Тема 5. Растрові алгоритми генерування кривих ліній

Графічні пристрої, що використовуються в КГ, є в основному растровими, тому при синтезі графічних зображень, що складаються з окремих простих елементів, виникає необхідність у растрових алгоритмах.

Растровий алгоритм – це алгоритм, який для роботи програми, що реалізує графічні примітиви, враховує властивість растра.

Хоча більшість графічних бібліотек містить у собі достатню кількість простіших растрових алгоритмів, часто виникає необхідність явної побудови растрових алгоритмів.

При побудові графічних зображень вдаються до певного набору графічних примітивів, однак найчастіше в графічних системах використовуються лінії.

Зобразити криву на дискретній поверхні (екрані) означає знайти таку її цілочислову апроксимацію, яка дасть можливість відтворити неперервну криву.

Наприклад, для відрізка прямої потрібно отримати послідовність дискретних точок (пікселів), що апроксимує неперервну лінію (рис. 5.1).

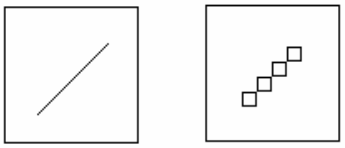

Рис. 5.1 Наближення пікселями неперервної лінії

Методи побудови ліній на дискретній поверхні поділяються на 2 класи:

- числові методи (методи прямих обчислень координат);
- інкрементні методи.

5.1. Числові методи

Числові методи базуються на числовому аналізі рівнянь кривих, тобто на обчисленні значень функцій.

Розглянемо пряму лінію загального вигляду. Для її зображення теж необхідно обчислювати координати кожного пікселя. Відомо декілька методів розрахунку координат кожного пікселя лінії.

Нехай задано координати кінців відрізка $A(x_1, y_1)$, $B(x_2, y_2)$, тоді рівняння прямої, що проходить через дві точки A і B, має вигляд $y = ax+b$, де

$$a = \frac{y_2 - y_1}{x_2 - x_1}, \qquad b = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}.$$

Для простоти вважатимемо, що $0 \le y_2 - y_1 < x_2 - x_1$, тобто цикл у програмі буде здійснюватись по x.

За цих припущень наведемо приклад запису простішого алгоритму побудови прямої лінії на мові Pascal.

```
procedure Line_1 (x1, y1, x2, y2, c : integer);
var a, b : real, x : integer;
begin
        a:= (y2–y1) / (x2–x1);
        b:= y1– a*x1;
        for x:= x1 to x2 do
        putpixel (x, int (a*x+b), c);
end; {Line_1}
```

Недоліком такої програми є те, що в циклі виконується множення та використовуються операції з плаваючою крапкою. А це зменшує швидкість виведення графічного зображення.

Обчислення значень функції $y = ax + b$ можна уникнути, якщо використати рекурентне співвідношення $y = y + a$, оскільки при зміні x на одиницю значення функції y змінюється на a. Ця модифікація алгоритму має такий вигляд:

```
procedure Line_2 (x1, y1, x2, y2, c : integer);
var a : real, x : integer;
  begin
        a:= (y2 – y1) / (x2 – x1);
        y:= y1;
        for x:= x1 to x2 do
        begin
              putpixel (x, int y, c);
              y:=y + a;
        end;
end; {Line_2}
```

Оскільки виділення цілої частини значення y не завжди може привести до коректного графічного результату, то покращити зовнішній вигляд відрізка можна за рахунок округлення значень функції до ближчого цілого, тобто це означає, що з двох пікселів, які лежать на одній вертикалі (так, що відрізок прямої проходить між ними), завжди вибирається той піксель, який лежить ближче до прямої. Для цього досить порівняти дробову частину значення y з 0,5.

Зауваження 1. Вибір одного з двох пікселів можна здійснити шляхом визначення середини між двома пікселями-кандидатами і перевірки,

місцезнаходження (вище чи нижче середини) точки перетину відрізка прямої з даною вертикаллю. Це метод серединної точки.

У цьому алгоритмі для обчислення координати y в тілі циклу є тільки одна операція '+', тому порівнюючи наведені алгоритми, останній кращий за швидкодією.

