4.1. Кривые второго порядка

Общий вид уравнения линии второго порядка:

$$a_1x^2+a_2y^2+a_3xy+a_4x+a_5y+a_6=0.$$

Рассмотрим различные формы линий второго порядка.

Уравнение окружности:

$(x-x_0)^2+(y-y_0)^2=r^2$ – *нормальное уравнение окружности,*

$C(x_0, y_0)$ – центр окружности,

$M(x,y)$ – произвольная точка окружности,

$$x^2+y^2+ax+by+g=0,$$
$$a=-2x_0\ ;\ b=-2y_0;\quad g=x_0^2+y_0^2-r^2.$$

$Ax^2+Ay^2+Dx+Ey+F=0$ – *общее уравнение окружности.*

Действительная окружность - при $(a^2+b^2)/4-g>0$, где $a=D/A$; $b=E/A$; $g=F/A$.

Таким образом, действительная кривая второго порядка является *окружностью* тогда и только тогда, когда:

1) коэффициенты при квадратах текущих координат равны между собой;
2) отсутствует член, содержащий произведение текущих координат.

Центральные кривые второго порядка – кривые, имеющие собственный центр симметрии.

$A(x-x_0)^2+C(y-y_0)^2=D, \quad A \neq C.$

$O(x_0, y_0)$ – центр симметрии кривой;

$x=x_0, \ y=y_0$ – оси симметрии кривой.

Пусть $x_0=0$ и $y_0=0$.

Кривая 2-го порядка принадлежит *эллиптическому типу*, если коэффициенты A и C имеют одинаковые знаки, т.е. $AC>0$.

Пусть $A>0$ и $C>0$, тогда если:

- $D>0$ – действительный эллипс, $x^2/a^2+y^2/b^2=1$ – каноническое уравнение эллипса;
- $D=0$ – вырожденный эллипс (кривая вырождается в точку);
- $D<0$ – мнимый эллипс.

Кривая 2-го порядка является кривой *гиперболического типа*, если коэффициенты A и C имеют противоположные знаки, т.е. $AC<0$.

Пусть $A>0$, тогда $C<0$:

- если $D>0$ – гипербола (рис. 4.1), $x^2/a^2-y^2/b^2=1$ – *каноническое уравнение гиперболы*;
- если $D=0$ – вырожденная гипербола представляет собой пару пересекающихся прямых

$$\left(\sqrt{A}x-\sqrt{-C}y\right)\left(\sqrt{A}x+\sqrt{-C}y\right)=0\,;$$

● если $D<0$ – сопряженная гипербола (рис. 4.1), $xy=a^2$ ($a>0$) – *обратная пропорциональность*. Графиком является равнобочная гипербола (к уравнению гиперболы легко прийти, повернув оси координат на 45 º – рис. 4.2).

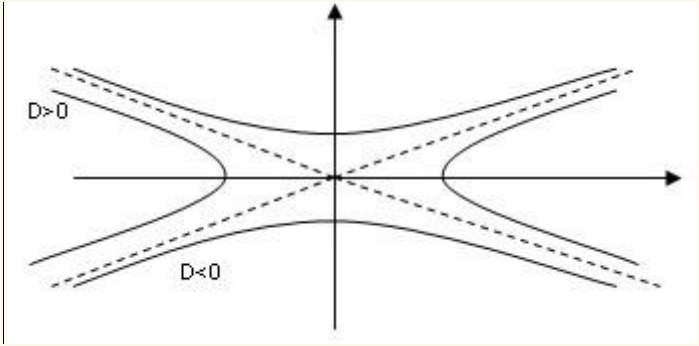


Рис. 4.1. Гипербола и сопряженная гипербола

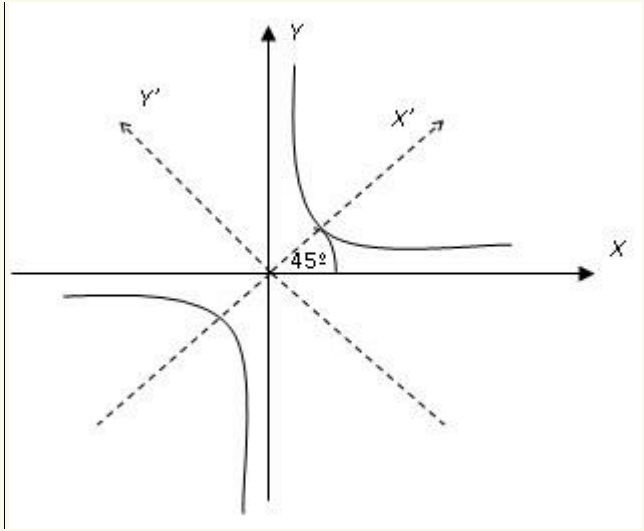


Рис. 4.2. Обратная пропорциональность

Нецентральные кривые второго порядка не имеют центра симметрии или имеют бесконечное множество центров, могут иметь ось симметрии.

$(y-y_0)^2=2p(x-x_0)$ – *парабола*, $O(x_0, y_0)$ – вершина параболы, P – параметр параболы.

Если вершина параболы находится в начале координат и $p>0$, то $y^2=2px$ – *каноническое уравнение параболы*.

Параметрические уравнения линий.

Параметрическое уравнение окружности:

$$\begin{cases} x = r\cos t \\ y = r\sin t \end{cases}.$$

Параметрическое уравнение эллипса:

$$\begin{cases} x = a\cos t \\ y = b\sin t \end{cases}.$$

Параметрическое уравнение параболы:

$$\begin{cases} x = t^2 \\ y = \sqrt{2p}\cdot t \end{cases}.$$

Циклоида (кривая, описываемая точкой окружности, катящейся без скольжения по прямой линии):

$$\begin{cases} x = a\,(\mathrm{t} - \sin t) \\ y = a\,(\mathrm{t} - \cos t) \end{cases}.$$

Спираль Архимеда: r=aj (a>0) – уравнение в полярных координатах.

4.2. Приведение уравнений кривых второго порядка к каноническому виду

Рассмотрим, как можно с помощью преобразований координат привести общее уравнение кривой второго порядка

$$a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_1x + 2a_2y + a_0 = 0$$

к каноническим уравнениям эллипса, гиперболы или параболы, или к случаям их вырождения.

С помощью поворота осей координат на некоторый угол α всегда можно избавиться от члена с произведением координат.

Действительно,

$$a'_{11}x'^2 + 2a'_{12}x'y' + a'_{22}y'^2 + 2a'_1x' + 2a'_2y' + a'_0 = 0$$

коэффициент которого a'_{12} будет равен

$$a'_{12} = (a_{22} - a_{11})\sin\alpha\cos\alpha + a_{12}(\cos^2\alpha - \sin^2\alpha)$$

Приравнивая коэффициент $\boldsymbol{a'}_{12}$ к нулю, получим тригонометрическое уравнение

$$(a_{22} - a_{11})\sin\alpha\cos\alpha + a_{12}(\cos^2\alpha - \sin^2\alpha) = 0\,.$$

Отсюда получаем

$$\operatorname{tg} 2\alpha = \frac{2a_{12}}{a_{11} - a_{22}}$$

Далее, по формулам тригонометрии, получаем нужные нам значения для sin α и cos α:

$$\operatorname{tg} 2\alpha = \frac{2\operatorname{tg}\alpha}{1 - \operatorname{tg}^2\alpha}$$

(здесь можно взять любое из двух значений ***tgα***),

$$\cos\alpha = \frac{1}{\mp\sqrt{1 + \operatorname{tg}^2\alpha}}$$

(здесь можно взять любой знак),

$$\sin \boldsymbol{\alpha} = \cos \boldsymbol{\alpha}\ \boldsymbol{tg}\ \boldsymbol{\alpha}.$$

Следовательно, уравнение кривой в новых координатах ***O'x'y'*** примет вид:

$$a'_{11}x'^2 + a'_{22}y'^2 + 2a'_1x' + 2a'_2y' + a'_0 = 0$$

Если в уравнении $\boldsymbol{a'_{11} \cdot a'_{22} > 0}$, ***то*** говорят, что это уравнение определяет линию ***эллиптического типа***; если же $\boldsymbol{a'_{11}\ a'_{22}} < 0$, то говорят, что уравнение определяет линию ***гиперболического типа*** и, если один из коэффициентов $\boldsymbol{a'_{11}}$ или $\boldsymbol{a'_{22}}$ равен нулю, то уравнение определяет линию ***параболического типа.*** Далее с помощью параллельного переноса системы координат ***O'x'y'*** уравнение всегда можно привести к виду:

$$a'_{11}x''^2 + a'_{22}y''^2 + a''_0 = 0$$

т.е. фактически к каноническому виду.

Из уравнения следует, что мы имеем либо ***эллипс*** (если a'_{11} и a'_{22} одного знака, а a'_{11} или a''_0 противоположного), либо ***мнимое место точек*** (если a'_{11}, a'_{22}, a''_0 имеют один знак), либо ***одну точку*** (если a'_{11} и a'_{22} имеют один знак, а $a''_0 = 0$), либо ***гиперболу*** (если a'_{11} и a'_{22} разных знаков и $a''_0 \neq 0$), либо ***две пересекающие прямые*** (если a'_{11} и a'_{22} разных знаков и $a''_0 = 0$).

Если же в уравнении один из коэффициентов a'_{11} и a'_{22}, например, a'_{22} обращается в нуль, то это уравнение с помощью переноса осей приведется к каноническому уравнению параболы $y'' = 2px''^2$ при $a'_{22} \neq 0$ или к виду $ax''^2 + d = 0$ при $a'_{22}=0$, что дает или две параллельные прямые, или мнимое место точек.

Отсюда следует, что всякая кривая второго порядка есть либо эллипс, либо гипербола, либо парабола, либо представляет собой их "вырождение".

Пример. Привести к каноническому виду уравнение кривой второго порядка

$$29x^2 - 24xy + 36y^2 + 82x - 96y - 91 = 0 .$$

Решение

Здесь $a_{11} = 29$, $a_{12} = -12$, $a_{22} = 36$.

Поэтому

$$\operatorname{tg} 2\alpha = \frac{-24}{29-36} = \frac{24}{7},$$

Откуда

$$\frac{24}{7} = \frac{2\operatorname{tg}\alpha}{1-\operatorname{tg}^2\alpha}$$

Тогда

$$12\operatorname{tg}^2\alpha + 7\operatorname{tg}\alpha - 12 = 0$$

решая последнее уравнение, получим

$$tg\,\alpha = -\frac{4}{3} \;\; и \;\; tg\,\alpha = \frac{3}{4}$$

Выбираем: пусть $\operatorname{tg}\alpha = \frac{3}{4}$ тогда $\cos\alpha = \pm\frac{4}{5}$

Выбираем: пусть $\cos\alpha = \frac{4}{5}$ тогда $\sin\alpha = \frac{3}{5}$.

И формулы преобразования координат запишутся в виде:

$$x = \frac{4}{5}x' - \frac{3}{5}y'$$

$$y = \frac{3}{5}x' + \frac{4}{5}y'.$$

Подставляем выражения "старых" координат через "новые" в исходное уравнение кривой и, проделав достаточно громоздкие, но простые преобразования, получаем:

$$20x'^2 + 45y'^2 + 8x' - 126y' - 91 = 0$$

или, выделяя полный квадрат по x' и y' можем записать:

$$20(x+\frac{1}{5})^2+45(y-\frac{7}{5})^2=180\,,$$

отсюда:

$$\frac{(x'+\frac{1}{5})^2}{9}+\frac{(y-\frac{7}{5})^2}{4}=1\,.$$

Введем новые координаты

$$x''=x'+\frac{1}{5},\qquad y''=y'-\frac{7}{5},$$

и в этих координатах уравнение примет вид

$$\frac{x''^2}{3^2}+\frac{y''^2}{2^2}=1\,,$$

т.е. данная кривая есть эллипс с полуосями a=3 и b=2 (рис. 4.3).

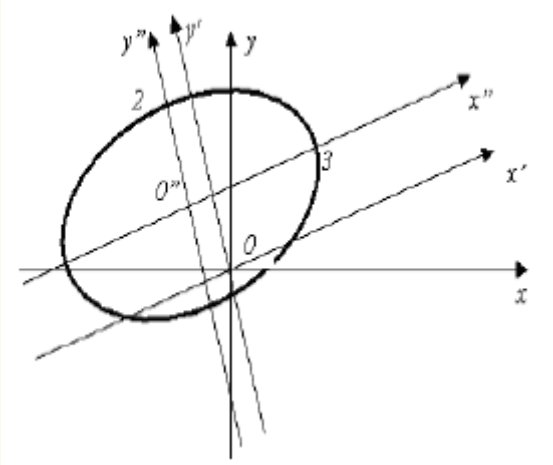


Рис. 4.3

4.3. Вычерчивание окружностей с использованием алгоритма Брезенхама

Для представления окружности, как и любой другой линии, могут быть использованы явные, неявные методы и параметрические методы.

Окружность – это кривая второго порядка. Самым быстрым и легким является способ вычерчивания окружностей, основанный также на алгоритме Брезенхама, похожем на алгоритм вычерчивания линий. Данный метод также не требует вычислений с плавающей точкой. Поэтому он обеспечивает достаточное быстродействие. Алгоритм основан на

приращении координат x и y на величину погрешности между ними. Значение погрешности присваивается переменной δ (в программе – delta). Полученная функция выполняет запись точек по окружности, используя свойство симметрии окружности, что позволяет вычислять точки только на дуге 45°.

При этом x меняется от x = 0 до x = y = r / v2 (тогда y = r … r / v2), где r – радиус (рис. 4.4).

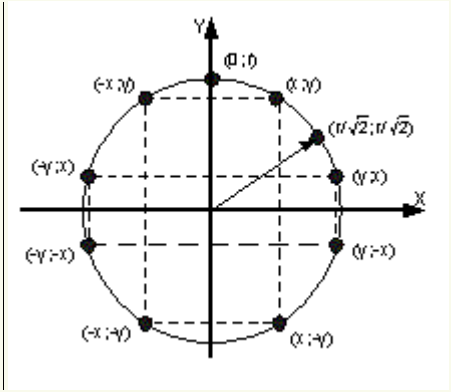


Рис. 4.4

Исходные данные задачи:

1) Радиус окружности – r (целое число).

