, $y_2 < x_2$.

▶ Сделаем замену координат. $x = ax+by+x_1$, $y = cx+dy+y_1$, где $a, b, c, d \in \{-1, 0, 1\}$. При такой замене точки растра перейдут в точки растра, поэтому вызовы putpixel будут корректными. Коэффициенты можно подобрать так, чтобы выполнились условия утверждения. ■

Воспользуемя этим утверждением и будем последовательно строить отрезок от начала $(0, 0)$ до конца (x_0, y_0), т.е. будем закрашивать пиксели, располагающиеся близко к заданному отрезку. $y_0 < x_0$, поэтому угол наклона отрезка меньше $\frac{\pi}{4}$ и после пиксела с координатами (x, y) может следовать только $(x + 1, y)$ или $(x + 1, y + 1)$. Переход $(x, y) \to (x + 1, y)$ назовём горизонтальным, а $(x, y) \to (x + 1, y + 1)$ — диагональным.

Идея первого алгоритма в том, что на каждом шаге вычисляется ошибка e и по ней определяется, какая из двух возможных точек будет следующей, где e — расстояние от $(x + 1, y)$ до $(x + 1, y')$, и $(x + 1, y')$ лежит на прямой, содержащей отрезок. Тогда расстояние от $(x + 1, y + 1)$ до $(x + 1, y')$ есть $1 - e$. Т.к. угол наклона отрезка постоянен, аналогичные, с точностью до одинакового множителя, соотношения имеют место для расстояний от отрезка до точек $(x+1, y)$ и $(x+1, y+1)$. Тогда условием горизонтального перехода будет $e < \frac{1}{2}$, в противном случае производится диагональный переход. На каждом шаге e не вычисляется снова, а модифицируется приращением Δe в соответствии со сделанным переходом, что существенно снижает необходимое количество операций.

Получаем алгоритм, называемый «Цифровой дифференциальный анализатор» (Digital Differential Analyser — DDA).

Алгоритм 1 Цифровой дифференциальный анализатор

Входные данные: (x_0, y_0) — конец отрезка, $y_0 < x_0$.
Результат: На растре строится изображение отрезка с концами $(0, 0)$ и (x_0, y_0).

```
x ← 0, y ← 0, e ← 0, Δe ← y0/x0
putpixel(x, y)
while x < x0 do
  e ← e + Δe
  x ← x + 1
  if e > 1/2 then
    y ← y + 1
    e ← e − 1
  end if
  putpixel(x, y)
end while
```

У алгоритма DDA есть существенный недостаток: e и Δe — действительные числа, а действия с действительными числами достаточно трудоёмки. Для преодоления этого недостатка заменим e на $2x_0\left(e - \frac{1}{2}\right)$ и Δe на $2x_0\Delta e$. Получим целочисленный алгоритм, называемый «алгоритмом Брезенхэма для отрезка» (Bresenham's line

algorithm).

Алгоритм 2 Алгоритм Брезенхэма для отрезка

Входные данные: (x_0, y_0) — конец отрезка, $y_0 < x_0$.
Результат: На растре строится изображение отрезка с концами $(0, 0)$ и (x_0, y_0).

```
x ← 0, y ← 0, e ← −x_0, Δe ← 2y_0
putpixel(x, y)
while x < x_0 do
  e ← e + Δe
  x ← x + 1
  if e > 0 then
    y ← y + 1
    e ← e − 2x_0
  end if
  putpixel(x, y)
end while
```

В случае, когда координаты концов отрезка не попадают в точки растра, можно их округлить или воспользоваться общим векторным алгоритмом, допускающим обобщение на многомерные растры.

В основе работы векторного алгоритма лежит параметрическая запись отрезка

$$\begin{aligned} x &= x_1 + (x_2 - x_1)t, \\ y &= y_1 + (y_2 - y_1)t, \\ t &\in [0, 1]. \end{aligned}$$

Определим h и v — расстояния вдоль отрезка от текущей точки до ближайшего следующего пересечения с сеткой растра по горизонтали и вертикали соответственно. Переход производится по горизонтали или вертикали к тому пересечению, которое произойдёт раньше. Аналогично, h и v модифицируются на каждом шаге с помощью приращений Δh, Δv. Построение закончено, если $h > 1$ или $v > 1$, это означает, что по одному из направлений отрезок уже пройден. Алгоритм приведём для случая $x_1 < x_2$, $y_1 < y_2$. Остальные случаи сводятся к этому заменой знаков приращений координат и расстояний h, v. В этом алгоритме h, v, Δh, Δv — действительные числа. Можно сделать его целочисленным, если вместо t взять параметр $\frac{t}{c}$ при достаточно большом c.