Але незважаючи на те, що вхідні дані є цілочисловими величинами і всі результати теж мають бути цілочисловими, ця модифікація алгоритму все одно використовує дійсні числа. Окрім цього, оскільки для обчислення y в останньому варіанті використовується операція $y := y + a$, то з кожним кроком циклу будуть накопичуватись похибки за рахунок неточного представлення дробових чисел у комп'ютері та за рахунок похибки арифметичних операцій із плаваючою крапкою. Тому в результаті виконання циклу може статися так, що на останньому кроці $y \neq y_2$ (хоча з математичної точки зору тут все коректно, при $x = x_2$ маємо $y = y_2$). Це теж необхідно враховувати при використанні даного алгоритму.

Отже, числові методи мають перевагу завдяки простоті та ясності побудови алгоритму і можливості працювати з дробовими значеннями координат точок відрізка.

Однак у числових методів є й свої недоліки:

1) використання операцій із плаваючою крапкою та операцій множення або ділення в циклі зумовлює малу швидкість обчислень, хоча це суттєво залежить від процесора (у сучасних комп'ютерах, де процесори використовують ефективні засоби прискорення, наприклад, конвеєр арифметичних операцій із плаваючою крапкою, час виконання цілочислових операцій не набагато менший);
2) при обчисленні координат шляхом додавання приросту може накопичуватись похибка обчислення координат.

5.2. Інкрементні алгоритми

У 1965 році Брезенхем (Bresenham J.E.) запропонував підхід для створення інкрементних методів (тоді ці методи використовувались для графопобудівників).

Основною ідеєю інкрементних методів є теж рекурентні співвідношення для обчислення y, але в них уже не застосовуються дійсні обчислення й операції множення та ділення, а застосовуються лише цілочислові операції додавання і віднімання.

Інкрементні алгоритми виконуються як послідовність обчислень координат сусідніх точок шляхом додавання відповідних приростів так, щоб сусідня цілочислова точка була найближчою до неперервної кривої. Прирости розраховуються на основі аналізу функції похибок. У циклі

виконуються лише цілочислові операції порівняння, додавання та віднімання. У такий спосіб досягається більш висока швидкодія обчислень координат кожного пікселя порівняно з числовими методами.

На сьогодні існує багато спеціалізованих інкрементних алгоритмів для генерування кривих того чи іншого типу (наприклад відрізків, кіл, еліпсів тощо). В основному вони реалізовані апаратно.

5.2.1. Алгоритм Брезенхема для відрізка

Серед графічних примітивів найбільшу питому вагу мають відрізки прямих. У зв'язку з цим при розробці графічних засобів алгоритмам лінійної інтерполяції приділяють особливу увагу.

Розглянемо спочатку простіший випадок формування відрізка прямої, коли пряма проходить через початок координат (0,0) і точку з координатами (n, m). Рівняння цієї прямої має вигляд

$$mx - ny = 0.$$

При цьому виникає задача за відомою точкою (x, y), що найкраще наближає пряму, визначити наступну точку (x+1, y) чи (x, y+1). Введемо в розгляд функцію $F(x, y) = mx - ny$. Нехай значення $F(x, y) = f \neq 0$. Знайдемо

$F(x + 1, y) = m(x + 1) - ny = mx - ny + m = f + m$,

$F(x, y + 1) = mx - n(y + 1) = mx - ny - n = f - n$.

Щоб зробити вибір між двома точками, потрібно знати, яке з двох чисел $f + m$ чи $f - n$ знаходиться ближче до нуля.

Якщо значення $f + m$ ближче до нуля, то рух із точки (x, y) відбудеться в точку (x +1, y), а якщо $f - n$ ближче до нуля, то в точку (x, y +1).

Для знаходження приросту координат (x, y) побудуємо функцію похибок $S(x, y)=F(x + 1, y) + F(x, y + 1) = 2f + m - n$ і виробимо ознаку вибору наступної точки. Для цього потрібно розглянути такі випадки:

1) якщо $f + m < 0$, $f - n < 0$, тобто $f + m$ є ближчим до нуля (рис. 5.2, *а*), а $S = 2f + m - n < 0$, то наступною є точка ($x + 1$, y);
2) якщо $f + m > 0$, $f - n > 0$, тобто $f - n$ є ближчим до нуля (рис. 5.2, *б*), а $S = 2f + m - n > 0$, то рух відбувається в точку (x, $y + 1$);
3) якщо $f + m > 0$, $f - n < 0$, тобто $f + m$ є ближчим до нуля (рис. 5.2, *в*), а $S = 2f + m - n < 0$, то надалі вибираємо точку ($x + 1$, y);
4) якщо $f + m > 0$, $f - n < 0$, тобто $f - n$ є ближчим до нуля (рис. 5.2, *г*), а $S = 2f + m - n > 0$, то рух відбувається в точку (x, $y + 1$);

