1 Основные понятия компьютерной графики

1.1 Способы задания геометрических объектов

Бесспорно, геометрическое моделирование является одним из наиболее важных областей человеческой деятельности, поскольку оно часто используется при проектировании. Например, при проектировании архитектурных строений, механических узлов, нового дизайна изделий бытового назначения и других предметов. Основой точного геометрического моделирования является прикладная геометрия. Однако в силу возрастающей мощности компьютерных систем эта область пополнилась новыми технологиями, такими как твердотельное моделирование, B-сплайны, лофтинг, скиннинг и др. Они успешно применяются на практике. Однако, несмотря на большие успехи в этом направлении, вопросы синтеза моделей геометрических объектов все еще остаются актуальными. Основные из них это:

- повышение быстродействия выполнения операций с геометрическими моделями;
- разработка технологий, упрощающих построение сложных геометрических объектов;
- интеграция разнородных геометрических моделей.

Рассмотрим распространенную классификацию существующих способов представления трехмерных геометрических объектов (моделей):

- Простейшие
- Поверхностные
- Объемные

1.1.1 Простейшие способы

К простейшим способам задания трехмерных объектов относятся точечное и проволочное (каркасное) представления.
В точечном представлении объект задан совокупностью вершин, принадлежащих поверхности объекта V = {V1,…,Vn}.
Проволочная модель является расширением предыдущего способа. Объекты задаются совокупностью вершин и соединяющих их ребер (отрезков прямой или кривой соединяющей вершины vi, vj.):
V = {v1,…,vn}, E = {vi, vj },
Основное преимущество этих способов - простота представления. Потому они применяются на промежуточных стадиях работы с геометрическим объектом: предварительная визуализация и как исходные модели, для синтеза более сложных моделей.

1.1.2 Поверхностное представление (Boundary representation, B-rep)

В поверхностном представлении объект создается при помощи набора тонких поверхностей, составляющих его границу. Как правило, поверхностное представление используется в тех областях, где нет необходимости обрабатывать каким-либо образом внутренность тела: дизайнерские проекты, моделирование обводов какого-либо изделия, создание объектов с нестандартными элементами (например, скругление с изменяемым радиусом, винтообразная улитка) и т. д.
Существует несколько основных способов задания границы тела.

Неявная функция
Задание 3D объекта при помощи неявной функции состоит в том, что мы указываем некую функцию $F(x,y,z)$, нулями которой будут точки (x_i, y_i, z_i), образующие нашу поверхность, либо дающие ее приближение. Так, например, неявная функция для сферы радиуса r (см. Рис. 1.1), с центром в точке (x_0, y_0, z_0) имеет вид:

$$F(x, y, z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2$$

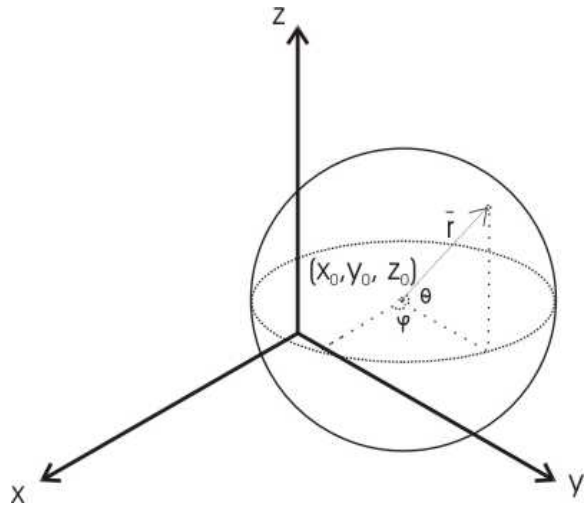

Рис. 1.1. Сфера, заданная неявной функцией

Параметрическое задание, сплайны.