2) Для простоты положим, что центр окружности находится в точке (0,0) (рис. 4.5).

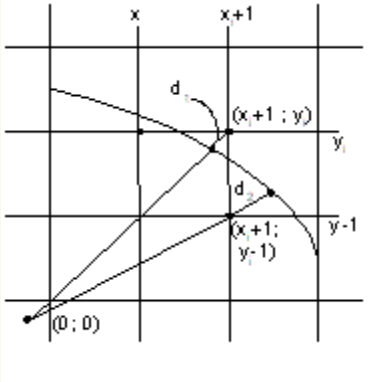


Рис. 4.5

3) Начальная и конечная точка дуги окружности: xst = 0, xen = r/$\sqrt{2}$ и yst = r, yen = r/$\sqrt{2}$.

4) Тогда dx = (xen – xst) = r/$\sqrt{2}$; dy = (yst – yen) = (r - r/$\sqrt{2}$); y – убывает, x – возрастает.

5) Так как dx>dy, то координата x должна получать приращение на каждом шаге алгоритма, а y – не на каждом.

Предположим, что точка (x_i, y_i) уже определена. Так как dx > dy, то следующее значение координаты x равно (x_i+1). Теперь мы должны определить, какой пиксель засвечивать (какой расположен ближе к окружности): (x_i+1, y_i) или (x_i+1, y_i -1) (рис. 4.6).

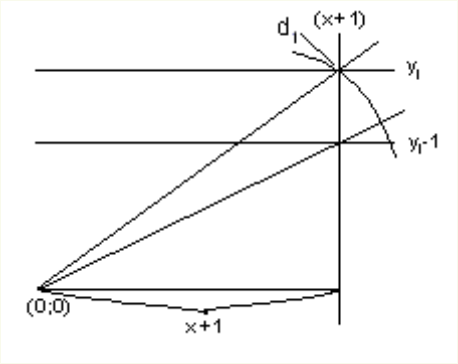


Рис. 4.6

Обозначим d_1, d_2 разницу между идеальным и действительным положением пикселя окружности.

Определим:

$d_1 \approx (x_i+1)^2 + y_i^2 - r^2 \quad > 0.$

$d_2 \approx (x_i+1)^2 + (y_i-1)^2 - r^2 < 0.$

Необходимо определить $\delta_i = |d_1| - |d_2|$.

Если $\delta_i < 0$, то ($d_1 < d_2$) и окружность проходит ближе к верхней точке, тогда следующим должен быть засвечен пиксель (x_i+1; y_i).

Если же $\delta_i >= 0$, то следующий пиксель, который должен быть засвечен, - (x_i+1; y_i-1).

Тогда засвечивается соответствующий пиксель, вычисляется следующее значение δ (т.е. δ_{i+1}) и алгоритм повторяется на следующем шаге.

Определим:

$\delta_i = |d_1| - |d_2| = (d_1 + d_2) = (x_i+1)^2 + y_i^2 - r^2 + (x_i+1)^2 + (y_i-1)^2 - r^2$. (1)

Так как d_1 и d_2 имеют разные знаки, то $|d_1|-|d_2|$ - это то же самое, что (d_1+d_2).

Вычислим δ_1 (начальное значение) из выражения для δ_i (1), учитывая, что ($x_1 = 0$; $y_1 = r$):

$\delta_1 = (0+1)^2 + r^2 - r^2 + (0+1)^2 + (r-1)^2 - r^2 = 3 - 2r$.

Далее определим рекуррентные формулы для δ_{i+1} через вычисленные значения δ_i.

Если $\delta_i < 0$, то на следующем шаге остается то же значение y_i; тогда из (1) следует:

$\delta_{i+1}= (x_i+2)^2 + y_i^2 - r^2 + (x_i + 2)^2 + (y_i-1)^2 - r^2$. (2)

Для получения рекуррентного выражения вычислим:

$\delta_{i+1}= \delta_i +((2)-(1)) = \delta_i + 4\ x_i + 6$.

Если же $\delta_i>=0$, то на следующем шаге y_i уменьшается на единицу; тогда из (1):

$\delta_{i+1}= (x_i+2)^2 + (y_i - 1)^2 - r^2 + (x_i + 2)^2 + (y_i-2)^2 - r^2$. (3)

Для получения рекуррентного выражения вычислим:

$\delta_{i+1}= \delta_i +((3)-(1)) = \delta_i + 4\ x_i - 4y_i + 10$.

То есть рекуррентные формулы для $\delta_i = (d_1 + d_2)$:

$$\delta_{i+1} = \delta_i + 6 + 4x_i, \text{ если } \delta_i < 0$$

$$\delta_{i+1} = \delta_i + 10 + 4x_i - 4y_i, \text{ если } \delta_i >= 0$$

Итак, алгоритм:

- вычисляем $\delta_1=(3 - 2r)$ и засвечиваем начальный пиксель (x = 0, y = r);
- вычисляем δ_{i+1} по значению δ_i и в зависимости от него засвечиваем (x_i+1,y_{i+1}); по одной (каждой) вычисленной точке засвечиваем сразу 8 симметричных пикселей;
- завершаем алгоритм по достижении $x_i = r/\sqrt{2}$.

Пример: программа, реализующая данный алгоритм.

cX, cY – координаты центра окружности;
rd – радиус окружности;
cl – цвет окружности.

```
// Вычерчивание окружности с использованием алгоритма Брезенхама
void circle (cX, cY, rd, cl)
int cX, cY, rd, cl;
{
  #define P(xx,yy) putpixel (xx + cX, yy + cY, cl)
  register x, y, delta;
  y = rd;
```

```
  delta = 3 - 2*rd;
  for (x=0; x<y; x++)
    {
       if (delta < 0) delta += 4*x + 6;
       else {delta += 4*(x - y) + 10; y--; }
      P( x, y);   P( y, x);   P( x,-y);   P( y,-x);
      P(-x, y);   P(-y, x);  P(-x,-y);   P(-y,-x);
     }
}
```

Рисование круга (заполненной окружности) путем повторного вызова circle с уменьшением радиуса:

```
void fill_circle (x, y, rd, cl)
int x, y, r, cl;
{
    while (r) {circle (x, y, r, cl); r--; }
}
```

Если вставить в формулы некоторый коэффициент, получим эллипс вместо окружности.

4.4. Сплайны

При решении задач двухмерной компьютерной графики часто требуется построить гладкую кривую или поверхность по набору заданных точек. Один из наиболее эффективных (по качеству получаемого результата и вычислительным затратам) методов решения этой задачи – применение сплайнов. Сплайн - это очень гладкая кривая. Математически гладкость кривых выражается в терминах непрерывности параметрических представлений x(t) и y(t) и их производных.

Типы кривых с разрывными производными приведены на рис.4.7.

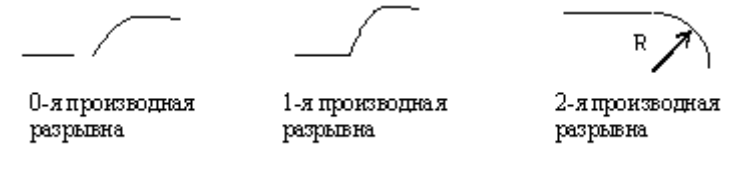


Рис.4.7

Кривые типа B-сплайна (кубический сплайн) обладают свойством непрерывности даже вторых производных x″(t) и y″ (t) в точках стыковки двух соседних сегментов кривой.

Для получения кривых с неразрывной 3-й производной необходимо использовать сплайн не ниже 4-й степени, что приводит к резкому возрастанию вычислительных затрат, но не дает существенного улучшения визуального эффекта.

Для кривой на плоскости при использовании кубических B-сплайнов из заданной последовательности точек выбираются две соседние точки и между ними строится кривая кубического полинома на основе позиции четырех точек – двух выбранных и двух соседних с ними. B-сплайны обеспечивают получение более гладких кривых, чем другие способы сглаживания, за счет того, что получаемые кривые не проходят точно через заданные точки.

Рассмотрим практическое использование B-сплайнов, при котором, как и при представлении большинства сложных кривых в компьютерной графике, обычно используется параметрическое представление. В этом случае любая точка на части кривой, лежащая между двумя заданными последовательными точками P_i и P_{i+1}, будет иметь координаты (x(t), y(t)), где значение t меняется от 0,0 до 1,0, если строится часть кривой от точки P_i до точки P_{i+1} (рис. 4.8). Параметр t может интерпретироваться как нормированное время построения кривой между двумя соседними точками.

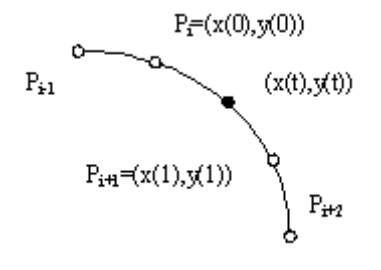


Рис.4.8

Если есть заданные точки $P_0(x_0,y_0)$, $P_1(x_1,y_1)$, … $P_n(x_n,y_n)$, то часть кривой B-сплайна между точками P_i и P_{i+1} определяется последовательностью точек с координатами (x(t), y(t)), при 0.0 <= t <= 1.0:

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3;$$
$$y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3.$$

Эти уравнения содержат коэффициенты:

$a_0 = (x_{i-1} + 4x_i + x_{i+1})/6;$

$a_1 = (-x_{i-1} + x_{i+1})/2;$

$a_2 = (x_{i-1} - 2x_i + x_{i+1})/2;$

$a_3 = (-x_{i-1} + 3x_i - 3x_{i+1} + x_{i+2})/6.$

Коэффициенты b_0, …, b_3 вычисляются аналогично по значениям y_{i-1}, …, y_{i+2}. Для производительности алгоритма важно, что a_i и b_i вычисляются только один раз для каждого сегмента кривой.

Для определения свойств кривой в точках стыковки двух сегментов рассмотрим функцию x(t) и ее первую и вторую производные для значений t=0 и t=1. Функция y(t) будет обладать аналогичными свойствами.

$x(0) = a_0 = (x_{i-1} + 4x_i + x_{i+1})/6;$

$x(1) = a_0 + a_1 + a_2 + a_3 = (x_i + 4x_{i+1} + x_{i+2})/6.$

То есть значение x(0) не равно в точности x-координате x_i точки P_i: оно зависит от позиций точек P_{i-1} и P_{i+1}. Поэтому опорные точки не лежат на кривой – это точки притяжения (управляющие точки), которые задают форму кривой, обеспечивая ее гладкость.

Для трех последовательных точек на кривой A, B и C (рис. 4.9) точка B принадлежит сегменту AB и одновременно сегменту BC. Для первого сегмента $A = P_i$, $B = P_{i+1}$, $C = P_{i+2}$. Тогда $\tilde{x}_B = x(1) = (x_A + 4x_B + x_C)/6$, где $\tilde{x}_B$ – вычисленное значение координаты x для точки B.

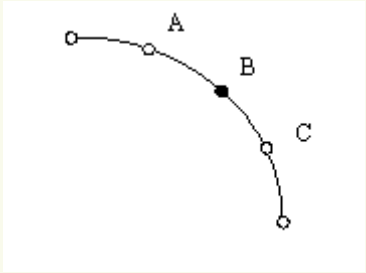


Рис.4.9

Для второго сегмента $A = P_{i-1}$, $B = P_i$, $C = P_{i+1}$ и $\tilde{x}_B = x(0) = (x_A + 4x_B + x_C)/6$. Одинаковые результаты означают непрерывность функции x(t) в точке B. Продифференцировав x(t) дважды, найдем $x'(t)$ и $x''(t)$:

$x'(t) = a_1 + 2a_2t + 3a_3t^2$;

$x''(t) = 2a_2 + 6a_3t$.

Подставляя в них значения t=0 и t=1, находим:

$x'(0) = a_1 = (-x_{i-1} + x_{i+1})/2$;

$x'(1) = a_1 + 2a_2 + 3a_3 = (-x_i + x_{i+2})/2$;

$x''(0) = 2a_2 = x_{i-1} - 2x_i + x_{i+1}$;

$x''(1) = 2a_2 + 6a_3 = x_i - 2x_{i+1} + x_{i+2}$.

Тогда значение первой производной в точке B, вычисленное для первого сегмента кривой, $(\tilde{x}')_B = x'(1) = (-x_A + x_C)/2$ совпадает со значением первой производной в той же точке, вычисленным для следующего сегмента, $(x')\tilde{}_B = x'(0)$.

Точно так же и значение второй производной в точке B, вычисленное для первого сегмента кривой, $(\tilde{x}'')_B = x''(1) = x_A - 2x_B + x_C$ совпадает со значением второй производной в той же точке, вычисленным для следующего сегмента, $(\tilde{x}'')_B = x''(0)$.

Таким образом, не только сама кривая, но и ее первые две производные непрерывны в каждой точке кривой, то есть вся кривая является очень гладкой.

Для расчета любого сегмента кривой между точками P_i и P_{i+1} используются также точки P_{i-1} и P_{i+2}. Тогда если есть точки P_0, P_1, ..., P_{n-1} и P_n, то первый сегмент кривой будет располагаться между точками P_1 и P_2, а последний – между точками P_{n-2} и P_{n-1}. То есть начальной и конечной точками кривой будут P_1 и P_{n-1}, а не P_0 и P_n. Поэтому *для замкнутой кривой* следует три точки сначала или в конце последовательности задавать дважды.