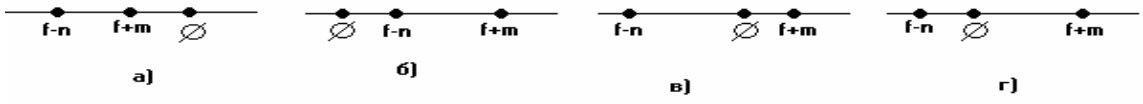

Рис 5.2. Розміщення точок

Отже, вибір наступної точки визначається знаком функції похибок S: якщо $S > 0$, то наступною є точка $(x, y+1)$, а якщо $S \le 0$, то $(x+1, y)$. При переході до наступної точки змінюється значення S, тому для простого обчислення S потрібно мати рекурентну формулу.

Оскільки $S(x, y) = 2f(x, y) + m - n$, то початкове значення $S_0 = S(0, 0) = m - n$. Розглянемо

$$S(x+1, y) = 2f(x+1, y) + m - n = 2(mx - ny + m) + m - n = 2f + m - n + 2m = S(x, y) + 2m;$$
$$S(x, y+1) = 2f(x, y+1) + m - n = 2(mx - ny - n) + m - n = 2f + m - n - 2n = S(x, y) - 2n;$$

Таким чином, маємо такі рекурентні формули для S:

$S = S + 2m$, якщо $S \le 0$;

$S = S - 2n$, якщо $S > 0$.

Один із варіантів алгоритму Брезенхема в простішому випадку має вигляд

```
x:= 0; y:= 0;  putpixel (0, 0, c);
m2:=m+m; n2:=n+n; S:=m–n;
for i:=1 to m+n do begin
if S= < 0 then begin S:=S+m2 ; x:=x+1; end
  else begin S:=S–n2 ; y:=y+1; end;
putpixel (x,y,c);
end {for}.
```

У цьому алгоритмі перехід із точки (x, y) до наступної точки може бути виконаний двома способами. Це може викликати подвійне зображення прямої лінії, тому потрібно передбачити приріст по x і по y одночасно, тобто як сусідню необхідно розглядати ще і точку $(x + 1, y + 1)$. Щоб зобразити весь відрізок між точками (0, 0) та (m, n), потрібно зробити $l = \max\{m, n\}$ кроків. Отже, приходимо до загального алгоритму Брезенхема для довільного відрізка з координатами кінців (x_1, y_1) і (x_2, y_2).

```
procedure Line_3 (x1, y1, x2, y2, c:integer);
{Bresenham algorithm}
var k, l, dy, S, m, n : integer;
begin
if (x1 > x2) then begin S:=x1; x1:=x2; x2:=S;
S:=y1; y1:=y2; y2:=S; end;
if y1 > y2 then dy:= –1 else dy:=1;
m:=abs(y2–y1); n:=x2–x1;
S:=m–n;
if m>n then l:=m else l:=n;
m:=m+m; n:=n+n;
putpixel (x1,y1,c);
```

```
for k:=1 to l do
  begin
        if S=<0 then begin S:=S+m: x1:=x1+1; end;
         if S>0 then begin S:=S–n; y1=y1+dy; end;
         putpixel (x1,y1,c);
   end {for};
end; {Line_3}
```

Зауваження 2. Для побудови широкої лінії можна її нарисувати декілька разів із певним зміщенням початкових координат або модифікувати алгоритм Брезенхема так, щоб замість оператора виведення пікселя putpixel (x,y,c) виступав оператор виведення фігури так званого логічного пера з центром у точці (x, y). Як фігуру пера можна обрати прямокутник, круг, відрізки прямої тощо. Такий підхід до побудови ліній має переваги і недоліки. Недоліком є неефективність для деяких форм пера; наприклад, якщо перо має форму квадрата, то більшість пікселів при цьому неоднократно перефарбовується (рис. 5.3).