При параметрическом задании координаты точек поверхности рассматриваются как некие функции от двух параметров, пробегающих некоторый набор значений.

$$\begin{cases} x = x(u,v) \\ y = y(u,v) \\ z = z(u.v) \end{cases}$$

где, для параметров u и v, как правило, определяют область определения прямоугольного ($ua < u < ub$, $va < v < vb$) либо треугольного ($ua < u < ub$, $va < u + v < vb$) вида.

Так, например, параметризация той же сферы (см. Рис. 1.7) относительно двух углов φ (долгота) и θ (широта) будет иметь вид:

$$\begin{cases} x = r\cos\theta\cos\varphi \\ y = r\cos\theta\sin\varphi, \quad 0 \le \varphi < 2\pi, \ -\dfrac{\pi}{2} \le \theta \le \dfrac{\pi}{2} \\ z = r\sin\theta \end{cases}$$

Помимо этого используются различного рода особые параметрические поверхности, например, поверхности Безье, B-сплайны и др.

Параметрические поверхности очень широко используются в различных CAD - системах (в России используется термин САПР - системы автоматического проектирования). Так, при помощи них моделируются и рассчитываются обводы автомобилей, формы деталей и т.п.

Полигональные поверхности

При использовании данного метода поверхность представляется в виде набора некоторых многоугольников в пространстве (рис. 1.2).

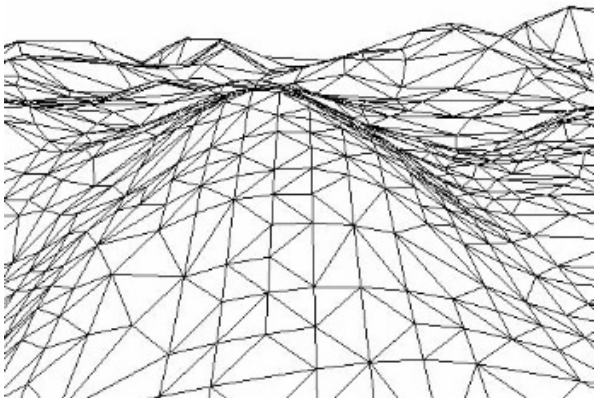

Рис. 1.2. Триангуляция поверхности.

Часто в качестве многоугольников используются треугольники, в этом случае само разбиение поверхности (приближенное представление треугольниками) называется триангуляцией поверхности. Топология полученной при этом сетки описывается следующим образом:

1. Объектами сетки являются вершины, ребра и треугольники (задаваемые тремя вершинами или тремя ребрами), а в более общем случае – многоугольники.
2. У любой вершины есть свойство валентности (т.е. число многоугольников, содержащих ее).
3. Для любого треугольника существует не более одного другого треугольника, инцидентного для первого по фиксированному ребру. Т.е., в частности, не может быть ситуации, приведенной на рисунке 1.3.

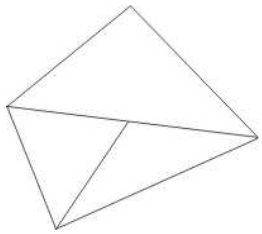

Рис. 1.3 Нарушение условия об инцидентности.

При таком способе задания мы можем, например, организовать обход в некотором направлении по треугольникам, содержащим фиксированную вершину, просто переходя каждый раз к треугольнику, инцидентному по ребру для предыдущего.
Разумеется, при триангуляции качество получаемой поверхности тем лучше, чем больше треугольников мы будем для этого использовать

Полигональное задание трехмерных объектов является наиболее распространенным и для него сформировались следующие разновидности топологий:

1. список вершин. В этой топологии грань выражается через вершины:
   V = {vi } – вершины |V| = n;
   F = {(vj1, vj2,…, vjk [, fj(u,v)]) } – грань (или параметрическая поверхность fj), состоящая из k вершин (k >= 3)
2. список ребер. Здесь грань выражена через ребра:
   V = {vi } – вершины |V| = n;
   E = { ek = (vi, vj [, fk(u)]) } – ребро (или параметрическое представление линии fk);
   F = { (ej1, ej2,…, ejk [, fj(u,v)]) } – грань (или параметрическая поверхность fj) состоящая из k ребер (k >= 3).
3. "крылатое" представление. Эта модель является развитием модели, основанной на информации о ребрах. Отличие состоит в том, что в структуру, описывающую ребра, добавляется информация о взаимном расположении граней

"Крылатое" представление является наиболее удобным для реализации важнейших алгоритмов над геометрическими объектами:
- проверка правильности задания;
- алгоритмы для полигональных моделей, связанные с обходом ребер (выделение плоских контуров, упрощение модели путем удаления граней и другие);
- возможность, не более, чем за линейное время, восстановить любую другую топологию, следуя по цепочкам связей между элементами.