1.1.3 Растеризация окружности

Пусть окружность задаётся координатами центра (x_0, y_0) и радиусом R.

Утверждение 2. *Достаточно построить окружность с центром* $(0, 0)$ *и радиусом* R *в области* $D_1 = \{x > 0,\ y > 0,\ y < x\}$.

Алгоритм 3 Векторный алгоритм построения отрезка

Входные данные: (x_1, y_1), (x_2, y_2) — концы отрезка, $x_1 < x_2$, $y_1 < y_2$.
Результат: На растре строится изображение отрезка с концами (x_1, y_1) и (x_2, y_2).

```
x ← [x1], y ← [y1]
Δh ← 1/(x2−x1), Δv ← 1/(y2−y1)
h ← Δh (1 − {x1}), v ← Δv (1 − {y1})
while (h < 1 and v < 1) do
  putpixel(x, y)
  if h < v then
    x ← x + 1
    h ← h + Δh
  else
    y ← y + 1
    v ← v + Δv
  end if
end while
```

► Аналогично утверждению 1. Воспользуемся переносом центра и симметрией, переводящей точки растра в точки растра. ■

Искать изображение окружности будем аналогично алгоритму DDA. В D_1 угол наклона касательной к окружности(с учётом направления) лежит в $\left[\frac{\pi}{2}, \frac{3\pi}{4}\right]$, поэтому возможны только 2 перехода к следующей точке: $(x, y) \to (x, y+1)$ и $(x, y) \to (x-1, y+1)$, которые будем называть вертикальным и диагональным соответственно. Обозначим $y_n = y + 1$, $x_d = x - 1$. Пусть R_v, R_d — расстояния от следующей точки до центра окружности при вертикальном и диагональном переходах соответственно, ΔR_v, ΔR_d — расстояния от соотвестствующих точек до окружности. Тогда

$$\Delta R_v = R_v - R = \sqrt{x^2 + y_n^2} - R,\ \Delta R_d = R - R_d = R - \sqrt{x_d^2 + y_n^2},$$

и нужно выбирать диагональный переход при $\Delta R_d < \Delta R_v$. Вычисление корня — трудоёмкая операция, поэтому попытаемся заменить расстояния их квадратами.

Утверждение 3. *При* $R \geqslant 1$

1. $R_v^2 - R^2 < R^2 - R_d^2 \Rightarrow \Delta R_v < \Delta R_d$;
2. $R_v^2 - R^2 > R^2 - R_d^2 \Rightarrow \Delta R_v > \Delta R_d - 1$.

►

1. От противного: пусть $\Delta R_v > \Delta R_d$. Тогда

$$0 > R_v^2 - R^2 - R^2 - R_d^2 = \Delta R_v(R_v + R) - \Delta R_d(R + R_d) > \Delta R_d(R_v - R_d) > 0.$$

Противоречие.

2. От противного: пусть $\Delta R_v < \Delta R_d - 1$. Тогда аналогично

$$0 < \Delta R_v(R_v + R) - \Delta R_d(R + R_d) < \Delta R_d(R_v - R_d) - (R_v + R) < 0,$$

т. к. $\Delta R_d < 1$, $R_v - R_d < 2$, $R_v + R \geqslant 2$.