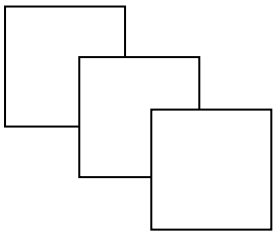

Рис. 5.3. Побудова широкої лінії

Такі алгоритми більш ефективні, коли перо має форму відрізка – найчастіше вертикального або горизонтального, але горизонтальне перо не може намалювати широку горизонтальну лінію.

Зауваження 3. Лінію можна зображати не завжди суцільною (лінії можуть мати різні структури та типи). Для цього в алгоритм рисування лінії передають ще один аргумент (це ціле число без знака), двійковий код якого багаторазово копіюється вздовж лінії. Кожний біт при цьому циклічно аналізується. Якщо значення біта дорівнює одиниці, то піксель зображаємо, якщо біт дорівнює нулю, то піксель не зображаємо.

Зауваження 4. Для зменшення ступінчастості растрового зображення лінії використовують ефект згладжування. На дисплеях з малою роздільною здатністю екрана 320 × 200 згладжування не дає ефекту. Для досягнення ефекту згладжування необхідні режими з вищою роздільною здатністю, наприклад, 640 × 480. Згладжування при цьому виконується за рахунок зміни яскравості пікселів, тому можливість згладжування залежить від шкали яскравості. Наприклад, для ребер многокутника можна встановити яскравість пікселів пропорційно

площі частини пікселя, що попадає в середину многокутника. Згладжування відрізків ще можна проводити шляхом розмивання різкої границі за рахунок підсвічування пікселів, які прилягають до відрізка. На дисплеях із роздільною здатністю 1280 × 1024 згладжування відбувається за рахунок малого зерна дисплея.

5.2.2. Алгоритм Брезенхема для кола

Для побудови кола $x^2 + y^2 = R^2$ можна реалізувати алгоритм шляхом прямого обчислення координат. Або, маючи програму побудови відрізка, коло можна апроксимувати хордами, але коло буде недостатньо гладким, поки кількість хорд не стане більшою 36. При високих роздільних здатностях потрібна ще більша кількість хорд. Ці методи вимагають багато обчислень, що зменшує швидкість побудови кола. Тому розглянемо інкрементний метод Брезенхема для кола.

Для спрощення алгоритму растрової розгортки кола можна використати його симетрію відносно координатних осей і прямих $y = \pm x$. Тому достатньо побудувати растрове зображення для 1/8 частини кола, а решту частин побудувати за рахунок симетрії кола (якщо точка (x,y) належить колу, то і точки $(-x, y)$, $(x, -y)$, $(-x, -y)$, (y, x), $(-y, x)$, $(y, -x)$, $(-y, -x)$ належать колу).

Із цією метою для виведення кола з центром у точці (xc, yc) побудуємо таку процедуру:

```
procedure circle_8 (x, y, xc, yc, c: integer);
begin
putpixel (xc+x, yc+y, c);
putpixel (xc+y, yc+x, c);
putpixel (xc–x, yc+y, c);
putpixel (xc–y, yx+x, c);
putpixel (xc+x, yc–y, c);
putpixel (xc+y, yc–x, c);
putpixel (xc–x, yc–y, c);
Putpixel (xc–y, yc–x, c);
end; {circle_8}
```

Тепер розглянемо ділянку кола з другого октанта, тобто при $x \in [0, R/\sqrt{2}]$, $y \in [R/\sqrt{2}, R]$. Далі для кожної точки $P(x, y)$ вводимо змінну $D(P)$ за правилом: якщо точка P є зовнішньою до кола, то $D(P) = x^2 + y^2 - R^2$, а якщо внутрішньою, то $D(P) = R^2 - (x^2 + y^2)$. Щоб на i-му кроці визначити, яка з двох точок зовнішня S_i чи внутрішня T_i найкраще апроксимуватиме коло, потрібно порівняти величини $D(S_i)$ та $D(T_i)$.

Нехай $d_i \equiv D(S_i) - D(T_i)$. Якщо $d_i < 0$, то на i-му кроці вибирається точка S_i, оскільки $D(S_i) < D(T_i)$, якщо $d_i \geq 0$, то вибирається точка T_i.

Розглянемо перший крок алгоритму. З рівняння кола очевидно, що точка (0, R) лежить на колі. На першому кроці потрібно вибрати точку серед $S_1(1, R)$ чи $T_1(1, R–1)$ (рис. 5.4), яка найкраще наближає коло. Вибір здійснюється тільки серед двох точок тому, що кутовий коефіцієнт дотичної до кола в цьому октанті належить проміжку [–1; 0].