Рассмотрим простую программу построения кубического сплайна. Она считывает числа n, x_0, y_0, x_1, y_1,…, x_n, y_n из файла spline.dat. При выводе каждая из n+1 точек обозначается маркером в виде крестика на экране. Затем вычерчивается кривая B-сплайна.

```
void spline (void)
{
#define MAX 100       // Максимальное число выводимых точек
#define N 30          // Число точек разбиения каждого сегмента кривой
double x[MAX], y[MAX];
double xA, xB, xC, xD, yA, yB, yC, yD, t;
double a0, a1, a2, a3, b0, b1, b2, b3;
int X, Y, i, j, n, first;
FILE *fp;
 fp=fopen("spline.dat","r");
 if(fp==NULL) { printf("Нет файла spline.dat\n"); return;}
 fscanf (fp,"%d",&n);
 for(i=0;i<=n;i++)
  if(fscanf(fp,"%f%f",x+i,y+i)<=0)
     { printf("Конец файла spline.dat\n");  return; }
 first=1;                  // Флаг: точка - первая
 for (i=1;i<n-1;i++)
  {xA=x[i-1]; xB=x[i]; xC=x[i+1]; xD=x[i+2];
   yA=y[i-1]; yB=y[i]; yC=y[i+1]; yD=y[i+2];
   a3=(-xA+3*(xB-xC)+xD)/6.0;          a2=(xA-2*xB+xC)/2.0;
   a1=(xC-xA)/2.0;                     a0=(xA+4*xB+xC)/6.0;
```

```
    b3=(-yA+3*(yB-yC)+yD)/6.0;          b2=(yA-2*yB+yC)/2.0;
    b1=(yC-yA)/2.0;                     b0=(yA+4*yB+yC)/6.0;

    for(j=0;j<=N;j++)
     { t=(double)j/(double)N;
       X=(int)(((a3*t+a2)*t+a1)*t+a0);  Y=(int)(((b3*t+b2)*t+b1)*t+b0);
       if (first)  { first=0; MoveTo(X,Y); }
       else LineTo(X,Y);
     }
   }
  fclose(fp);
 }
```

Алгоритм построения сплайна (для n+1 точек: 0..n)

1. Задается число точек n+1 : 0…n (кривая проводится через точки 1…n-1).
2. Цикл по точкам i = 1, … , n-1 (через точки 0 и n сплайн не проводится).
3. Расчет $a_0,\ldots,a_3$ и $b_0,\ldots,b_3$ (производится один раз).
4. Цикл по времени t=0…1 с заданным шагом N.
5. Расчет x(t), y(t) и построение текущего отрезка, если точка не первая.
6. Окончание цикла по t.
7. Окончание цикла по i.

4.5. Кривые Безье

Кривые и поверхности Безье были использованы впервые в компьютерной графике в 1960-х годах, и с тех пор применяются очень широко практически во всех пакетах программ КГ.

Кривые Безье описываются в параметрической форме: $x(t)=P_x(t)$, $y(t)=P_y(t)$.

Значение t выступает как параметр, которому соответствуют координаты некоторой точки линии. Многочлены Безье для $P_x(t)$ и $P_y(t)$ имеют вид:

$$P_x(t) = \sum_{i=1}^{m} C_m^i t^i (1-t)^{m-i} x_i \,,$$

$$P_y(t) = \sum_{i=1}^{m} C_m^i t^i (1-t)^{m-i} x_i \,,$$

где C^i_m – сочетание m по i: $C^i_m = m!/(i!(m-i)!)$; x_i, y_i – координаты управляющих точек P_i. Значение m – степень полинома; оно на единицу меньше количества управляющих точек. Коэффициенты $C^i_m\, t^i\, (1-t)^{m-i}$ неотрицательны и их сумма равна единице:

$$\sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} = t + (1-t)^m = 1 \,.$$

Рассмотрим кривые Безье первых трех степеней. Кривая Безье первой степени (m=1) строится по двум управляющим точкам P_0 и P_1 (рис. 4.10) и вырождается в отрезок прямой линии, так как полином первой степени линеен:

$P(t) = (1-t)\cdot P_0 + t\cdot P_1$.

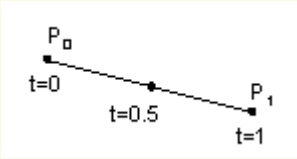


Рис.4.10

Кривая Безье второй степени (m=2) строится по трем управляющим точкам P_0, P_1, P_2 и представляет собой кривую второго порядка (рис. 4.11):

$P(t) = (1-t)^2\cdot P_0 + 2t\cdot(1-t)\cdot P_1 + t^2\cdot P_2$.

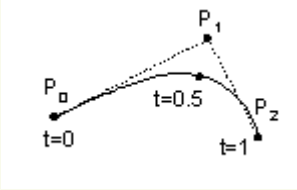


Рис.4.11

Кривая Безье третьей степени (m=3) строится по четырем управляющим точкам P_0, P_1, P_2, P_3 (рис. 4.12) и представляет собой кривую третьего порядка. Она используется в компьютерной графике наиболее часто, так как является очень гладкой и не требует очень больших вычислительных ресурсов для построения:

$P(t) = (1-t)^3 \cdot P_0 + 3t \cdot (1-t)^2 \cdot P_1 + 3t \cdot (1-t)^2 \cdot P_2 + t^3 \cdot P_3$.

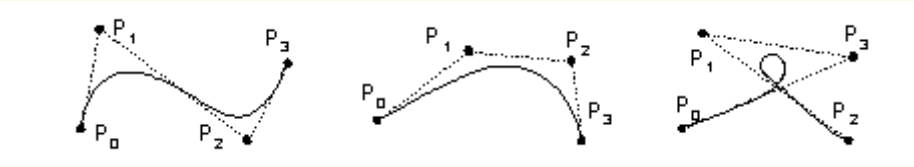


Рис.4.12

Алгоритм построения кривой Безье, позволяющий вычислить координаты (x,y) всех точек кривой по изменяющемуся от 0 до 1 с некоторым шагом значению параметра t:

1. Задается порядок кривой m, координаты управляющих точек (x_i,y_i), i=0, 1,..., m и шаг N.

2. Цикл по параметру t = (0...1) с заданным шагом N.

3. Цикл по параметру j=m, m-1,..., 1

4. Каждая сторона контура многоугольника, построенного вначале по управляющим точкам, делится пропорционально значению t. Число таких точек деления получается вначале

равным порядку кривой m. Точки деления соединяются отрезками прямых, образуя новый многоугольник с числом сторон на единицу меньше, чем на предыдущем шаге.

5. Полученная в результате выполнения в цикле по j шага 4 точка визуализируется и соединяется с предыдущей найденной точкой.

6. Все точки, полученные в результате выполнения в цикле по t шагов 3,4, образуют необходимую кривую Безье.

На рис. 4.13 в качестве примера работы алгоритма приведены построения, необходимые для получения точки кривой Безье второго и третьего порядков для значения параметра t=0,5.

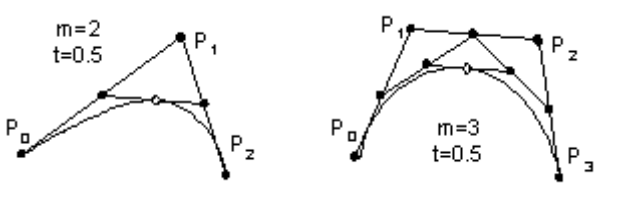


Рис.4.13

Алгоритм расчета одной точки кривой Безье (для текущего значения t), соответствующий шагам 3, 4 и 5, может быть записан на языке С в виде следующего кода:

```
for (k=0; k<=m; k++)
  { xn[k] = x[k];    // Массивы x []  и  y [] - координаты управляющих точек
```

```
    yn[k] = y[k]; } // Массивы xn[] и yn[] - вспомогательные
for (j=m; j>0; j--)
   for (k=0; k<j; k++)
          { xn[k] = xn[k] + t * (xn[k+1] - xn[k]);
            yn[k] = yn[k] + t * (yn[k+1] - yn[k]);  }
```

В результате значения xn[0] и yn[0] будут являться x- и y-координатами текущей (соответствующей текущему значению параметра t) точки кривой Безье.

Кривая Безье обладает полезными свойствами:

1) является гладкой;

2) начинается в точке P_0 и заканчивается в точке P_n (первая и последняя из заданных управляющих точек), касаясь при этом отрезков P_0P_1 и $P_{n-1}P_n$ ломаной линии, построенной через управляющие точки;

3) целиком лежит в выпуклой оболочке, образованной ломаной линией, построенной через управляющие точки.

Основные недостатки кривой Безье:

1) степень функциональных коэффициентов напрямую связана с количеством точек в заданном наборе (на единицу меньше);

2) при добавлении хотя бы одной точки в заданный набор необходимо проводить полный пересчет функциональных коэффициентов в параметрическом уравнении кривой;

3) изменение положения хотя бы одной точки приводит к заметному изменению всей кривой.

В практических вычислениях удобно пользоваться кривыми, составленными из элементарных кусков кривой Безье, как правило кубических. Для соблюдения условия гладкости всей кривой необходимо, чтобы каждые три точки в месте стыковки сегментов лежали на одной прямой (точки P_2, P_3 первого куска и P'_0, P'_1 второго куска, причем точки P_3 и P'_0 должны совпадать).

5.1. Вычерчивание отрезков прямых линий с использованием алгоритма Брезенхама

Функции вычерчивания линий являются основными подпрограммами графики и используются для отображения линий в заданном цвете путем определения координат начальной и конечной точки.

Изображение вертикальных и горизонтальных линий несложно (засвечиваем все пиксели по оси x или y). Труднее создать функцию, рисующую линии вдоль диагоналей, например, от точки (0,0) к точке (8,4) (рис.5.1).

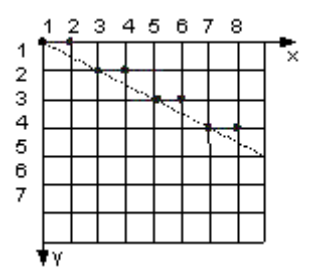


Рис.5.1

Можно вычислять отношение между смещениями по координатам x и y. В данном примере это отношение равно 1/2. То есть приращение координаты x в два раза больше, чем приращение y. Для правильной работы такого алгоритма, чтобы избежать ошибок округления,

необходимо использовать операции с плавающей точкой, что медленно. Если предыдущая точка имеет координаты (x_i, y_i), то следующая за ней точка – координаты (x_{i+1}, y_{i+1}), где $x_{i+1}= x_i+1$; $y_{i+1}= \text{Round}(y_i+0,5)$.

Необходим алгоритм с целочисленной арифметикой и с использованием простых арифметических операций.

Брезенхам разработал в 1965 г. классический алгоритм, использующий только операции с целыми числами. В нем отношение между значениями координат x и y представляется косвенным образом через серии сложений и вычитаний.

Основной идеей алгоритма Брезенхама является регистрация средних значений погрешности между идеальным положением каждой точки и той позицией на экране дисплея, в которой она действительно отображается. Погрешности между идеальным и действительным положением точки возникают из-за дискретности экрана дисплея. Следовательно, действительное положение каждой точки на линии требует наилучшей аппроксимации. В каждой итерации цикла вычерчивания линии вызываются две переменные ex (error x) и ey (error y), которые увеличиваются в зависимости от изменения значений координат x и y соответственно. Когда значение погрешности достигает определенного значения, оно вновь устанавливается в исходное положение, а соответствующий счетчик координат увеличивается. Этот процесс продолжается до тех пор, пока линия не будет полностью вычерчена.

Рассмотрим алгоритм Брезенхама для отрезка прямой. Исходные данные:

1. Отрезок проходит от точки (x_{start},y_{start}) до точки (x_{end},y_{end}).

2. Пусть $x_{start} < x_{end}$, $y_{start} < y_{end}$.
3. Пусть $\Delta x>0$, $\Delta y>0$, где $\Delta x= x_{end} - x_{start}$, $\Delta y= y_{end} - y_{start}$.
4. Для простоты $(x_{start},y_{start}) = (x_1,y_1) = (0,0)$.
Рассмотрим экран на уровне пикселей (рис.5.2).

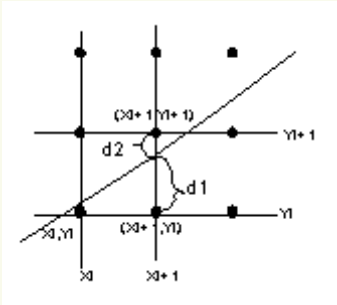


Рис.5.2

Если на экране пиксель (x_i,y_i) уже засветился (так как выводимая прямая линия проходит вблизи точки (x_i,y_i)), какой пиксель должен быть засвечен следующим: (x_i+1,y_i) или (x_i+1,y_i+1)? Для решения этой задачи рассмотрим идеальное положение прямой на экране (без учета его дискретности). Соединим массив пикселей на экране условными отрезками прямых линий и найдем точку пересечения идеальной прямой с отрезком, соединяющим пиксели (x_i+1,y_i) и

(x_i+1,y_i+1). Если d_1>=d_2, то следующим должен быть засвечен пиксель (x_i+1,y_i+1). Если d_1<d_2, то следующий пиксель - (x_i+1,y_i).

То есть если разность (d_1-d_2) >= 0, то засвечивается (x_i+1,y_i+1).

Знак этой разности может быть определен из подобных треугольников (рис 5.3).

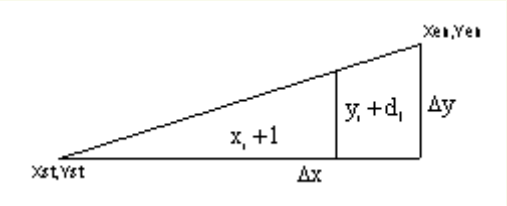


Рис.5.3

Отношения длин соответствующих сторон большого и малого треугольников равны:

$$\frac{\Delta y}{\Delta x} = \frac{y_i + d_1}{x_i + 1},$$

откуда после преобразований получаем:

$d_1 = \Delta y \cdot (x_i+1)(1/\Delta x) - y_i$.

Но поскольку $d_2 = 1 - d_1$, то $d_1-d_2=d_1-1+d_1=2d_1-1$. Подставляя найденное значение d_1, получим $d_1-d_2 = (1/\Delta x)(2\cdot\Delta y\cdot x_i + 2\cdot\Delta y) - 2y_i - 1$.

Умножим обе части равенства на Δx. Так как Δx>0, то знак разности не изменится, поэтому: *Знак* (d_1-d_2) = *Знак* Δx· (d_1-d_2). Обозначим P_i =Δx·(d_1- d_2). Тогда:

$P_i = 2 \cdot \Delta y \cdot x_i + 2 \cdot \Delta y - 2 \cdot \Delta x \cdot y_i - \Delta x$.

Так как

$P_{i+1} = 2 \cdot \Delta y \cdot (x_i + 1) - 2 \cdot \Delta x \cdot y_{i+1} + 2 \cdot \Delta y - \Delta x$, то

$P_{i+1} = P_i + 2 \cdot \Delta y - 2 \cdot \Delta x \cdot y_{i+1} + 2 \cdot \Delta x \cdot y_i$.