■

Таким образом, замена ΔR_v и ΔR_d на $F_v = R_v^2 - R^2$ и $F_d = R^2 - R_d^2$ соответственно даёт ошибку при построении не более одного пиксела, что вполне допустимо. Поэтому критерием выбора диагонального перехода будет $F_v > F_d$ или $F > 0$, где

$$F = F(x, y) = F_v - F_d = x^2 + y_n^2 + x_d^2 + y_n^2 - 2R^2.$$

Найдём, как изменяется F после каждого из переходов:

$$\begin{aligned}
\Delta_v F &= F(x, y+1) - F(x, y) = x^2 + (y+2)^2 + (x-1)^2 + (y+2)^2 - \\
&\quad - \left(x^2 + (y+1)^2 + (x-1)^2 + (y+1)^2\right) = 2\,(4y + 4 - 2y - 1) = 4y + 6; \\
\Delta_d F &= F(x-1, y+1) - F(x, y) = (x-1)^2 + (y+2)^2 + (x-2)^2 + (y+2)^2 - \\
&\quad - \left(x^2 + (y+1)^2 + (x-1)^2 + (y+1)^2\right) = -4x + 4y + 10.
\end{aligned}$$

Эти значения легко вычисляются на каждом щаге с помощью приращений. На первом шаге ($x = R$, $y = 0$):

$$F = R^2 + 1 + R^2 - 2R + 1 + 1 - 2R^2 = 3 - 2R;\ \Delta_v F = 6;\ \Delta_d F = 10 - 4R.$$

Получаем алгоритм построения окружности. Заметим, что в полученном алгоритме все приращения кратны 2, поэтому можно разделить их на 2 и сдвинуть F на $\frac{1}{2}$, чтобы уменьшить возможность переполнения переменных.

1.2 Кривые Безье

1.2.1 Построение кривых Безье

Определение. $B_{k,n} = \binom{n}{k} x^k (1-x)^{n-k}$ — k-й базисный многочлен Бернштейна степени n, $k = 0..N$.

Определение. $P^n(t) = \sum_{i=0}^{n} P_i B_{i,n}$ — параметрическая кривая Безье n-го порядка, построенная по опорным точкам P_i, $i = 0..n$. Равенство понимается для радиус-векторов точек.

Пусть заданы опорные точки $P_i^0 = P_i$, $i = 0..n$ кривой Безье. Зададим некоторое $t \in [0, 1]$ и разделим каждый отрезок $[P_i^0, P_{i+1}^0]$ точкой P_i^1 в отношении t, т.е. $P_i^1 = (1-t)P_i^0 + tP_{i+1}^0$, $i = 0..n-1$. Далее повторим процесс, взяв в качестве опорных новые точки: $P_i^k = (1-t)P_i^{k-1} + tP_{i+1}^{k-1}$, $i = 0..n-k$. Таким образом, при $k = n$ получим единственную точку P_0^n.

Алгоритм 4 Алгоритм построения окружности

Входные данные: R — радиус.

Результат: На растре в области D_1 строится изображение окружности $x^2 + y^2 = R^2$.

```
x ← R, y ← 0
F = 3 − 2R, Δ_v F = 6, Δ_d F = 10 − 4R.
while x ⩾ y do
  putpixel(x, y)
  if F>0 then
    F ← F + Δ_d F
    x ← x − 1
    Δ_d F ← Δ_d F + 4
  else
    F ← F + Δ_v F
  end if
  y ← y + 1
  Δ_d ← Δ_d + 4
  Δ_v ← Δ_v + 4
end while
```

Лемма 1.

$$(1-t)B_{i,n-1} + tB_{i-1,n-1} = B_{i,n}$$

►

$$(1-t)B_{i,n-1} + tB_{i-1,n-1} = (1-t)\binom{n-1}{i}t^i(1-t)^{n-1-i} + t\binom{n-1}{i-1}t^{i-1}(1-t)^{n-i} =$$
$$= \left(\binom{n-1}{i} + \binom{n-1}{i-1}\right)t^i(1-t)^{n-i} = \binom{n}{i}t^i(1-t)^{n-i} = B_{i,n}.$$

■

Утверждение 4.

$$P_0^n = P^n = \sum_{i=0}^{n} P_i B_{i,n}$$

► Индукция по n:

1. База. $P_0^0 = P_0 = P_0 B_{0,0} = P^0$.