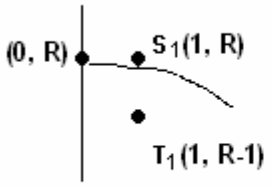

Рис. 5.4

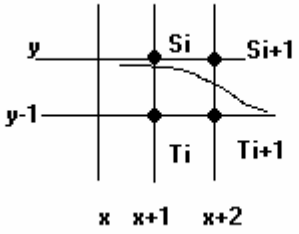

Рис. 5.5

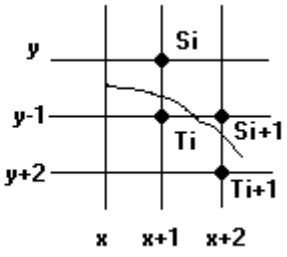

Рис. 5.6

Знайдемо $d_1 = D(S_1) – D(T_1) = 1 + R^2 – R^2 – (R^2 – (1 + (R – 1)^2)) = 3 – 2R$. Якщо $d_1 = 3 – 2R > 0$, то на першому кроці вибираємо внутрішню точку $T_1(1, R – 1)$, а якщо $d_1 = 3 – 2R < 0$, то вибираємо зовнішню точку $S_1(1, R)$.

Вибір точки на i + 1-му кроці залежить від того, яка точка була вибрана на i-му кроці – S_i чи T_i. (рис. 5.5).

Нехай на i-му кроці була вибрана точка S_i, тобто $d_i = D(S_i) – D(T_i) < 0$, $d_i = (x + 1)^2 + y^2 – R^2 – (R^2 – (x + 1)^2 – (y – 1)^2)$. Знайдемо $d_{i+1} = D(S_{i+1}) – D(T_{i+1}) = (x +2)^2 + y^2 – R^2 – (R^2 – (x +2)^2 – (y – 1)^2) = d_i + 4x + 6$.

Нехай $d_i > 0$, тобто на i-тому кроці краще апроксимує коло точка T_i (рис. 5.6), тоді $d_{i+1} = D(S_{i+1}) – D(T_{i+1}) = (x +2)^2 + (y – 1)^2 – R^2 – (R^2 – (x +2)^2 – (y –2)^2) = d_i + 4(x – y) + 10$.

Запишемо алгоритм Брезенхема для кола.

```
x:= 0;  y:= R; d:=3–2*R;
while (x <=y) do  begin
circle_8 (x, y, xc, yc, c);
if d < 0 then d:= d+4*x+6
else begin d:= d+4*(x–y)+10; y:=y–1; end;
x:= x+1;
end; {while}
```

Зауваження 5. Щоб одержати широке коло, необхідно нарисувати декілька кіл усередині даного кола, послідовно зменшуючи радіус на одиницю (або зовні даного кола).

Зауваження 6. Кола можуть мати різну структуру. Для побудови таких кіл циклічно аналізуються біти із заданого байта або слова, як і при побудові прямої лінії.

5.2.3. Інкрементний алгоритм виведення еліпса

Цей алгоритм більш складний, ніж алгоритм побудови кола. Внаслідок симетрії еліпса відносно осей координат координати точок еліпса обчислюватимуться лише в першій чверті. Канонічне рівняння еліпса має вигляд $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$, де a та b – велика і мала півосі еліпса.

Зведемо рівняння еліпса до такого вигляду:

$$\frac{x^2}{a^2}+\frac{y^2}{(b+0.5)^2}=1$$

Звідси маємо

$$\frac{x^2}{a^2}+\frac{(2y)^2}{(2b+1)^2}=1, \quad \text{або} \quad (2b+1)^2x^2+4a^2y^2=(2b+1)^2a^2.$$

Позначимо $(2b+1)^2=m$, $a^2=n$, $F(x,y)=mx^2+4ny^2-mn$, тоді рівняння еліпса матиме вигляд $F(x, y)=0$.

Ідея алгоритму полягає ось у чому: для кожного y, що змінюється від b до 0, необхідно підібрати ціле значення x так, щоб $F(x, y)$ було найближчим до нуля (точка (x, y) найкраще апроксимує еліпс).

Якщо $F(x, y) < 0$, то збільшуємо x ($x:= x+1$), якщо $F(x, y) > 0$, то зменшуємо y ($y:= y-1$). Але знак функції $F(x, y)$ не може бути ознакою вибору точки (x, y).