Тогда если известно P_i, то

$P_{i+1} = P_i + 2 \cdot (\Delta y - \Delta x)$, если $P_i >= 0$ (так как $y_{i+1} = y_i + 1$);

$P_{i+1} = P_i + 2 \cdot \Delta y$, если $P_i < 0$ (так как $y_{i+1} = y_i$).

При этом первое значение $P_1 = 2 \cdot \Delta y - \Delta x$, так как $(x_1, y_1) = (0,0)$.

Алгоритм:

Засвечиваем точку (x_0, y_0); присваиваем $P_1 = 2 \cdot \Delta y - \Delta x$; в зависимости от полученного значения P_i засвечиваем следующую точку, вычисляем P_{i+1} и т.д.

Пример: программа, реализующая данный алгоритм.

xst, yst – начальные точки;

xen, yen – конечные точки;

xin, yin – приращение x и y;

t – переменная цикла (приращение x или y);

b – переменная цикла (P_i);

cl – цвет линии;

b1, b2 – приращения P_i.

```
// Вычерчивание линии заданного цвета с помощью  алгоритма Брезенхама
void line (xst, yst, xen, yen, cl)
int xst, yst, xen, yen, cl;
{
  register int t, b;
  int dx, dy, b1, b2;
  int xin, yin;
// Вычисление расстояния в обоих направлениях
  dx = xen - xst;
  dy = yen - yst;
// Определение направления шага;
// Шаг вычисляется либо по вертикальной, либо по горизонтальной линии
   if (dx > 0) xin = 1;
   else  if (dx == 0) xin = 0;
         else  xin= -1;
   if (dy > 0) yin = 1;
   else  if (dy == 0) yin = 0;
         else  yin= -1;
// Определение большего расстояния и вычерчивание линии
```

```
dx = abs(dx);
dy = abs(dy);
putpixel (xst, yst, cl);
if (dx>dy)
 { b1 = 2*(dy-dx);
   b2 = 2*dy;
   b   = 2*dy-dx;
   for (t=0; t<dx; t++)
    {
      if (b <=0 ) b += b2;
      else { b += b1; yst += yin;}
      xst += xin;
      putpixel (xst, yst, cl);
    }
  }
else
 { b1 = 2*(dx-dy);
   b2 = 2*dx;
   b  = 2*dx-dy;
   for (t=0; t<dy; t++)
   {
```

```
        if (b <= 0) b += b2;
        else { b += b1; xst += xin;}
        yst += yin;
        putpixel (xst, yst, cl);
      }
    }
}
```

5.2. Устранение лестничного эффекта

Приведенные выше алгоритмы в чистом виде не используются из-за лестничного эффекта (aliasing): при выводе наклонных линий пиксели образуют ступеньки (рис.5.4).

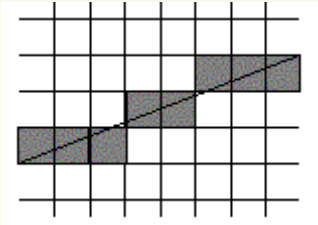


Рис.5.4

Поэтому в алгоритмы Брезенхама вводятся уточняющие методы, позволяющие сгладить лестничный эффект. Такие методы можно разделить на две группы: алгоритмы генерации сглаженных изображений и алгоритмы сглаживания изображений уже сформированных.

Устранение лестничного эффекта (antialiasing) с использованием алгоритмов первой группы связано с изменением интенсивности граничных пикселей.

В обычной ситуации условия корректного вывода линий и других графических объектов можно сформулировать так: если в контур объекта попадает больше половины площади ячейки сетки растра, то соответствующий пиксель закрашивается цветом объекта (например, черным с интенсивностью $I_о=I_ч$), иначе – пиксель сохраняет цвет фона (например, белый с интенсивностью $I_ф=I_б$). На рисунке на растровое изображение толстой прямой линии наложен идеальный контур исходной линии (рис.5.5).

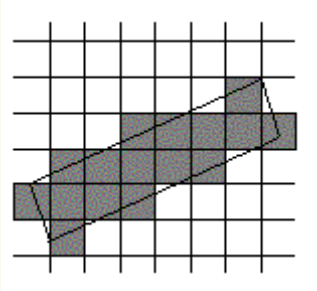


Рис.5.5

Чтобы растровое изображение линии выглядело более гладким, можно интенсивность (яркость) каждой цветовой компоненты граничных пикселей заменить на некоторую интенсивность, промежуточную между интенсивностью объекта и фона. Наиболее корректно вычислять интенсивность пропорционально площади ячейки растра, покрываемой идеальным контуром объекта. Если площадь всей ячейки обозначить как S, а часть площади, покрываемой контуром - S_x, то искомая интенсивность

$$I_x = I_ф + (I_o - I_ф) \cdot S_x/S$$

или, с учетом того что площадь ячейки S=1 (ее длина и высота равны 1 пикселю),

$$I_x = I_ф + (I_o - I_ф) \cdot S_x.$$

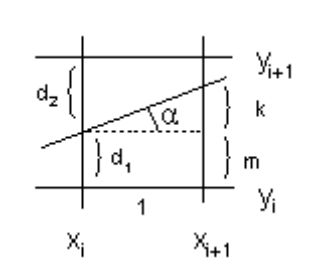


Рис.5.6

Площадь части ячейки растра, лежащей ниже пересекающей линии, S_x = *Площадь прямоугольника* + *Площадь треугольника* (рис.5.6), то есть

$S_x=1\cdot m+1\cdot k/2=m+tg(\alpha)/2=d1+\Delta y/(2\cdot\Delta x)$,

так как $tg(\alpha) = \delta y/\delta x$ (треугольник подобен большому треугольнику со сторонами Δx, Δy – см. алгоритм Брезенхама).

Поэтому окончательно:

$I_x = I_ф + (I_o - I_ф) \cdot [d1 + \Delta y/(2\Delta x)]$.

На рис. 5.7 показано как будет скорректировано растровое изображение с помощью приведенного выше способа.

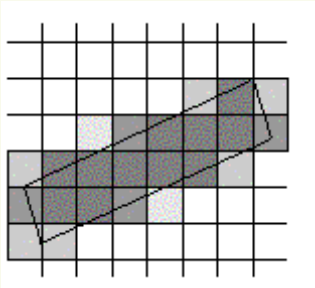


Рис.5.7

Другую группу методов устранения лестничного эффекта составляют алгоритмы сглаживания уже сформированного изображения, обычно методом цифровой фильтрации. Локальная цифровая фильтрация выполняется методом взвешенного суммирования яркостей пикселей, расположенных в некоторой окрестности текущего обрабатываемого пикселя. В ходе действия алгоритма обработки по растру скользит квадратное окно (размер, как правило, 3x3 пикселя, в его центре – текущий пиксель), которое объединяет информацию о пикселях, окружающих текущий пиксель, и на основе этой информации меняется яркость текущего пикселя. Тогда новая интенсивность текущего пикселя, расположенного на i-й строке и j-м столбце, будет такой (для матрицы 3x3):

$$I_{ij} = \frac{1}{K} \sum_{m=i-1}^{i+1} \sum_{n=j-1}^{j+1} I_{mn} M_{m-i+1,n-j+1} \ ,$$

где K – нормирующий коэффициент, M – маска фильтра (матрица коэффициентов 3x3), I_{mn} – интенсивности пикселей.

Один из возможных вариантов сглаживающего фильтра 3x3:

$$\frac{1}{16} \begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix} .$$

При сглаживании цветных изображений фильтрация выполняется отдельно по каждой цветовой компоненте.

5.3. Заполнение области

Задача заключается в закрашивании произвольного контура, нарисованного в растре. Задается область, ограниченная набором пикселей заданного цвета, и точка (x,y), лежащая внутри этой области (рис.5.8).

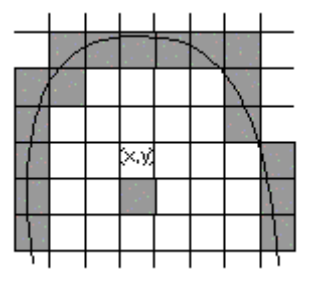


Рис.5.8

Простейший алгоритм заполнения заключается в следующем. Цвет текущего пикселя изменяется на нужный цвет заполнения. Затем проводится анализ цветов всех соседних пикселей. Если цвет некоторого соседнего пикселя не равен цвету границы контура или цвету заполнения, то цвет этого пикселя изменяется на цвет заполнения. Затем анализируется цвет пикселей, соседних с предыдущими. И так далее до тех пор, пока внутри контура все пиксели не перекрасятся в цвет заполнения.

Соседними могут считаться только четыре пикселя (четырехсвязность) или все восемь пикселей (восьмисвязность).

Запись данного рекурсивного алгоритма:

```
ЗАКРАСКА(x,y)
 {
   Если цвет пикселя (x,y) не равен цвету границы, то
    {
      установить для пикселя (x,y) цвет заполнения;
      ЗАКРАСКА(x+1,y);
      ЗАКРАСКА(x-1,y);
      ЗАКРАСКА(x,y+1);
      ЗАКРАСКА(x,y-1);
      }
  }
```

Алгоритм абсолютно корректен, но неэффективен, так как рекурсивная функция вызывается для каждого пикселя (и не обязательно только один раз), что медленно и требует большого размера стека. Более предпочтительными являются алгоритмы, обрабатывающие сразу целые группы пикселей.

Алгоритм закрашивания линиями отличается тем, что на каждом шаге закрашивания рисуется горизонтальная линия, которая размещается между пикселями контура (рис.5.9). По

сравнению с предыдущим рекурсивным алгоритмом здесь количество вложенных вызовов уменьшается пропорционально длине линии.

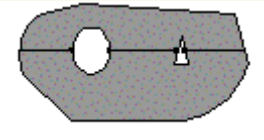


Рис.5.9

Волновой алгоритм закрашивания основан на поиске трассы на графе. Здесь для начальной точки (вершины на графе) находятся соседние точки (другие вершины), которые отвечают двум условиям: эти вершины связаны с начальной и они рассматриваются впервые. Соседние вершины текущей итерации становятся текущими точками для поиска новых соседних вершин в следующей итерации. По достижении конечной точки (границы области) совершается обратный цикл к начальной точке, в ходе которого находятся все кратчайшие пути между двумя заданными точками на графе и определяется порядок закраски пикселей. В отличие от первого простого алгоритма метод не использует рекурсию и генерирует другую последовательность обработки пикселей при закрашивании, в которой каждый пиксель рассматривается только один раз.

5.4. Движущиеся двухмерные объекты

Используя описанные методы построения двухмерных моделей, можно создавать движущиеся изображения объектов. Основной подход здесь заключается в создании серии кадров, каждый из которых несколько отличается от предыдущего и последующего. При быстрой смене кадров сцены сливаются и возникает эффект движения.

Спрайты в двухмерной графике

Изначально под спрайтами понимали небольшие рисунки, которые выводились на экран с применением аппаратного ускорения. На некоторых машинах программная прорисовка приводила к определённым ограничениям, а аппаратные спрайты этого ограничения не имели. Впоследствии с увеличением мощности центрального процессора от аппаратных спрайтов отказались и понятие «спрайт» распространилось на всех двумерных персонажей и спрайтовой считается графика, анимированная с помощью перебора изображений из атласа спрайтов.

К аппаратно-ускоренным спрайтам вернулись, когда развитие мультимедиа и повышение разрешения и глубины цвета потребовало специализированного процессора в видеокарте. В современных программах, использующих спрайтовую графику, двухмерные спрайты выводятся как текстурированный прямоугольник.

Атлас спрайтов является растровым изображением, объединяющим спрайты в набор раскадровок анимаций, или иным комплектом спрайтов. Целесообразность применения заключается в уменьшении количества файлов и применение атласа спрайтов в процессе

рендеринга: видеокарте передаётся одно изображение, набор координат, на которых его необходимо отрендерить и трафаретов, вырезающих из атласа отдельные спрайты. В результате, двухмерная графика рендерится значительно быстрее, чем одиночными передачами видеокарте одиночных спрайтов с указанием координат отрисовки. Для дополнительного сжатия и уменьшения количества неиспользуемого пространства, атлас может комбинировать изображения не сеткой, а в максимально плотно сжатом виде, таким образом, что, если описать вокруг каждого спрайта прямоугольник — часть прямоугольников будет пересекаться. Данный формат требует дополнительной информации о вершинах, которыми можно описать каждый спрайт без пересечения с другими, и увеличивает время загрузки приложения, но позволяет сократить память.

Спрайты являются популярным способом для создания больших и сложных сцен, так как можно манипулировать и управлять каждым изображением в отдельности. Это позволяет лучше контролировать происходящее на сцене.

Совсем не редкость, когда, например, в компьютерной игре существуют десятки и сотни спрайтов. Загрузка каждого из них в качестве индивидуального и самостоятельного изображения будет потреблять много памяти и вычислительной мощности. Для того, чтобы избежать этих неудобств, и используют атласы спрайтов.

Атлас спрайтов

Когда множество спрайтов, являющихся отдельными кадрами (рис. 5.10), объединяется в одно изображение, получается атлас спрайтов (рис. 5.11).

Рис. 5.10

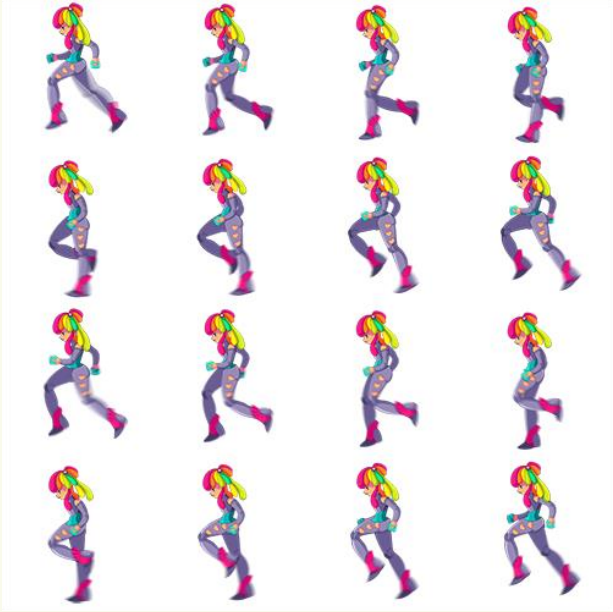


Рис. 5.11

Атласы спрайтов используют для ускорения процесса отображения картинки на экране. Гораздо эффективнее и быстрее загрузить одно большое изображение и показать его часть, чем загрузить множество спрайтов и показать их полностью.