2. Переход.

$$P_0^n = (1-t)P_0^{n-1} + tP_1^{n-1} = (\text{предположение индукции}) =$$
$$= (1-t)\sum_{i=0}^{n-1} P_i B_{i,n-1} + t\sum_{i=0}^{n-1} P_{i+1} B_{i,n-1} =$$
$$= (1-t)P_0 B_{0,n-1} + \sum_{i=1}^{n-1} P_i \left(B_{i,n-1} + B_{i-1,n-1}\right) + tP_n B_{n-1,n-1} = (\text{Лемма 1}) =$$
$$= P_0 B_{0,n} + \sum_{i=1}^{n-1} P_i B_{i,n} + P_n B_{n,n} = \sum_{i=0}^{n} P_i B_{i,n} = P^n.$$

■

Из этого утверждения следует, что построенная методом деления точка лежит на кривой Безье P^n.

Теперь рассмотрим кривую Безье $Q_-^n(s)$, построенную по точкам $Q_{-,i} = P_0^i(t)$, определённым ранее.

Лемма 2.

$$\binom{i}{j}\binom{n}{i} = \binom{n}{j}\binom{n-j}{i-j}.$$

►

$$\binom{i}{j}\binom{n-j}{i-j} = \frac{i!}{j!(i-j)!}\cdot\frac{n!}{i!(n-i)!} = \frac{n!}{j!(n-j)!}\cdot\frac{(n-j)!}{(i-j)!(n-i)!} = \binom{n}{j}\binom{n-j}{i-j}.$$

■

Лемма 3.

$$\sum_{i=j}^{n} B_{j,i}(t)B_{i,n}(s) = B_{j,n}(ts).$$

►

$$\sum_{i=j}^{n} B_{i,j}(t)B_{i,n}(s) = \sum_{i=j}^{n}\binom{i}{j}t^j(1-t)^{i-j}\binom{n}{i}s^i(1-s)^{n-i} = (\text{Лемма 2}) =$$
$$= (ts)^j\sum_{i=j}^{n}\binom{n}{j}\binom{n-j}{i-j}(s-ts)^{i-j}(1-s)^{n-i} =$$
$$= \binom{n}{j}(ts)^j\sum_{i=0}^{n-j}\binom{n-j}{i}(s-ts)^i(1-s)^{n-j-i} =$$
$$= \binom{n}{j}(ts)^j(1-s+s-ts)^{n-j} = B_{j,n}(ts).$$

■

Утверждение 5.

$$Q_-^n(s) = P^n(st)$$

▶

$$Q_-^n(s) = \sum_{i=0}^{n} P_0^i(t) B_{i,n}(s) = (\text{Утверждение 4}) = \sum_{i=0}^{n}\sum_{j=0}^{i} P_j B_{j,i}(t) B_{i,n}(s) =$$
$$= \sum_{j=0}^{n} P_j \left(\sum_{i=j}^{n} B_{j,i} Bi, n \right) = (\text{Лемма 3}) = \sum_{j=0}^{n} P_j B_{j,n}(st) = P^n(st).$$

■

Аналогичное утверждение верно для кривой $Q_+^n(s)$, построенной по точкам $Q_{+,i} = P_i^{n-i}$.

Из двух полученных утверждений следует, что точка $P_0^n(t)$ делит кривую Безье на две части, каждая из которых является кривой Безье, построенной по соответствующим промежуточным точкам. Если выбрать $0 < t < 1$, то длина каждой части будет меньше длины всей кривой. Это наблюдение позволяет строить кривые Безье рекурсивно, т. е. делить каждую часть до тех пор, пока все точки не будут лежать внутри одного пикселя, который и выводится на растр. Обычно при этом берут $t = \frac{1}{2}$, чтобы в большинстве случаев деление происходило примерно пополам и длина убывала экспоненциально. Пусть далее $P.x$, $P.y$ — координаты точки. Получаем «алгоритм де Кастельё(de Casteljau) построения кривой Безье».

1.2.2 Сплайны на кривых Безье

Определение. Сплайн на кривых Безье, проходящий через точки A_i, $i = 0..k$ — множество кривых Безье $B_i^N(t)$, $i = 0..k-1$ N-го порядка, таких что $B_i^N(0) = A_i$, $B_i^N(1) = A_{i+1}$, т.е. сплайн проходит через заданные точки и между точками является кривой Безье.