Для вибору точки (x, y), що найкраще апроксимує еліпс, утворюємо функцію похибок $S(x, y)$.

$$S(x, y)=F(x, y)+F(x+1, y)=2m(x^2+x)+8ny^2-2mn+m.$$

Тоді, якщо $S(x, y) < 0$, то збільшуємо $x:= x+1$, (рис. 5.7, *а*); якщо $S(x, y)$ стає більшим нуля (рис. 5.7, *б*), то ми підібрали потрібну точку x (цей піксель зображаємо на екрані) і здійснюємо перехід до наступного y ($y:= y-1$). Отже, точка x – це перша точка, для якої $S(x, y) > 0$.

а *б*

Рис. 5.7. Вибір пікселя

Для початкової точки $(0, b)$ $S_0=8nb^2-2mn+m$. Для наступних точок (x, y) потрібно мати значення $S(x, y)$, але його не можна знайти безпосередньо, тому що в цей вираз входять операції множення (втра-

чається швидкодія алгоритму). Для обчислення значень S(x, y) побудуємо рекурентні співвідношення без операцій множення та ділення.

Нехай $S(x+1, y) = S(x, y) + P(x, y)$, тоді $P(x, y) = (S(x+1, y) - S(x, y)) = 2m((x+1)^2 + (x+1)) + 8ny^2 - 2mn + m - 2m(x^2 + x) - 8ny^2 + 2mn - m = 4mx + 4m$.

З останнього співвідношення видно, що при збільшенні x на одиницю, P збільшується на $4m$, тому для P маємо рекурентне співвідношення $P(x+1, y) = P(x, y) + dp$, де $dp = 4m$, $P(0, y) = 4m$.

Аналогічно при зміні y на $y - 1$ маємо

$Q(x, y) = S(x, y-1) - S(x, y) = 2m(x^2 + x) + 8n(y-1)^2 - 2mn + m - 2m \times (x^2 + x) - 8ny^2 + 2mn - m = -16ny + 8n$, тобто $Q(x, y-1) = Q(x, y) + dq$, $dq = 16n$, $Q(0, b) = -16nb + 8n$.

Отже, одержуємо таку процедуру побудови еліпса.

```
procedure Ellips (x0, y0, Rx, Ry, c: integer);
var S, P, Q, m, n, dp, dq: longint, a, b, x, y: integer;
begin
b:= Ry;  a:=Rx;  n:=a*a;  m:=sqr(2*b+1);
x:= 0;  S:=8*n*b*b–2*m*n+m;
P:=4*m;  dp:= P;  Q:= –16*n*b+8*n; dq:=16*n;
putpixel (x0, y0 + Ry, c);  putpixel (x0, y0 – Ry, c);
for y:=Ry downto 0  do
begin
   if S < 0 then while S < 0 do
         begin
         x:= x+1; S:= S+P;  P:= P+dp;
         P4(x0, y0, x, y, c);
         end;
  S:=S+Q; Q:=Q+dq; P4(x0, y0, x, y, c);
end {for};
end; {Ellips}
```

Наведемо процедуру побудови симетричних точок відносно осей координат.

```
procedure P4(x0, y0, x, y, c: integer);
begin
putpixel (x0 + x, y0 + y, c);
putpixel (x0 + x, y0 – y, c);
putpixel (x0 – x, y0 + y, c);
putpixel (x0 – x, y0 – y, c);
end; {P4}
```

Зауваження 7. Якщо в основній процедурі замість оператора виклику процедури P4 вставити оператори

Hor ($x0 - x$, $x0 + x$, $y0 + y$, c);

Hor ($x0 - x$, $x0 + y$, $y0 - y$, c);

то одержуємо процедуру зафарбовування еліпса.

5.2.4. Інкрементний метод Жордана

Нехай крива задана неявним рівнянням $f(x, y) = 0$ і похідні $f_x, f_y, f_{xx}, f_{xy}, f_{yy}$ та ін. – неперервні. Припустимо, що деяка цілочислова точка $P(x,y)$ найкраще апроксимує криву $f(x, y) = 0$.