Анимация, основанная на атлас спрайтов, есть не более чем обычный атлас спрайтов, который показывается не целиком, а покадрово. Причем эти кадры меняются с достаточно большой скоростью.

Части атласа спрайтов

Сам по себе атлас спрайтов состоит из двух частей: кадров и циклов. Кадр является обычным изображением (или спрайтом). Если кадры поставить в правильном порядке, то получится непрерывное движение. Это и есть цикл.

Программирование атласа спрайтов анимации

Существует три этапа создания атласа спрайтов анимации:

1. Создание изображения.
2. Обновление изображения для каждого кадра анимации.
3. Рисование анимации на экране.

Создание изображения

Начнем с создания функции (или класса), который будет обрабатывать спрайтовую

анимацию. Эта функция будет создавать изображение и установит его путь для того, чтобы можно было использовать его для дальнейшего рисования на экране.

```
function SpriteSheet(path, frameWidth, frameHeight) {
   var image = new Image();
   var framesPerRow;
   // вычисление количества кадров в строке после загрузки
изображения
   var self = this;
   image.onload = function() {
      framesPerRow = Math.floor(image.width / frameWidth);
   };
   image.src = path;
}
```

Так как атласы спрайтов могут иметь разные размеры, то, зная размер одного кадра, можно вычислить количество кадров в строке и столбце. Очень важно, чтобы каждый кадр анимации имел один и тот же размер, иначе анимация получится неестественная.

Обновление изображений

Чтобы обновить атлас спрайтов анимацию, понадобится изменить тот кадр, который рисуется в данный момент. Ниже атлас спрайтов разделен на кадры, каждый из которых

пронумерован (рис.5.12).

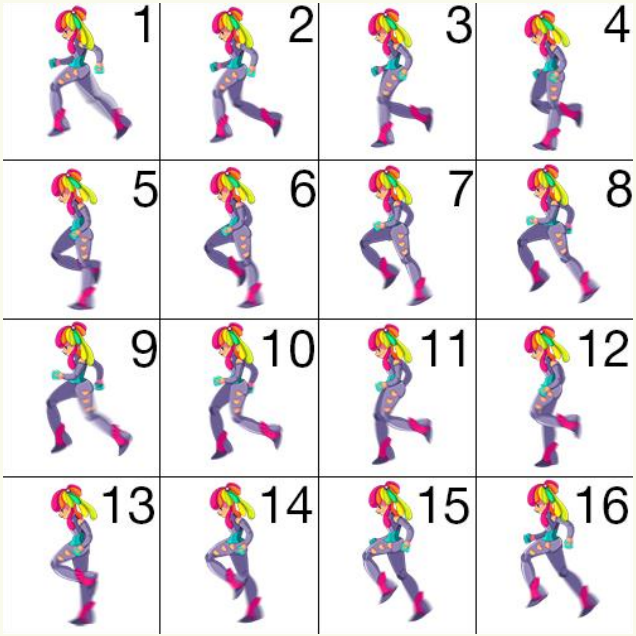


Рис. 5.12

Менять текущий кадр на следующий нужно не моментально, а через какое-то время. То есть при показе одного из кадров требуется сделать определенную задержку перед тем, как поменять его.

Не каждый атлас спрайтов имеет доступный спрайт в каждом из своих кадров. Если это не будет учитываться, то анимация получится со вспышкой. Избежать всплесков можно просто: не отрисовывать эти пустые изображения.

```
    function SpriteSheet(path, frameWidth, frameHeight, frameSpeed,
endFrame) {
      var currentFrame = 0;  // текущий кадр для отрисовки
      var counter = 0;       // счетчик ожидания
      // обновление анимации
      this.update = function() {
        // если подошло время смены кадра, то меняем
        if (counter == (frameSpeed - 1))
          currentFrame = (currentFrame + 1) % endFrame;
        // обновление счетчика ожидания
        counter = (counter + 1) % frameSpeed;
        }
    };
```

При использовании операции по взятию остатка от деления (%) для `currentFrame` мы создаем непрерывный цикл чисел от 0 до `endFrame`. Таким образом воспроизводятся только те кадры, изображения для которых есть в атласе спрайтов.

Рисование анимации

Строка с нужным кадром вычисляется взятием остатка от деления номера текущего кадра на количество кадров в строке. Также вычисляется столбец с нужным кадром путем деления номера текущего кадра на количество кадров в строке.

Используя найденные величины, можно найти координаты той рамки, внутри которой и находится искомый кадр:

```
// отрисовка текущего кадра
this.draw = function(x, y) {
   // вычисление столбика и строчки с нужным кадром
   var row = Math.floor(currentFrame / framesPerRow);
   var col = Math.floor(currentFrame % framesPerRow);
   ctx.drawImage(
      image,
      col * frameWidth, row * frameHeight,
      frameWidth, frameHeight,
      x, y,
      frameWidth, frameHeight);
```

```
    }
```

Теперь можно создать анимацию:

```
    spritesheet = new SpriteSheet('Walk_Cycle_Image.png', 125, 125,
3, 16);
    function animate() {
       requestAnimFrame( animate );
       ctx.clearRect(0, 0, 150, 150);
       spritesheet.update();
       spritesheet.draw(12.5, 12.5);
    }
```

Для формирования движущихся изображений и работы со спрайтами обычно используются геометрические преобразования.

5.5. Восприятие изображений графических объектов

Программы компьютерной графики отображают синтезированное изображение на экране дисплея; в отличие от фото- и видеосистем источник информации здесь не физические процессы или объекты, а математические модели. Эти модели в общем случае – упорядоченная совокупность данных, числовых характеристик, параметров, математических

и логических зависимостей, отображающих структуру, свойства, взаимосвязи и отношения между элементами объекта, объектом и окружением. Математические модели обычно являются обобщенными и предназначаются для определенного класса объектов. При вводе конкретных значений параметров система компьютерной графики на основе общей модели синтезирует изображение и визуализирует его.

Специфика моделей и обрабатывающих программ в компьютерной графике обусловлена необходимостью передавать ощущение глубины пространства, пространственной формы и структуры объектов. Существенное влияние на построение математической модели оказывает требуемый *уровень соответствия (подобия)* синтезируемого изображения визуально наблюдаемой картине или изображению с камеры (оригиналу). Существует три уровня подобия:

1. Физическое подобие: изображение по основным физическим характеристикам повторяет оригинал. Устанавливается на уровне трех групп характеристик:

а) геометрические (совпадают пропорции);

б) цветовые и яркостные;

в) временные (для движущихся объектов).

При физическом подобии все параметры изображения должны полностью соответствовать параметрам оригинала либо быть пропорциональны им.

2. Психофизическое подобие: соответствие на уровне зрительных ощущений. В силу ограниченных возможностей зрительного аппарата наблюдатель при некотором уровне искажений не может ощутить разницы между синтезированным изображением и оригиналом,

т.к. зрительные ощущения идентичны, хотя яркость, форма и цвет одноименных участков не одинаковы.

3. Психологическое подобие: по общему восприятию изображение и оригинал являются сходными; изображение дает наблюдателю вполне определенные суждения о реальном объекте или сюжете, хотя синтезированное изображение и оригинал значительно различаются по физическим характеристикам.

Достижение уровней подобия.

Синтез изображения на уровне *психологического подобия* широко используется в компьютерной графике. Приемы передачи объемности и глубины пространства схожи с используемом в изобразительном искусстве: формированием очертаний предмета и характерных линий, передающих его объемность, загораживанием одних предметов другими, передачей перспектив. Эти приемы находят в компьютерной графике свое математическое выражение: алгоритмы формирования сечений объемной фигуры, удаления невидимых линий, формирование перспективной проекции, яркостное или цветовое выделение некоторых элементов изображения, моделирование теней. Но компьютерной графика использует меньше приемов, чем искусство, так как алгоритмическая и программная реализации таких приемов затруднены. К числу задач синтеза на уровне психологического подобия может быть отнесено формирование чертежей в аксонометрии.

Достижение *психофизического подобия* – одна из основных задач компьютерной графики, которая возникает при имитации визуально наблюдаемых явлений на тренажерах,

создании банков эталонных изображений для систем распознавания, оценке внешнего вида и эстетических свойств объектов и др.

В общем виде процедура синтеза изображения $G_И = A_{МГ} \cdot G_М$, где $G_М$ – математическая модель объекта (сцены), $A_{МГ}$ – оператор преобразования системы компьютерной графики, $G_И$ – синтезированное системой компьютерной графики 2D-изображение.

Синтезированное изображение должно сопоставляться с оригиналом: $G_{ОР} = A_В \cdot G_{0,}$ где G_0 – поле излучения объекта, $A_В$ – оператор преобразования при визуальном наблюдении или в съемочной аппаратуре, $G_{ОР}$ – изображение-оригинал.

При физическом подобии синтезированное изображение и оригинал должны быть идентичны друг другу: $G_И \leftrightarrow G_{ОР}$. Это означает, что в синтезированном изображении и оригинале должны быть одинаковыми (или пропорциональными) ракурсы наблюдения, яркостные и цветовые соотношения, тени, текстура поверхностей, масштабы и др.

Условия физического подобия при синтезе принципиально могут быть выполнены, если оригинал получен на съемочной аппаратуре. Для оптической аппаратуры разработаны достаточно точные математические модели, описывающие оператор преобразования $A_В$. Для выполнения синтеза необходимо сформировать модель объекта (сцены) $G_М$, в которой должны быть отражены источники освещения, трехмерная геометрическая форма, взаимное положение объектов, характеристики отраженного и собственного излучений, положение съемочной аппаратуры. (При разработке модели сцены должен быть учтен процесс преобразования информации в съемочной аппаратуре; если аппаратура не воспроизводит

какие-либо особенности наблюдаемых объектов – мелкие детали, текстуру, цвет, то их учет в модели избыточен).

Если синтезируется изображение, аналогичное визуально наблюдаемому, то условия физического подобия практически не могут быть реализованы. Системы компьютерной графики синтезируют 2D-изображения, в которых теряется информация, определяемая бинокулярным зрением. Если объект – объемное тело, то у наблюдателя образуется два сетчаточных образа в левом и правом глазу. Оптическая система каждого глаза строит на сетчатке изображение. Центром проекции является центр зрачка глаза. Вследствие пространственного разноса глаз их оптические оси сходятся на объекте под некоторым углом (угол конвергенции). Поэтому сетчаточные изображения в правом и левом глазу не тождественны. Совместное действие этих сетчаточных изображений формирует у наблюдателя зрительное восприятие объема и пространства.

Система компьютерной графики может синтезировать изображение, соответствующее только одному сетчаточному изображению, то есть система способна воспроизвести лишь условия монокулярного наблюдения. Объемность изображения, пространственное положение объектов при этом воспринимаются благодаря линейной перспективе, взаимному загораживанию объектов, характеру теней и изменению тона по полю изображения. Существенное значение здесь имеет предшествующий опыт наблюдения, то есть наблюдатель непроизвольно «достраивает» объемную структуру наблюдаемой сцены.

То есть синтезированное компьютерное изображение может быть соответствующим оригиналу при визуальном наблюдении только на психофизическом уровне (здесь не берутся во внимание аппаратные системы виртуальной реальности).

Максимальное приближение к оригиналу при этом, как и в случае аппаратурной съемки, можно обеспечить, если математическая модель сцены и обрабатывающая программа точно передают условия освещения, геометрическую форму объектов, их взаимное положение, размер и положение теней, и другие особенности сцены.

Более подробно особенности восприятия человеком графической информации изучаются в информационном дизайне: восприятие графической сцены и отдельных ее элементов, особенности восприятия цветов и линий и др.

6.1. Поточечная обработка изображений

Одной из основных задач компьютерной графики является придание изображению таких качеств, улучшающих его восприятие человеком. Часто бывает полезным подчеркнуть, усилить какие-то особенности наблюдаемой сцены с целью улучшения ее субъективного восприятия. Подавляющее большинство процедур обработки для получения результата в каждой точке кадра привлекает входные данные из некоторого множества точек исходного изображения, окружающих обрабатываемую точку. Однако имеется группа процедур, где осуществляется так называемая поточечная обработка. Здесь результат обработки в любой точке кадра зависит только от значения входного изображения в этой же точке.

Сущность поэлементной обработки изображений сводится к следующему. Пусть $f(x,y)=f_{x,y}$ и $g(x,y)=g_{x,y}$ – значения яркости исходного и полученного после обработки изображения в точке кадра, имеющей декартовы координаты x (номер строки) и y (номер столбца) соответственно. Поэлементная обработка означает, что существует однозначная функциональная зависимость между этими яркостями:

$$g_{x,y} = h_{x,y}\,(f_{x,y}),$$

позволяющая по значению исходного сигнала определить значение выходного сигнала.

Линейное контрастирование изображения (линейная коррекция). Задача контрастирования связана с улучшением согласования динамического диапазона изображения

и экрана, на котором выполняется визуализация. Если для цифрового представления каждого отсчета изображения отводится 1 байт (8 бит) запоминающего устройства, то входной или выходной сигналы могут принимать одно из 256 значений. Обычно в качестве рабочего используется диапазон 0-255; при этом значение 0 при визуализации соответствует уровню черного, а значение 255 – уровню белого. Предположим, что минимальная и максимальная яркости исходного изображения равны $f_{\min}$ и $f_{\max}$ соответственно. Если эти параметры, или один из них, существенно отличаются от граничных значений яркостного диапазона, то визуализированная картина выглядит как ненасыщенная, неудобная, утомляющая при наблюдении. При линейном контрастировании используется линейное поэлементное преобразование вида

$$g = a \cdot f + b,$$

параметры которого (a и b) определяются желаемыми значениями минимальной $g_{\min}$ и максимальной $g_{\max}$ выходной яркости. Решив систему уравнений

$$\begin{cases} g_{\min} = a \cdot f_{\min} + b, \\ g_{\max} = a \cdot f_{\max} + b, \end{cases}$$

относительно параметров преобразования a и b, нетрудно привести её к виду

$$g = \frac{f - f_{min}}{f_{max} - f_{min}}(g_{max} - g_{min}) + g_{min}\,.$$

Нелинейное контрастирование изображения (нелинейная коррекция). Линейная коррекция изображения не всегда дает необходимый результат (например, там, где наблюдается неравномерная освещенность изображения), поэтому в ряде случаев применяется нелинейная коррекция изображения. Часто применяются следующие функции.