1.1.3 Объемное представление

Методы объемного представления тел сохраняют информацию о любых, не обязательно видимых, частях тела. Все они, как правило, требуют гораздо больше места для хранения объекта, но как уже было сказано, дают дополнительную информацию об объектах,

которая может нам потребоваться. Также эти методы необходимы, когда тела не имеют как таковой границы (например, туман, взвесь в жидкости и т.д.)

Конструктивная геометрия тел (Constructive Solid Geometry, CSG)

Метод конструктивной геометрии тел заключается в том, что мы получаем нужную нам поверхность как результат применения последовательности различных множественных операций (объединения, пересечения, разности и т.д.) к некоторым примитивам (т.е. минимальным объектами для построения). В качестве примитивов могу выступать параллелепипеды, шары, конусы, пирамиды и т.д. Сама поверхность задается при помощи дерева построения (см. рис 1.4).

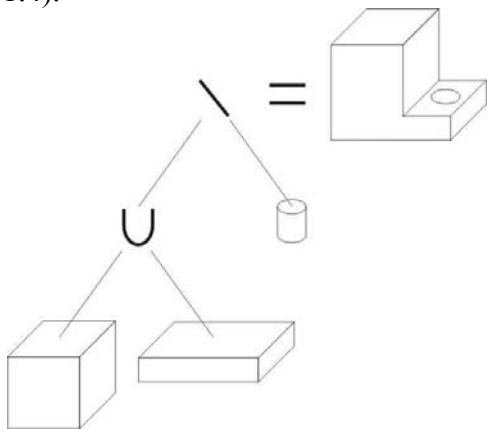

Рис. 1.4. Дерево построения.

3D растр (Воксельное представление)

3D растр представляет собой набор кубиков (см. рис. 1.5) (voxel'ов, от «volume element») в пространстве. Каждый воксел может иметь некоторое числовое значение, являющееся атрибутом соответствующей точки в пространстве.

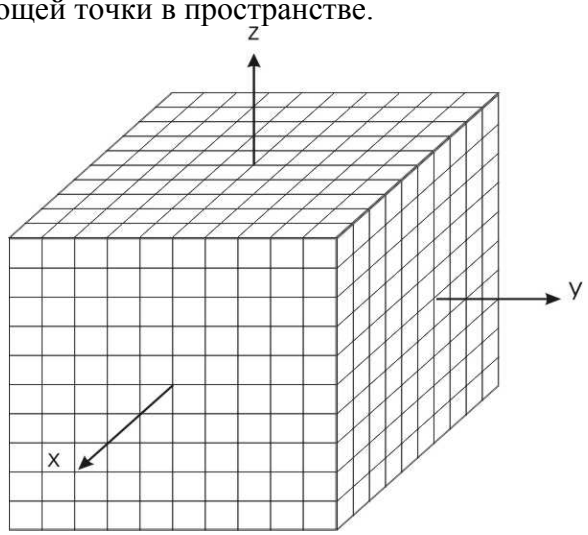

Рис. 1.5. 3D растр.

Данный метод, несмотря на свою простоту, имеет серьезный недостаток: для хранения даже небольшого объекта в приемлемом качестве требуется очень много места. Например, если мы храним кубик с ребром 1024 и выделяем по одному байту (8 бит) на атрибут воксела, то нам потребуется $1024 \times 1024 \times 1024 \times 1 байт = 1 Гб$. Учитывая тот факт, что хранить объект нам, скорее всего, надо будет в оперативной памяти компьютера, применение этого метода в чистом виде становится крайне затруднительным.