Метод Жордана полягає в тому, щоб, рухаючись уздовж кривої, з максимально можливою точністю підбирати наступні цілочислові точки. Із точки $P(x, y)$ можна перейти до сусідніх точок вісьмома способами, тобто сусідні точки можуть мати координати $(x \pm 1, y)$, $(x, y \pm 1)$, $(x \pm 1, y \pm 1)$ (рис. 5.9).

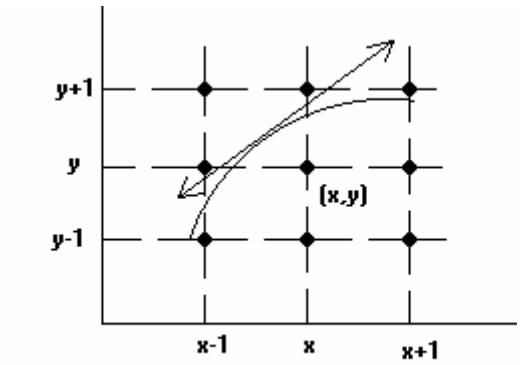

Рис. 5.9. Сусідні точки

Загальна формула сусідніх точок $(x + \Delta x, y)$, $(x, y + \Delta y)$, $(x + \Delta x, y + \Delta y)$, де $\Delta x, \Delta y = \pm 1$.

Знаючи знаки Δx, Δy, вибір серед восьми сусідніх точок зводиться до вибору серед трьох точок. Рух по кривій у точці $P(x, y)$ замінюємо рухом по дотичній. Можливі два напрямки руху вздовж дотичної $V^+ = (-f_y, f_x)$, $V^- = (f_y, -f_x)$.

Якщо вибрати напрям V^+, то знак Δx протилежний знаку f_y, а знак Δy збігається зі знаком f_x.

Якщо вибрати напрям V^-, то знак Δx збігається зі знаком f_y, а знак Δy протилежний знаку f_x.

Покладемо

$$d = \begin{cases} 1, \text{ якщо вибрано напрям } V^+, \\ 0, \text{ якщо вибрано напрям } V^-. \end{cases}$$

Тоді здійснюємо вибір:

якщо ($f_y < 0$ і d=1) або ($f_y \geq 0$ і $d = 0$), то $\Delta x = 1$;

якщо ($f_y \leq 0$ і d=1) або ($f_y < 0$ і $d = 0$), то $\Delta x = -1$; (5.3)

якщо ($f_x \geq 0$ і d=1) або ($f_x < 0$ і $d = 0$), то $\Delta y = 1$;

якщо ($f_x < 0$ і d=1) або ($f_x \geq 0$ і $d = 0$), то $\Delta y = -1$.

Тепер серед трьох точок вибираємо найближчу точку до кривої $f(x, y) = 0$, тобто точку для якої $|f(x, y)|$ мінімальне.

Позначимо

$f(x+\Delta x, y) \equiv f^x$, $f(x, y+\Delta y) \equiv f^y$, $f(x+\Delta x, y+\Delta y) \equiv f^{xy}$,

тоді

якщо $|f^x| < |f^y|$ і $|f^x| < |f^{xy}|$, то $(y := y, x := x+\Delta x)$;

якщо $|f^y| < |f^x|$ і $|f^y| < |f^{xy}|$, то $(x := x, y := y+\Delta y)$; (5.4)

якщо $|f^{xy}| < |f^x|$ і $|f^{xy}| < |f^y|$, то $(x := x+\Delta x, y := y+\Delta y)$.

Для знаходження значень функції використаємо розклад у ряд Тейлора

$$f^x = f(x+\Delta x, y) = f(x,y) + f_x\,\Delta x + \frac{1}{2} f_{x^2}\Delta x^2 + \frac{1}{6} f_{x^3}(\Delta x)^3 + \ldots..$$

$$f^y = f(x, \Delta y + y) = f(x, y) + f_y\,\Delta y + \frac{1}{2} f_{y^2}\Delta y^2 + \frac{1}{6} f_{y^3}(\Delta y)^3 + \ldots.. \quad (5.5)$$

$$f^{xy} = f(x+\Delta x, y+\Delta y) = f(x, y) + f_x\,\Delta x + f_y\Delta y + \frac{1}{2}(f_{x^2}\Delta x^2 + 2 f_{xy}\Delta x\Delta y + f_{y^2}\Delta y^2) + \ldots$$