Гамма-коррекция, изначальная цель которой – коррекция для правильного отображения на мониторе:

$$g = c \cdot f^{\gamma},$$

где c и γ – константы.

Некоторые компьютерные системы имеют встроенную частичную гамма-коррекцию. Кроме этого, гамма-коррекция используется для универсального управления контрастом, в частности при обработке медицинских изображений. В зависимости от значения γ возможно целое семейство преобразований.

Логарифмическая коррекция, цель которой – сжатие динамического диапазона при визуализации данных:

$$g = c \cdot \log(1+f) ,$$

е c – константа.

Использование логарифма позволяет узкий диапазон малых значений яркости преобразовать в более широкий диапазон выходных значений. Для больших значений входного сигнала верно противоположное утверждение. Такой тип преобразования используется для растяжения диапазона значений темных пикселей на изображении с одновременным сжатием диапазона значений ярких пикселей. При использовании обратного логарифмического преобразования происходит растяжение диапазона значений ярких пикселей на изображении с одновременным сжатием диапазона значений темных пикселей. Логарифмическая функция имеет важную особенность – позволяет сжимать динамический диапазон изображений, имеющих большие вариации в значениях пикселей.

Зависимость яркости исходного и результирующего изображений при линейном и нелинейном контрастировании представлена на рис. 6.1.

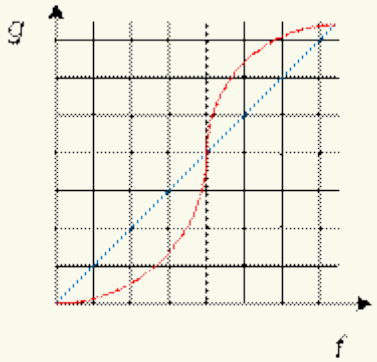
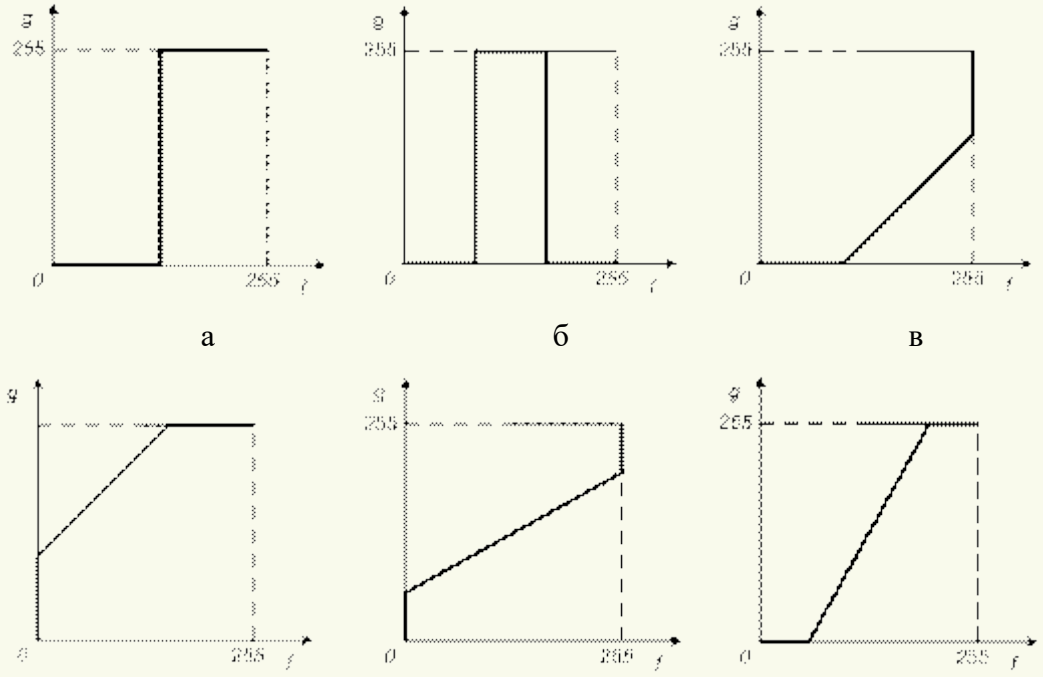


Рис. 6.1

Препарирование изображения. Препарирование представляет собой целый класс поэлементных преобразований изображений (рис. 6.2, а-е).



а б в

г д е

Рис. 6.2

Преобразование с пороговой характеристикой (см. рис. 6.2, а) превращает полутоновое изображение, содержащее все уровни яркости, в бинарное, точки которого имеют яркости g=0 или $g=g_{max}$. Такая операция, называемая ***бинаризацией***, может быть полезной, когда для наблюдателя важны очертания объектов, присутствующих на изображении, а детали, содержащиеся внутри объектов или внутри фона, не представляют интереса.

На рис. 6.2 б преобразование выполняет ***яркостный срез*** изображения, выделяя те его участки, где яркость соответствует выделенному интервалу. При этом остальные участки оказываются полностью «погашенными» (имеют яркость, соответствующую уровню черного). Перемещая выделенный интервал по яркостной шкале и изменяя его ширину, можно детально исследовать содержание картины.

На рис. 6.2 в-е преобразования выполняют процедуры изменения ***яркости*** и ***контраста*** изображения. Преобразования, показанные на рис. 6.2, в, г, позволяют, соответственно, понизить или повысить яркость редактируемого изображения, используя следующие соотношения входного и выходного диапазона яркости изображения:

$f_{min} = 255 - g_{max}$, $g_{min} = 0$, $f_{max} = 255$ (понижение яркости);

$g_{min} = 255 - f_{max}$, $f_{min} = 0$, $g_{max} = 255$ (повышение яркости);

Преобразования, показанные на рис. 6.2 в-е, позволяют повысить ***контраст*** наблюдаемого изображения в выбранном диапазоне яркости, когда выходное изображение использует полный динамический диапазон 0-255. По существу, это преобразование представляет собой линейное контрастирование, применяемое к избранному входному диапазону яркости исходного изображения. Точки, попавшие в интервал $[0, f_{min}]$, после препарирования образуют черный фон, а точки, попавшие в интервал $[f_{max}, 255]$ – белый фон.

Существует еще ряд характеристик, в соответствии с которыми также можно провести препарирование изображения.

Соляризация изображения. Смысл соляризации заключается в том, что участки исходного изображения, имеющие уровень белого или близкий к нему уровень яркости, после обработки приобретают уровень черного. При этом уровень черного сохраняют и те участки, которые имели его на исходном изображении. А уровень белого на выходе приобретают участки, имеющие на входе средний уровень яркости (уровень серого). При данном виде обработки преобразование имеет вид

$$g = k \cdot f \cdot (f_{max} - f) ,$$

где k – константа, позволяющая управлять динамическим диапазоном преобразованного изображения, а f_{max} – максимальное значение исходного сигнала. Функция, описывающая

данное преобразование, является квадратичной параболой, ее график при k=1 приведен на рис. 6.3. При g_{max}=f_{max} динамические диапазоны изображений совпадают, что может быть достигнуто при k = 4/ f_{max}.

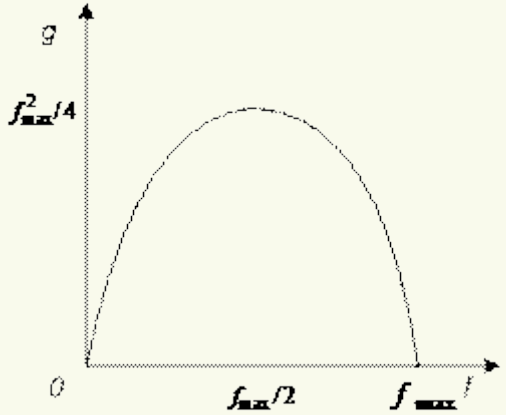


Рис. 6.3

6.2. Точечные фильтры

Преобразование изображения в негатив. Такой переворот уровней яркостей цифровых изображений создает эффект фотографического негатива. Преобразование используется для

усиления белых или серых деталей на фоне темных областей изображения, особенно когда темные области имеют преобладающие размеры, и широко применяется при обработке цифровых медицинских снимков. Цифровой негатив изображения получается путем вычитания значения пикселя из 255 (при условии, что 256 градаций цвета):

$$I' = 255 - I ,$$

где I – интенсивность каждого канала (R, G, B) точки изображения.

То есть для цветного изображения: $R' = 255 - R$; $G' = 255 - G$; $B' = 255 - B$.

Преобразование к оттенкам серого. Данное преобразование заключается в получении яркости каждой точки и последующем копировании полученного значения во все три канала (R = G = B = Y):

$$Y = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B.$$

Изменение яркости и контрастности. Для изменения яркости на N процентов используется следующая формула фильтрации:

$$\mathbf{I = I + N \cdot 128 / 100,}$$

где I – также интенсивность соответственно R, G, B каналов каждой точки изображения.

Уменьшение контрастности на N процентов:

$$\mathbf{I = (I \cdot (100 - N) + 128 \cdot N) / 100.}$$

Увеличение контрастности на N процентов:

$$\mathbf{I = (I \cdot 100 – 128 \cdot N) / (100 – N).}$$

Если новое значение интенсивности пикселя I не попадает в диапазон 0..255, то его урезают до 255.

Изменение цветового баланса. Для изменения цветового баланса по одному из каналов R, G, B на N процентов следует вычислить новое значение цветового канала по формуле:

$$\mathbf{I = I + N \cdot 128 / 100,}$$

где I – интенсивность R, G или B каждой точки изображения.

Если новое значение I не попадает в диапазон 0..255 – то его урезают до 255.

6.3. Матричная фильтрация изображений

Многоточечная обработка изображений

При многоточечной обработки изображений для получения результата в каждом пикселе кадра обрабатываются некоторое множество точек исходного изображения, окружающих обрабатываемый пиксель.

Подходы к улучшению изображений распадается на две категории: методы обработки в пространственной области (пространственные методы) и методы обработки в частотной области (частотные методы).

К пространственной области относится совокупность пикселей, составляющих изображение. Функция предварительной обработки в пространственной области записывается в виде

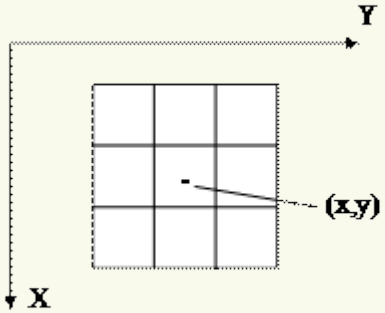
$$g(x,y) = h[f(x,y)],$$

где $f(x, y)$ – входное изображение, $g(x, y)$ – выходное (обработанное) изображение, h – оператор функции f, определенный в некоторой области (x, y).

Операции такого вида относятся к общему классу *операций над соседними элементами*. Эти операции являются основным инструментарием при *низкоуровневой обработке изображений* или *обработке изображений в пространственной области*.

Основным подходом при определении окрестности точки (x, y) является использование квадратной или прямоугольной области части изображения с центром в точке (x, y). Центр этой части изображения перемещается от пикселя к пикселю начиная, например, с левого верхнего угла. При этом для получения $g(x, y)$ оператор применяется для каждого положения (x, y). Хотя используются иногда и другие формы окрестности (например, круг), квадратные формы более предпочтительны из-за простоты их реализации.

Один из наиболее применяемых методов пространственной области основан на использовании фильтров (ядер свертки). Обычно маска фильтра представляет собой небольшую (например, размерность 3*3) двумерную систему, коэффициенты которой выбираются таким образом, чтобы обнаружить заданное свойство изображения (рис. 6.4, а).



w_1 $(x-1, y-1)$	w_2 $(x-1, y)$	w_3 $(x-1, y+1)$
w_4 $(x, y-1)$	w_5 (x, y)	w_6 $(x, y+1)$
w_7 $(x+1, y-1)$	w_8 $(x+1, y)$	w_9 $(x+1, y+1)$

а б

Рис. 6.4: а – маска фильтра; б – коэффициенты маски фильтра

Если величины $w_1, w_2, \ldots, w_9$ представляют собой коэффициенты, маски пикселя (x, y) и его восьми соседей (рис.6.4, б), то алгоритм можно представить как выполнение следующей операции на окрестности 3*3 точки (x, y):

$$h[f(x,y)] = w_1 f(x-1,y-1) + w_2 f(x-1,y) + w_3 f(x-1,y+1) + w_4 f(x,y-1) +$$
$$+ w_5 f(x,y) + w_6 f(x,y+1) + w_7 f(x+1,y-1) + w_8 f(x+1,y) + w_9 f(x+1,y+1).$$

При использовании фильтров могут возникнуть следующие вопросы, связанные с особенностями обработки пикселей:

1. Устранение краевых эффектов. Например, у верхнего левого пикселя не существует «соседей» слева и сверху, следовательно, нам не на что умножать коэффициенты матрицы (рис. 6.5).

Рис. 6.5

2. Вычисленное новое значение пикселя выходит за пределы [0,…,255].

Для первого вопроса возможны следующие пути решения:

1. Исключить из преобразования граничные пиксели изображения. В этом случае выходное изображение будет иметь меньшие размеры (рис. 6.6).