Чаще всего используются кривые Безье 3-го порядка в силу простоты и наглядности построения. На сплайн могут накладываться дополнительные условия гладкости в точках P_i. Для построения кривой Безье 3-го порядка нужно 4 опорных точки. Поскольку B_i^3 проходит через A_i и A_{i+1}, то концевые точки определены. Зададим ещё 2 опорных точки C_i, D_i. Выясним уловия на C_i, D_i, обеспечивающие C^1-гладкость. Для этого найдём в точках A_i, A_{i+1} производные B_i^3 как функции $y_i(x_i)$, где $x_i(t) = B_i^3(t).x$, $y_i(t) = B_i^3(t).y$.

$$B_i^3(t) = A_i B_{0,3}(t) + C_i B_{1,3}(t) + D_i B_{2,3}(t) + A_{i+1} B_{3,3}(t);$$
$$B'_{0,3}(t) = \left((1-t)^3\right)' = -3(1-t)^2;$$
$$B'_{1,3}(t) = \left(3t(1-t)^2\right)' = 3(1-t)^2 - 6t(1-t) = 3(1-t)(1-3t);$$
$$B'_{2,3}(t) = \left(3t^2(1-t)\right)' = 6t(1-t) - 3t^2 = 3t(2-3t);$$
$$B'_{3,3}(t) = \left(t^3\right)' = 3t^2;$$

Алгоритм 5 Алгоритм де Кастельё построения кривой Безье

dist(P, Q)

Входные данные: P, Q — точки

Результат: Расстояние dist(P, Q) между точками

  dist$(P, Q) \leftarrow \sqrt{(P.x - Q.x)^2 + (P.y - Q.y)^2}$

Bezier$(P_0, P_2, \dots P_n)$

Входные данные: $P_0, P_2, \dots P_n$ — опорные точки кривой Безье

Результат: Изображение кривой на растре

  $t \leftarrow \frac{1}{2}$ {t — параметр алгоритма}

  $d \leftarrow 2$ {В d ищем максимальное из расстояний от P_0 до P_i, $i = 1..n$}

  for $i = 1$ **to** n **do**

    $d \leftarrow \max\{d, \text{dist}(P_0, P_i)\}$

  end for

  if $d < 1$ **then**

    putpixel$(P_0.x,\ P_0.y)$

  else

    for $i = 0$ **to** $n - 1$ **do**

      $P_i^1.x = (1 - t)P_i^0.x + tP_{i+1}^0.x$

      $P_i^1.y = (1 - t)P_i^0.y + tP_{i+1}^0.y$

    end for

    for $k = 2$ **to** n **do**

      for $i = 0$ **to** $n - k$ **do**

        $P_i^k.x = (1 - t)P_i^{k-1}.x + tP_{i+1}^{k-1}.x$

        $P_i^k.y = (1 - t)P_i^{k-1}.y + tP_{i+1}^{k-1}.y$

      end for

    end for

    Bezier$(P_0^1, P_0^2, \dots P_0^n)$

    Bezier$(P_0^n, P_1^{n-1}, \dots, P_n^0)$

  end if

$$B_i^3(0)' = -3A_i + 3C_i = 3(C_i - A_i);$$
$$B_i^3(1)' = -3D_i + 3A_{i+1} = 3(A_{i+1} - D_i);$$
$$y_i(x_i)'|_{A_i} = \frac{B_i^3(0)'.y}{B_i^3(0)'.x} = \frac{C_i.y - A_i.y}{C_i.x - A_i.x};$$
$$y_i(x_i)'|_{A_{i+1}} = \frac{B_i^3(1)'.y}{B_i^3(1)'.x} = \frac{A_{i+1}.y - D_i.y}{A_{i+1}.x - D_i.x}.$$

Таким образом, производные(а значит и касательные к кривым) совпадают, если векторы D_iA_{i+1} и $A_{i+1}C_{i+1}$ коллинеарны. Если они совпадают по величине, то скорости входа кривых в A_{i+1} также совпадают, что даёт визуально более гладкую стыковку, поэтому часто применяется именно это условие.