Частинні похідні в наступних точках у свою чергу визначаємо співвідношеннями

$$f_x(x+\Delta x, y) = f_x + f_{x^2}\Delta x + \frac{1}{2} f_{x^3}\Delta x^2 + \ldots,$$

$$f_y(x+\Delta x, y) = f_y + f_{xy}\Delta x + \frac{1}{2} f_{x^2 y}\Delta x^2 + \ldots,$$

$$f_{x^2}(x+\Delta x, y) = f_{x^2} + f_{x^3}\Delta x + \frac{1}{2} f_{x^4}\Delta x^2 + \ldots, \quad (5.6)$$

$$f_{y^2}(x+\Delta x, y) = f_{y^2} + f_{xy^2}\Delta x + \frac{1}{2} f_{x^2 y^2}\Delta x^2 + \ldots,$$

$$f_{xy}(x+\Delta x, y) = f_{xy} + f_{x^2 y}\Delta x + \frac{1}{2} f_{x^3}\Delta x^2 + \ldots .$$

Оскільки Δx, Δy можуть набувати значень ±1, 0, то обчислення значень функції та похідних у сусідніх точках виконуються тільки за допомогою операцій додавання та віднімання.

У випадку, коли $f(x, y)$ є алгебраїчною кривою у формулах (5.5), (5.6) наявна скінченна кількість доданків, тоді алгоритм Жордана буде точним, інакше – наближеним.

Отже, алгоритм Жордана має такий вигляд:

підібрати першу точку $P(x, y)$ і відобразити її на екрані;

початкова ініціалізація: знайти $f(x, y)$, частинні похідні $f_x(x, y)$, $f_y(x, y)$ і т.д., визначити d;

поки точка (x, y) не є кінцевою *виконувати*

початок

обчислити Δx, Δy для наступної точки за формулами (5.3);

обчислити $|f^x|$, $|f^y|$, $|f^{xy}|$ за формулами (5.5);

за формулами (5.4) вибрати наступну точку і зобразити її на екрані;

для наступної точки обчислити частинні похідні за формулами (5.6);

кінець.

Остання точка на кривій не обов'язково обчислюється за допомогою цього алгоритму. Якщо відомі координати (xk, yk) останньої точки на кривій, то критерієм зупинки роботи цього алгоритму може бути умова $|x - xk| \le 1$ і $|y - yk| \le 1$, або $y = yk$ і $|x - xk| \le 1$, або $x = xk$ і $|y - yk| \le 1$.

Зауваження 8. Для відрізків прямих та кривих другого порядку більш ефективними є спеціалізовані растрові алгоритми, що були розглянуті вище, оскільки метод Жордана вимагає обчислення значення функції та її похідних і не враховує особливості кожної з кривих.

Контрольні питання та завдання

1. Які алгоритми називаються растровими?
2. Які методи використовуються для побудови растрових алгоритмів?
3. В чому полягають переваги та недоліки числових методів?
4. Розкрийте суть ідеї інкрементних методів.
5. Записати простіший алгоритм Брезенхема для відрізка.
6. Який вигляд має функція похибок в алгоритмі Брезенхема для відрізка?
7. Записати алгоритм Брезенхема для побудови кола.
8. Яка частина кола формується основним алгоритмом? Як формується решта частин кола?
9. Який вигляд має функція похибок в алгоритмі Брезенхема для кола?
10. Яка основна ідея інкрементного алгоритму побудови еліпса.
11. Записати інкрементний алгоритм Жордана.
12. В чому полягають переваги та недоліки методу Жордана?
13. Для яких кривих метод Жордана є точним?

Вправи і задачі для самостійного виконання

1. Реалізувати числовий метод побудови кола та еліпса.
2. Написати програму побудови дуги еліпса.
3. Розробити ефективний алгоритм для обчислення координат довільного прямокутника, якщо задано координати кінців однієї з його сторін та ширину прямокутника.
4. Розробити алгоритм побудови широкої лінії, яка задається двома точками і шириною лінії, використовуючи процедуру Line_3.
5. Ребро многокутника задано своїми вершинами. Розробити ефективний алгоритм визначення точок перетину рядків растра з цим ребром.
6. Розробити інкрементний алгоритм побудови гіперболи.
7. Розробити інкрементний алгоритм побудови параболи.
8. Розробити інкрементний алгоритм побудови лінії $y = ax^3$, $a > 0$.
9. Записати алгоритм Жордана для алгебраїчних кривих другого та третього порядків.