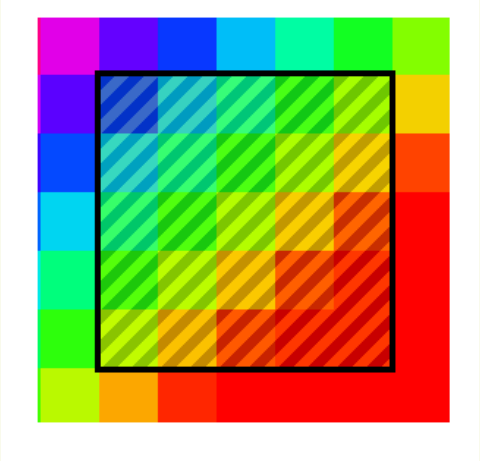


Рис. 6.6

Это не лучший способ, так как фильтр не применяется ко всему изображению. Качество при этом довольно сильно страдает, если размер фильтра велик.

2. Не включать соответствующий пиксель в суммирование, равномерно распределив его вес среди других пикселей окрестности.

3. Дополнить (достроить) исходное изображение, добавив необходимое количество пикселей по границе. Количество достраиваемых строки столбцов, как правило, зависит от размера ядра. Здесь возможны два варианта:

- Доопределить значения пикселей за границами изображения при помощи экстраполяции. Например, считать постоянным значение интенсивности вблизи границы или считать постоянным градиент интенсивности вблизи границы;

- Доопределить значения пикселей за границами изображения при помощи зеркального отражения (рис. 6.7).

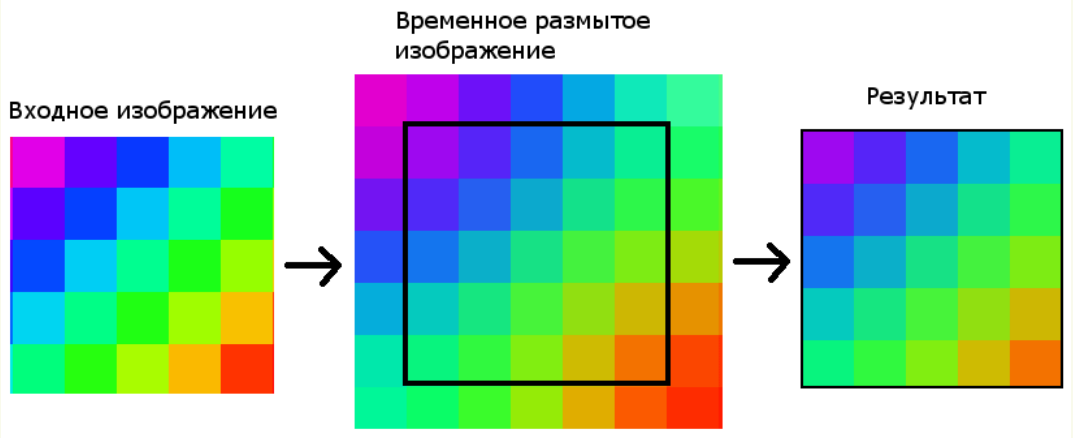


Рис. 6.7

Данный метод не имеет недостатков в качестве, но необходимо производить лишние вычисления.

Для решения проблем, связанных с выходом значения за пределы [0,…,255], возможны следующие действия:

1. Масштабировать полученные значения при положительных откликах фильтра.

2. При отрицательном отклике фильтра брать либо абсолютное значение (по модулю), либо приводить к нулю.

6.4. Матричные фильтры

Низкочастотный фильтр – ослабляет высокочастотные компоненты и усиливает роль низкочастотных. Частота в применении к изображениям отражает количество имеющихся в изображении деталей. Резкие перепады яркости, помехи и шумы являются примером высокочастотных элементов в изображении. Сглаживание изображения реализуется с помощью следующих ядер:

$$H_1 = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad H_2 = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad H_3 = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

Высокочастотный фильтр – ослабляет низкочастотные компоненты в изображении и усиливает роль высокочастотных. Фильтры высокой частоты применяются для выделения таких деталей, как контуры, границы или для повышения резкости изображения. Каждый скачок яркости и каждый контур представляют собой интенсивные детали, связанные с повышенными частотами. Выделение высокочастотных компонент осуществляется с помощью следующих ядер:

$$H_1 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \quad H_2 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}, \quad H_3 = \begin{pmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{pmatrix}.$$

Оператор Робертса. Оператор Робертса является примером нелинейного фильтра. Преобразование каждого пикселя перекрёстным оператором Робертса может показать производную изображения вдоль ненулевой диагонали, и комбинация этих преобразованных изображений может также рассматриваться как градиент от двух верхних пикселов к двум нижним. Оператор Робертса используется ради быстроты вычислений, но проигрывает в сравнении с альтернативами из-за значительной проблемы чувствительности к шуму. Он даёт линии тоньше, чем другие методы выделения границ.

В обработке участвуют четыре пикселя, расположенные следующим образом (рис. 6.8).

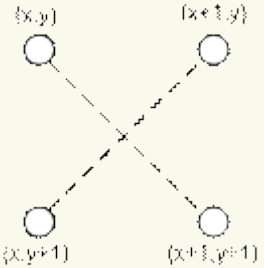


Рис. 6.8. Пиксели, участвующие в обработке оператором Робертса

Отклик оператора Робертса:

$$S(x,y) = \sqrt{[f(x+1,y) - f(x,y+1)]^2 + [f(x,y) - f(x+1,y+1)]^2}.$$

Ядра свертки в данном случае будут выглядеть таким образом:

$$H_1 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \quad H_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Свертка для каждого ядра вычисляется отдельно. В качестве отклика данного фильтра выступает величина

$$S = \sqrt{P^2 + Q^2}\,,$$

где P и Q – отклик ядер H_1 и H_2.

Иногда в качестве оператора Робертса берется величина S = |P| + |Q|.

Оператор Собеля. Оператор Собеля применяют в алгоритмах выделения границ. Это дискретный дифференциальный оператор, вычисляющий приближенное значение градиента яркости изображения. Результатом применения оператора Собеля в каждой точке изображения является либо вектор градиента яркости в этой точке, либо его норма. Метод усиления края с помощью оператора Собеля рассматривает два различных ядра свертки:

$$\boldsymbol{H_1} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{2} & \mathbf{0} & \mathbf{-2} \\ \mathbf{1} & \mathbf{0} & \mathbf{-1} \end{pmatrix}, \quad \boldsymbol{H_2} = \begin{pmatrix} \mathbf{-1} & \mathbf{-2} & \mathbf{-1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{2} & \mathbf{1} \end{pmatrix}.$$

Исходя из этих сверток вычисляется величина и направление краев. Свертка для каждого ядра вычисляется отдельно. В качестве отклика данного фильтра выступает величина

$$S = \sqrt{P^2 + Q^2}\,,$$

где P и Q – отклик ядер H_1 и H_2.

Иногда в качестве оператора Собеля берется величина S = |P| + |Q|.

Оператор Превитта. Аналогично оператору Собеля действует оператор Превитта. Детектор границ Превитта является подходящим способом для оценки величины и ориентации границы. В то время как детектор с дифференциальным градиентом нуждается в трудоёмком вычислении оценки ориентации по величинам в вертикальном и горизонтальном направлениях, детектор границ Превитта даёт направление прямо из ядра с максимальным результатом. Метод усиления края с помощью оператора Превитта рассматривает два различных ядра свертки:

$$\boldsymbol{H_1} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{1} & \mathbf{0} & \mathbf{-1} \end{pmatrix}, \quad \boldsymbol{H_2} = \begin{pmatrix} \mathbf{-1} & \mathbf{-1} & \mathbf{-1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}.$$

Результат работы оператора Превитта есть

$$\max\{P,Q\},$$

где P и Q – отклик ядер H_1 и H_2.

Оператор Лапласа. Дискретный оператор Лапласа часто используется в обработке изображений, например в задаче выделения границ или в приложениях оценки движения. Дискретный лапласиан определяется как сумма вторых производных и вычисляется как сумма перепадов на соседях центрального пикселя. Метод усиления края по Лапласу рассматривает целый ряд различных ядер свертки. Приведем некоторые их них:

$$H_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad H_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad H_3 = \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}.$$

Как видно, сумма элементов матриц равна нулю, поэтому отклик фильтра может быть отрицательным. В этом случае значение отклика берется по модулю. В результате обработки области с постоянной или линейно возрастающей интенсивностью становятся черными, а области быстро изменяющихся значений интенсивности ярко высвечиваются.

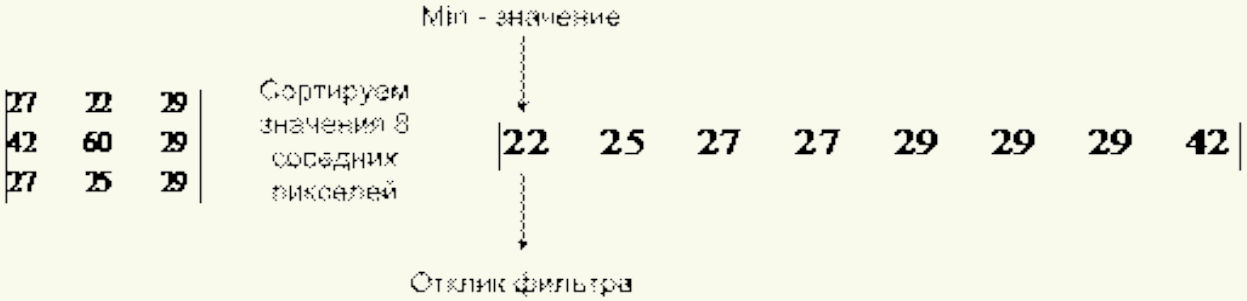
Ниже приведем некоторые пространственные процессы, которые не подпадают под категорию свертки и могут применяться для устранения различного вида шума.

Фильтр «гармоническое среднее». Гармоническое среднее ряда $f(n)$ вычисляется по формуле

$$G(f) = \frac{n}{\sum_{i=1}^{n} \frac{1}{f_i}}.$$

В процессе фильтрации значение текущего пикселя изображения заменяется на $f(n)$ множества значений девяти пикселей, включая текущий и соседние.

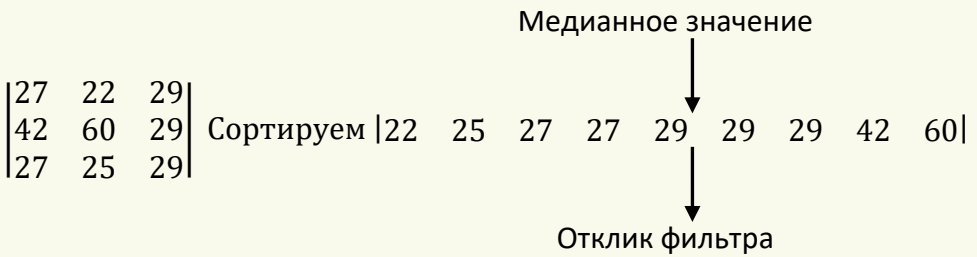
Min – фильтр. В процессе фильтрации значение текущего пикселя заменяется на минимальное значение соседних пикселей. Так, например, для ядра размерности 3 будем иметь:



Max – фильтр. В процессе фильтрации значение текущего пикселя заменяется на максимальное значение соседних пикселей (по аналогии с предыдущим фильтром).

Min-Max–фильтр. В процессе фильтрации значение текущего пикселя изображения сначала заменяется на минимальное значение соседних пикселей, а при повторном проходе на максимальное.

Медианный фильтр. Усредненное фильтрование использует значения элементов, содержащихся в области примыкания, для определения нового значения. Фильтр располагает элементы области примыкания в отсортированном порядке и отбирает среднее значение. Так, например, для ядра размерности 3 медианное значение будет пятым:



С помощью методов пространственной обработки изображений можно получить ряд интересных эффектов. Приведем некоторые из них.

6.5. Алгоритмы фильтрации

Существует несколько различных вариантов обработки растровых изображений: изменение цветовых компонент для получения определенного баланса, увеличение резкости изображения, размывание контрастных изображений для имитации эффекта смягчающих фотофильтров и др.

Под задачей фильтрации изображений в широком смысле понимают любые процедуры обработки изображений, при которых на вход процедуры подается растровое изображение и на выходе формируется растровое изображение. Однако чаще под «фильтрацией» понимают так называемую *помеховую фильтрацию*.

Главная цель помеховой фильтрации заключается в такой обработке изображений, при которой результат оказывается более подходящим с точки зрения конкретного применения. В общем случае можно выделить *линейные фильтры* (сглаживающие фильтры, контрастоповышающие фильтры, разностные фильтры) и *нелинейные фильтры* (медианный фильтр).

Рассмотрим несколько широко используемых в компьютерной графике эффектов, которые дает обработка растровых изображений.

Каждый из них достигается применением ядра свертки - матрицы чисел (обычно размером 3x3). Чтобы преобразовать один пиксель в изображении, значение его цвета умножается на число в центре ядра. Затем восемь значений цветов пикселей, окружающих центральный пиксель, умножаются на соответствующие им коэффициенты ядра, все девять значений суммируются, и в результате получается новое значение цвета центрального пикселя. Этот процесс повторяется для каждого пикселя в изображении; тем самым изображение фильтруется (рис. 6.9).

Коэффициенты ядра определяют результат процесса фильтрации. Если сумма коэффициентов больше чем 1, яркость увеличится; если меньше чем 1, яркость уменьшится. Изменяя определенным способом размер и состав ядра свертки, можно получить различные полезные спецэффекты.

Ядро свертки может быть размером 5x5, 7x7 и т.д. Нечетные размеры окна необходимы для однозначного определения центрального элемента.

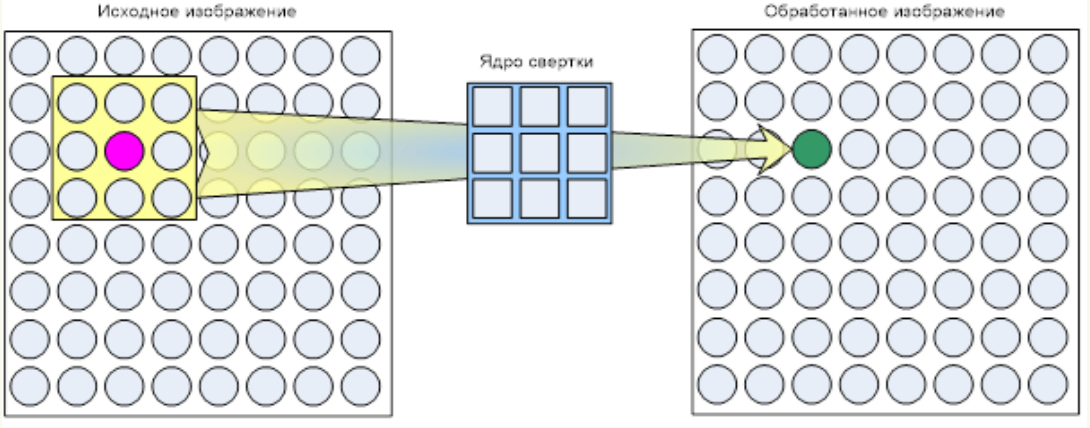


Рис. 6.9

В каждом положении окна происходит операция свертки – линейная комбинация значений элементов изображения:

$$p'_5 = \sum_{i=1}^{9} p_i k_i$$

где матрица пикселей

$$P = \begin{vmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{vmatrix},$$

ядро свёртки

$$K = \begin{vmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{vmatrix},$$

p′5 – новое значение центрального пикселя.