Воксельное представление является очень удобным для реализации пространственных алгоритмов и теоретико-множественных операций над объектами (объединение,

вычитание, пересечение), но обладает рядом недостатков, которые ограничивают область его применения:
- низкая точность для представления для большинства объектов;
- большой объем занимаемой памяти;

Восьмеричное дерево

В большинстве случаев в 3D растре есть некоторые области вокселов, имеющих одинаковые атрибуты. Этим можно воспользоваться для того, чтобы сократить объем занимаемого объектом места. Будем хранить растр в виде дерева (см. 1.6), вершины которого имеют степень 0 или 8. Сначала проверим, не лежат ли во всех вокселах в растре одинаковые значения, если да, то нам достаточно будет сохранить это значение (*p)* и обозначить, что дальше дробить кубик не надо, например, положим в вершину это значение *p* и 0 (обозначение терминальности вершины). Если у нас есть вокселы с разными значениями, то продолжим процесс. А именно, разделим растр на октанты, соответствующие координатным плоскостям. Для каждого октанта снова проверим, не содержит ли он одинаковые вокселы. И т.д. Получим дерево, в вершинах которого лежат 1 (что обозначает, что соответствующий октант не однороден) или 0 и некоторые числа p_i, соответствующие значениям всех вокселов в данном октанте.

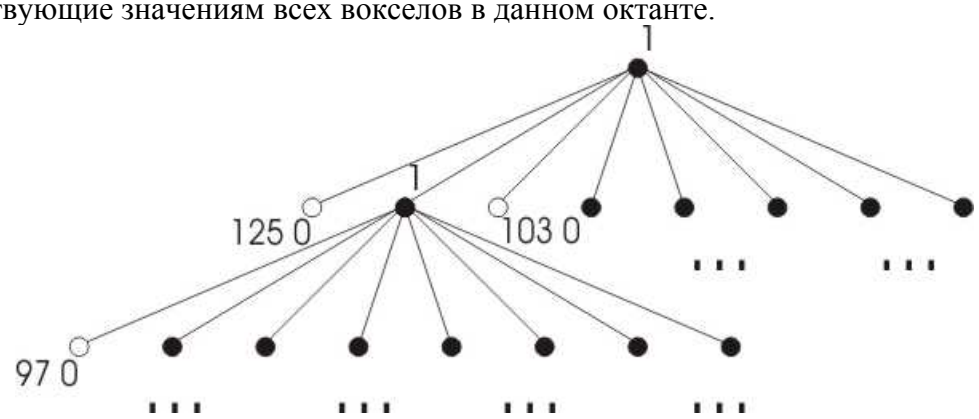

Рис. 1.6 Восьмеричное дерево 3D растра.

Таким образом, если растр состоит из вокселов, имеющих один и тот же атрибут, то мы сэкономим $1\,Гб - (1байт + 1бит)$; если же все воксели различны, то лишнее место, которое мы потратим (на 1 и 0), составит $(1 + 8 + 8^2 + ... + 8^{10})\ битов = 1227133513\ битов \approx 146\,мб$. Но последняя ситуация встречается крайне редко, поэтому использование восьмеричных деревьев в большинстве случаев позволяет значительно сократить объем памяти для хранения объекта.

Двоичное дерево

Идея построения двоичного дерева полностью аналогична построению восьмеричного дерева. Но в данном случае растр последовательно делится не на октанты, а на половинки: сначала параллельно плоскости OXY, потом OYZ и т.д. Соответственно, аналогично строится дерево, вершины которого имеют степень 0 или 2.

1.2 Растровые представления изображений.

1.2.1 Виды растра

Цифровое изображение – набор точек (пикселей) изображения; каждая точка изображения характеризуется координатами x и y и яркостью V(x,y), это дискретные величины, обычно целые. В случае цветного изображения, каждый пиксель характеризуется координатами x и y, и тремя яркостями: яркостью красного, яркостью синего и яркостью зеленого (V_R , V_B , V_G). Комбинируя данные три цвета можно получить большое количество различных оттенков.