1.3 Заполнение областей

Рассмотрим многоцветный прямоугольный растр R: $I_1 \times I_2 \to \Omega$, где I_1, I_2 — отрезки, Ω — конечное множество цветов. Будем считать, что граница ∂D области D задана множеством точек цвета b на растре, и даны пиксел (x_0, y_0) внутри D и цвет f. Задача состоит в том, чтобы заполнить изображение D на растре цветом f.

Приведённые далее алгоритмы используют структуру S типа «стек», для которого определены функции:

S.clear сделать стек пустым;

S.empty истинна, если стек пуст;

S.push(a) положить в S элемент a;

S.pop взять элемент с вершины S.

1.3.1 Точечный алгоритм

Идея заключается в том, что на каждом жаге в стеке хранится список точек, подлежащих закрашиванию. После закрашивания очередной точки в стек добавляются все соседние незакрашенные точки, за исключением точек, принадлежащих ∂D. Таким образом, алгоритм заканчивает свою работу при заполнении всех точек внутри D. При практической реализации следует добавить проверку на выход за границы растра. Полученный алгоритм называется «алгоритмом короеда».

Алгоритм 6 Алгоритм короеда

Входные данные: $b \in \Omega$ — цвет, которым на растре R представлена ∂D, (x_0, y_0) — пиксел внутри D, $f \in \Omega$ — цвет заполнения.

Результат: Изображение области D, заполненной цветом f.

```
S — стек
S.clear
S.push((x_0, y_0))
while not S.empty do
    (x, y) ← S.pop
    if R(x, y) ≠ b and R(x, y) ≠ f then
        R(x, y) ← f
        S.push((x + 1, y))
        S.push((x − 1, y))
        S.push((x, y + 1))
        S.push((x, y − 1))
    end if
end while
```

1.3.2 Линейный алгоритм

Принцип действия аналогичен алгоритму короеда, но вместо точек используются горизонтальные отрезки. В стеке хранятся точки отрезков, подлежащих закрашиванию. На каждом шаге добавляются отрезки сверху и снизу, получающиеся при разбиении линии точками границы. Линейный алгоритм использует стек меньшей глубины и эффективнее, т. к. изменение множества точек на одной линии обычно выполняется быстрее из-за особенностей работы памяти.

2 Вычислительная геометрия

2.1 Задача отсечения

В реальных устройствах растр всегда имеет ограниченные размеры, а при отображении различных накладывающихся друг на друга фигур(например, окон на экране) имеет форму, отличную от прямоугольника. Отсюда возникает задача отсечения: вывести на растр только ту часть фигуры, которая лежит внутри его границ. Также при заливке контуров важно, чтобы в границе фигуры не было разрывов, т. е. нужно заменить части фигуры, выходящие за растр отрезками границы растра. Рассмотрим случай, когда фигура, подлежащая выводу, и растр представляют собой многоугольники.

Задача (Пересечение многоугольников)**.** Многоугольники P и Q заданы своими вершинами в порядке обхода: $P = P_1P_2 \dots P_n$, $Q = Q_1Q_2 \dots Q_m$. Требуется найти их пересечение — множество многоугольников $R^1, R^2, \dots R^k$.

Для решения задачи о пересечении многоугольников нужно решить несколько вспомогательных задач.

2.2 Поиск пересечений отрезков

Задача (Пересечение отрезков)**.** Дано множество отрезков $S = \{S_1, S_2, \dots, S_n\}$, отрезки заданы своими концами $S_i = P_iQ_i$. Требуется найти множество $I = \{I_1, I_2, \dots, I_k\}$ точек пересечения отрезков.

2.2.1 Пересечение двух отрезков

Пусть есть отрезки P_1Q_1, P_2Q_2. Зададим прямые, содержащие отрезки уравнениями $a_1x + b_1y = c_1$, $a_2x + b_2y = c_2$ соответственно. Найдём коэффициенты из условия, что прямые проходят через концы отрезка:

$$\begin{cases} a_1(P_1.x) + b_1(P_1.y) = c_1, \\ a_1(Q_1.x) + b_1(Q_1.y) = c_1; \end{cases} \quad \begin{cases} a_2(P_2.x) + b_2(P_2.y) = c_2, \\ a_2(Q_2.x) + b_2(Q_2.y) = c_2. \end{cases}$$

Алгоритм 7 Линейный алгоритм заполнения области

Входные данные: $b \in \Omega$ — цвет, которым на растре представлена ∂D, (x_0, y_0) — пиксел внутри D, $f \in \Omega$ — цвет заполнения.