Алгоритм размывания

При использовании алгоритма размывания в изображении перераспределяются цвета и смягчаются резкие границы. Ядро размывания состоит из совокупности коэффициентов, каждый из которых меньше 1, а их сумма составляет 1. То есть применяется ***низкочастотный фильтр***. Это означает, что после фильтрации каждый пиксель поглотит что-то из цветов соседей, но полная яркость изображения останется неизменной. При подготовке к размыванию цифровое изображение считывается в память компьютера в виде красной, зеленой и синей компонент цвета каждого пикселя. Ядро размывания R размером 3x3 применяется к красной, зеленой и синей компонентам цвета каждого пикселя в изображении:

$$R = \begin{vmatrix} 0{,}05 & 0{,}05 & 0{,}05 \\ 0{,}05 & 0{,}60 & 0{,}05 \\ 0{,}05 & 0{,}05 & 0{,}05 \end{vmatrix}.$$

Значение цвета пикселя, который находится под центром ядра, вычисляется умножением весовых коэффициентов ядра на соответствующие значения цвета в изображении и суммированием результатов.

Итоговое изображение получается размытым по сравнению с оригиналом потому, что цвет каждого пикселя распространился среди соседей. Степень размывания можно увеличить тремя способами: используя ядро большего размера, чтобы распределить цвета среди большего числа соседей; подбирая коэффициенты ядра с целью уменьшения влияния центрального коэффициента; фильтруя изображение многократно с ядром размывания.

Алгоритм увеличения резкости

При использовании алгоритма увеличения резкости подчеркиваются различия между цветами смежных пикселей и выделяются незаметные детали. Увеличение резкости достигается точно так же, как и размывание, за исключением того, что используется другое ядро, так как цель преобразования - увеличить, а не уменьшить четкость изображения. В ядре резкости центральный коэффициент больше 1, а окружен он отрицательными числами, сумма которых на единицу меньше центрального коэффициента. То есть применяется ***высокочастотный фильтр***. Таким образом, увеличивается любой существующий контраст между цветом пикселя и цветами его соседей. При обработке каждого пикселя в изображении используется ядро резкости G размером 3x3:

$$G = \begin{vmatrix} -0{,}1 & -0{,}1 & -0{,}1 \\ -0{,}1 & 1{,}8 & -0{,}1 \end{vmatrix}.$$

$$\begin{vmatrix} -0{,}1 & -0{,}1 & -0{,}1 \end{vmatrix}$$

Как и прежде, красная, зеленая и синяя цветовые составляющие обрабатываются отдельно и позже объединяются, чтобы сформировать 24-битное значение цвета.

Конечное изображение получается более четким, чем оригинал. Дополнительные детали не возникли из ничего; процесс увеличения резкости просто повысил существующий контраст между пикселями. При повторной обработке изображения четкость может увеличиться еще больше.

Алгоритм тиснения

Алгоритм тиснения преобразует изображение так, что объекты сцены выглядят выдавленными на металлической поверхности, подобно чеканке на монетах. Тиснение выполняется почти так же, как размывание и увеличение резкости. Каждый пиксель обычного цветного изображения обрабатывается ядром тиснения Т размером 3x3. Несколько возможных вариантов ядра тиснения:

$$T_1 = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix};\ T_2 = \begin{vmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{vmatrix};\ T_3 = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix}.$$

В отличие от ядер размывания и резкости, в которых сумма коэффициентов равна 1, сумма весов в ядре тиснения равна 0 – ***оператор Собеля, Превитта***. Это означает, что фоновым пикселям (тем, которые не находятся на границах перехода от одного цвета к другому) присваиваются нулевые значения, а не фоновым - значения, отличные от нуля. То

есть если все девять пикселей, находящихся в области матрицы тиснения, имеют одинаковые визуальные свойства, то значение центрального пикселя после преобразования станет нулевым (черный цвет).

После того как значение пикселя обработано ядром тиснения, к нему прибавляется 128. Таким образом, значением фоновых пикселей станет средний серый цвет (красный = 128, зеленый = 128, синий = 128). Суммы, превышающие 255, можно округлить до 255 или взять остаток по модулю 255, чтобы значение оказалось между 0 и 255.

В тиснёном варианте изображения контуры кажутся выдавленными над поверхностью. Направление подсветки изображения можно изменять, меняя позиции 1 и -1 в ядре. Если, например, поменять местами значения 1 и -1, то реверсируется направление подсветки:

$$T_4=\begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix};\ T_5=\begin{vmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{vmatrix};$$

$$T_6=\begin{vmatrix} 2 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 2 \end{vmatrix};\ T_6=\begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \end{vmatrix}.$$

Алгоритм акварелизации

В алгоритме акварелизации акварельный фильтр преобразует исходное изображение так, что после обработки оно выглядит выполненным акварелью.

Первый шаг в применении акварельного фильтра - сглаживание цветов в изображении. Один из способов сглаживания - процесс медианного осреднения цвета в каждой точке. Значения цвета каждого пикселя и его 8 соседей (или 24 соседей для матрицы 5x5) помещаются в список и сортируются от меньшего к большему. Медианное (тринадцатое) значение цвета в списке присваивается центральному пикселю. Пример медианной фильтрации для матрицы 3x3 приведен на рис. 6.10.

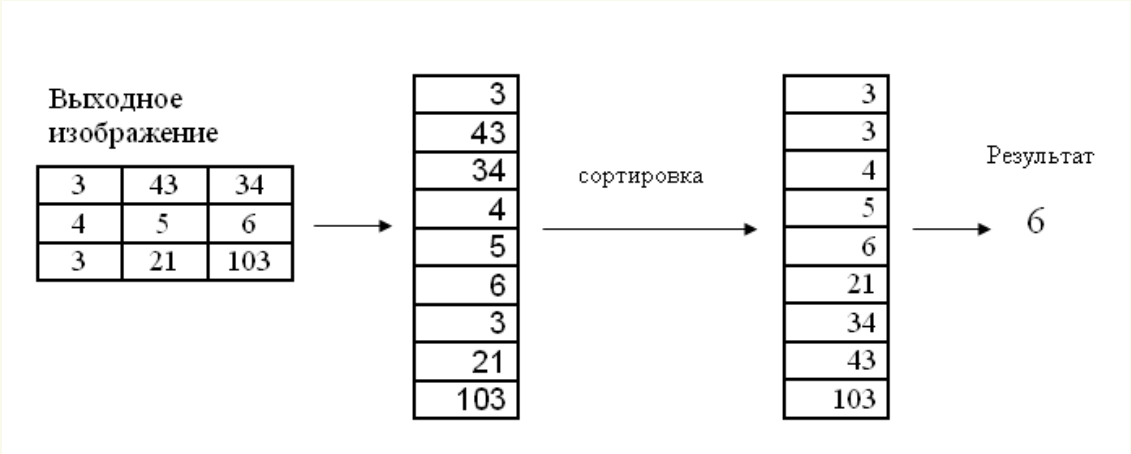


Рис. 6.10

Возможно также применение медианного низкочастотного фильтра M, выполняющего ту же операцию:

$$M = \frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}.$$

Второй шаг после сглаживания цветов - обработка каждого пикселя в изображении ядром резкости G для выделения границы переходов цветов:

$$G = \begin{vmatrix} -0,5 & -0,5 & -0,5 \\ -0,5 & 5,0 & -0,5 \\ -0,5 & -0,5 & -0,5 \end{vmatrix}.$$

Результирующее изображение напоминает акварельную живопись. Этот пример показывает также, что можно объединять различные методы обработки изображений и добиваться новых визуальных эффектов.

Алгоритм получения полутонового изображения

Бинарное «псевдополутоновое» изображение получают обработкой исходного изображения при помощи ядра D_2 или D_4: если значение пикселя меньше пропорционального значения соответствующего ему элемента ядра, то он обнуляется, иначе ему присваивается 255. Ядро накладывается на изображение без перекрытия. Ядра D_2 и D_4:

$$D_2 = \begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix}; \qquad D_4 = \begin{pmatrix} 0 & 4 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}.$$

Алгоритмы устранения шумов

Отдельной задачей является устранения шума на изображении. Вариант классификации такого шума:

1. Шум «соль и перец» – случайные белые и черные пиксели.
2. Импульсный шум – случайные белые пиксели.
3. Гауссов шум – колебания интенсивности, распределенные по нормальному закону.

Для решения этой задачи применяются те же алгоритмы фильтрации со своими ядрами свертки.

Рассмотрим некоторые фильтры, сглаживающие шум. Пусть ядро размером 3x3 имеет вид:

$$H_1 = \frac{1}{1}\begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

Тогда яркость (i,j) -го пикселя после фильтрации определится как

$$g_{ij} = 1/10\left(f_{(i-1),(j-1)} + f_{(i-1),j} + f_{i,(j-1)} + f_{i,(j-1)} + 2f_{ij} + f_{ij+1} + f_{i+1,(j-1)} + f_{i+1,j} + f_{i+1,j+1}\right).$$

Хотя коэффициенты a_{kl} можно выбрать из среднеквадратического или иного условия близости не искаженного шумом $s_{i,j}$ и преобразованного $g_{i,j}$ изображений, обычно их задают эвристически. Приведем еще некоторые матрицы шумоподавляющих фильтров:

$$H_2 = \frac{1}{2}\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}, \quad H_3 = \frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}, \quad H_4 = \frac{1}{14}\begin{vmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{vmatrix}.$$

У фильтров $H_1 - H_4$ нормирующие множители K подобраны таким образом, чтобы не происходило изменения средней яркости обработанного изображения. Наряду с масками 3x3 используются маски большей размерности, например, 5x5, 7x7 и т.п. В отличие от фильтра H_2, у фильтров H_1, H_3, H_4 весовые коэффициенты на пересечении главных диагоналей матрицы больше, чем коэффициенты, стоящие на периферии. Фильтры H_1, H_3, H_4 дают более плавное изменение яркости по изображению, чем H_2.

Пусть отсчеты полезного изображения $f_{k,m}$ мало меняются в пределах ядра. На изображение накладывается аддитивный шум: $f_{k,m} + n_{k,m}$, отсчеты шума $n_{k,m}$ случайны и независимы (или слабо зависимы) со статистической точки зрения. В этом случае механизм подавления шума с использованием приведенных фильтров состоит в том, при суммировании

шумы компенсируют друг друга. Эта компенсация будет происходить тем успешнее, чем большее число членов в сумме, т.е. чем больше размер ядра. Пусть, например, используется ядро $N \times N$, в его пределах полезное изображение имеет постоянную яркость f, шум с независимыми значениями отсчетов $n_{k,m}$, средним значением $\mu = 0$ и дисперсией σ^2 в пределах маски (такой шум называют *белым*). Отношение квадрата яркости (i,j)-го пикселя к дисперсии шума, т.е. отношение сигнал/шум, равно f^2/σ^2.

Рассмотрим, например, маску типа H_2:

$$g_{ij} = 1/N^2 \sum_{k=1}^{N} \sum_{m=1}^{N} (f + n_{k,m}) = f + 1/N^2 \sum_{k=1}^{N} \sum_{m=1}^{N} n_{k,m}.$$

Средний квадрат яркости равен f^2, средний квадрат интенсивности шума:

$$\left\langle n_{i,j}^2 \right\rangle = 1/N^4 \sum_{k=1}^{N} \sum_{m=1}^{N} \left\langle n_{k,m}^2 \right\rangle = f + 1/N^4 \sum_{k=1}^{N} \sum_{m=1}^{N} \sum_{q=1}^{N} \sum_{p=1}^{N} \left\langle n_{k,m} n_{q,p} \right\rangle.$$

Двойная сумма отвечает $k=p$, $m=q$, эта сумма равна σ^2/N^2. Четырехкратная сумма равна нулю, так как отсчеты шума при $k \neq p$, $m \neq q$ независимы: $\langle n_{k,m} n_{p,q} \rangle = 0$. В результате фильтрации

отношение сигнал/шум становится равным N^2f^2/σ^2, т.е. возрастает пропорционально площади ядра. Отношение яркости *(i,j)*-го пикселя полезного изображения к среднеквадратическому отклонению шума возрастает пропорционально *N*. Применение маски 3x3 в среднем повышает отношение сигнала к шуму в 9 раз.

При импульсной помехе механизм подавления состоит в том, что импульс "расплывается" и становится мало заметным на общем фоне. Статистические свойства импульсных помех резко отличаются от статистических свойств изображений. Для изображений характерны плавные, небольшие изменения от элемента к элементу, а быстрые, скачкообразные изменения редки и образуют протяженные контуры. Импульсные же помехи представляют собой значительные по величине одиночные изолированные выбросы. Этим объясняется тот факт, что визуально очень легко отличить выбросы помехи от изображения, хотя такие помехи и оказывают сильное мешающее действие. Благодаря такому резкому отличию помех от изображений алгоритм фильтрации импульсных помех оказывается весьма простым.

Однако часто в пределах апертуры значения полезного изображения все же изменяются заметным образом. Это бывает, в частности, когда в пределы маски попадают контуры. С физической точки зрения, все H_1 - H_4 являются фильтрами нижних частот (усредняющими фильтрами), подавляющими высокочастотные гармоники и шума, и полезного изображения. Это приводит не только к ослаблению шума, но и к размыванию контуров на изображении.