Результат: Изображение области D, заполненной цветом f.

```
S — стек
S.clear
S.push((x_0, y_0))
while not S.empty do
  (x, y) ← S.pop
  if R(x, y) ≠ f then
    while R(x − 1, y) ≠ b do {Заполнение отрезка}
      x ← x − 1
    end while
    x_1 ← x
    R(x, y) ← f
    while R(x + 1, y) ≠ b do
      x ← x + 1
      R(x, y) ← f
    end while
    x_2 ← x
    for y_1 ∈ {y − 1, y + 1} do {Добавление отрезков сверху и снизу}
      boundary ← true{Предыдущая точка была на границе}
      for x ← x_1 to x_2 do
        if R(x, y) ≠ b then
          if boundary = true then
            S.push((x, y_1))
            boundary ← false
          end if
        else
          boundary ← true
        end if
      end for
    end for
  end if
end while
```

Для каждой прямой получено 2 уравнения на 3 неизвестных. Возьмём любое ненулевое решение, например, с нормировкой $\sqrt{a_i^2 + b_i^2} = 1$. Пусть далее $I = (I.x, I.y)$ — точка пресечения. Найдём её из уравнений:

$$\begin{cases} a_1(I.x) + b_1(I.y) = c_1, \\ a_2(I.x) + b_2(I.y) = c_2. \end{cases}$$

Теперь осталось проверить, что точка I принадлежит обоим отрезкам, т. к. уравнения гарантируют её принадлежность лишь содержащим прямым. Для этого необходимы и достаточны покординатные условия:

$$\begin{gathered} (I.x - P_1.x)(I.x - Q_1.x) < 0, \\ (I.y - P_1.y)(I.y - Q_1.y) < 0, \\ (I.x - P_2.x)(I.x - Q_2.x) < 0, \\ (I.y - P_2.y)(I.y - Q_2.y) < 0, \end{gathered}$$

которые означают, что точка лежит с разных сторон от концов каждого отрезка, т. е. внутри.

2.2.2 Пересечения множества отрезков

Очевидный метод: найти точку пересечения каждого отрезка с каждым, но это требует слишком много операций. Попытаемя ограничить множество отрезков, для которых нужно искать пересечения. Воспользуемя часто встречающимся методом вычислительной геометрии — методом заметающей прямой(Sweep line).

Рассмотрим множество отрезков, пересекающих прямую $y = y_i$, называемое списком активных рёбер L_a. Для эффективного построения L_a введём порядок на множестве точек: $P \prec Q \Leftrightarrow (P.y < Q.y$ или $(P.y = Q.y$ и $P.x < Q.y))$. Вначале отсортируем отрезки соответственно введённому порядку для их начал, при этом предполагаем, что $P_i \prec Q_i$, всегда можно переименовать точки таким способом. После сортировки получаем границы $y \in [y_{min}, y_{max}]$ для всех отрезков. Будем проходить прямой $y = y_i$ от y_{min} до y_{max}, на каждом шаге добавляя в L_a начинающиеся отрезки(P_kQ_k, т.ч. $P_k.y = y_i$) и удаляя заканчивающиеся(P_kQ_k, т.ч. $Q_k.y = y_i$). В начале имеем $L_a = \varnothing$.

На каждом шаге найдём точки пересечения отрезков $P_kQ_k \in L_a$ с заметающей прямой:

$$x_k(i) = \frac{Q_k.x - P_k.x}{Q_k.y - P_k.y} \cdot (y_i - P_k.y)$$

и отсортируем $x_k(i)$ по возрастанию. Сравнив списки $x_k(i-1)$ и $x_k(i)$ на предыдущем и на текущем шаге, вычисляем пересечения: отрезки P_jQ_j и P_lQ_l пересекаются, если $x_j(i-1) < x_l(i-1)$ и $x_j(i) > x_l(i)$. Из этих условий и способа построения L_a видно, что достаточно рассмотреть только те y_i, для которых на прямой $y = y_i$ лежит хотя бы один из концов отрезков.

Для исходной задачи пересечения многоугольников количество отрезков, пересекающихся с заметающей прямой обычно невелико, по сравнению с общим количеством отрезков, поэтому предложенный метод завершается быстрее.

2.3 Проверка принадлежности точки многоугольнику

Задача (Принадлежность точки многоугольнику)**.** Многоугольник Q задан своими вершинами: $Q = Q_1Q_2 \dots Q_n$, задана точка P. Выяснить, лежит ли точка внутри многоугольника.

Рассмотрим пересечения x_i рёбер Q_iQ_{i+1} многоугольника с прямой $y = P.y$. В основе алгоритма следующее наблюдение: любая прямая пересекает границу многоугольника (или любую замкнутую кривую) по чётному числу точек. Таким образом, можно найти количество N точек пересечения x_i, для которых $x_i < P.x$. Если N — нечётно, то точка лежит внутри многоугольника.

Заметим, что при реализации алгоритма возникают сложности, вызванные тем, что действительные числа хранятся в компьютере в виде приближений с точностью ε. Поэтому если какая-нибудь из вершин лежит на прямой $y = P.y$, возможен учёт лишних точек или пропуск пересечений, что приводит к ошибочному результату. Для того, чтобы этого избежать, можно вместо прямой $y = P.y$ рассмотреть прямую $y = P.y - 2\varepsilon$ около тех вершин, которые попали на прямую $y = P.y$.

2.4 Пересечение многоугольников

Пусть даны многоугольники $P = P_1P_2 \dots P_n$, $Q = Q_1Q_2 \dots Q_m$. Найдём пересечения $C_1, C_2, \dots C_k$ рёбер многоугольников, считая, что рёбра одного многоугольника не пересекаются, за исключением концов. Тогда многоугольник $P = P'_1P'_2 \dots P'_{n+k}$, где $\{P'_1, P'_2, \dots P'_{n+k}\} = \{P_1, P_2, \dots, P_n\} \cup \{C_1, C_2, \dots, C_k\}$, P'_i расположены в порядке обхода многоугольника. Аналогично $Q = Q'_1Q'_2 \dots Q'_{n_k}$.

Разделим множество точек пересечения на два класса: точки входа в Q и точки выхода. Это можно сделать следующим способом: выберем какую-либо вершину P, не являющуюся точкой пересечения. Для неё выясним, лежит ли она внутри Q(задача принадлежности точки многоугольнику). Если принадлежит, то следующая точка пересечения будет точкой выхода, иначе - точкой входа. Далее точки входа и выхода будт чередоваться, за один обход многоугольника формируются классы $I = \{I_1, I_2, \dots I_l\}$ точек входа и $O = \{O_1, O_2, \dots, O_l\}$ точек выхода, следующих за соответствующими точками входа.

Будем формировать R_1 — элемент пересечения. Сначала $R_1 = \varnothing$. Возьмём некоторую точку входа $I_1 = P'_{i_1}$. Следующая за ней точка выхода $O_1 = P'_{i_2}$. Ломаная $P'_{i_1}P'_{i_1+1} \dots P'_{i_2}$ лежит внутри Q, поэтому принадлежит пересечению и на следующем шаге $R_1 \leftarrow R_1 + P'_{i_1}P'_{i_1+1} \dots P'_{i_2}$. $O_1 = Q'_{j_1}$, следующая за ней точка входа: Q'_{j_2}. Получаем, что $Q'_{j_1}Q'_{j_1+1} \dots Q'_{j_2}$ лежит внутри P, на следующем шаге $R_1 \leftarrow R_1 + Q'_{j_1}Q'_{j_1+1} \dots Q'_{j_2}$. Далее повторяем процесс прохода то по P, то по Q, пока R_1 не замкнётся. Получим один из элементов пересечения. Удалим точки I_s, принадлежащие R_1 из множества I точек входа. Ели после этого I пусто, то процесс построения пересечения закончен. Иначе - возьмём произвольную точку входа из оставшихся и построим аналогично следующий элемент R_2. В результате построим пересечение $R_1, R_2, \dots, R_N$ многоугольников и задача решена.