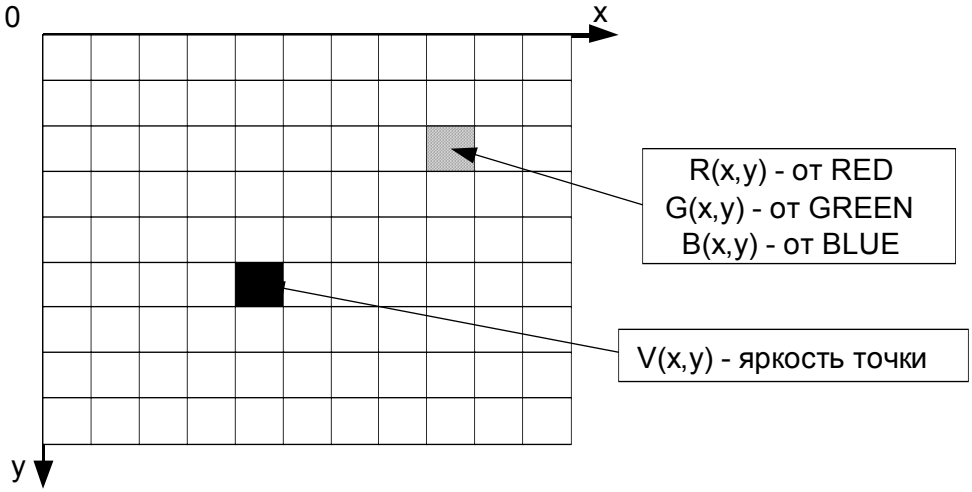

Рис. 1.7 Квадратный растр

Растр – это порядок расположения точек (растровых элементов). На рис. 1.7 изображен растр элементами которого являются квадраты, такой растр называется ***квадратным***, именно такие растры наиболее часто используются. Хотя возможно использование в качестве растрового элемента фигуры другой формы, соответствующего следующим требованиям:

1. Все фигуры должны быть одинаковые;
2. Должны полностью покрывать плоскость без наезжания и дырок.

Так в качестве растрового элемента возможно использование равностороннего треугольника, правильного шестиугольника (гексаэдра) рис. 1.8. Можно строить растры, используя неправильные многоугольники, но практический смысл в подобных растрах отсутствует.

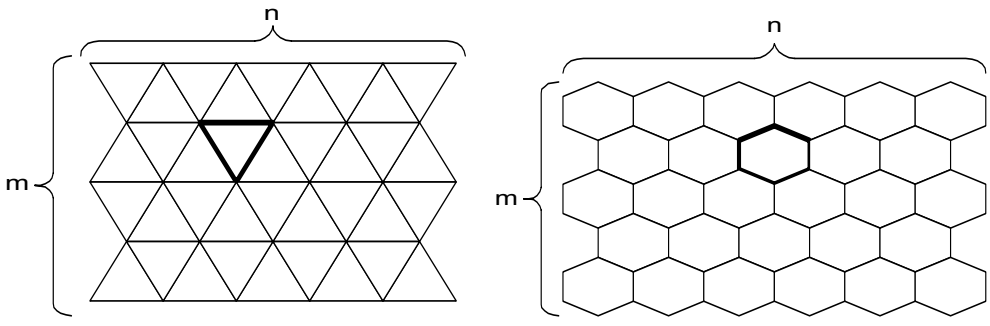

Рис. 1.8 Треугольный и гексагональный растр

Рассмотрим способы построения линий в прямоугольном и гексагональном растре. В квадратном растре построение линии осуществляется двумя способами:

1) Результат – восьмисвязная линия. Соседние пиксели линии могут находится в одном из восьми возможных (см. рис. 1.9а) положениях. Недостаток – слишком тонкая линия при угле 45°.
2) Результат – четырехсвязная линия. Соседние пиксели линии могут находится в одном из четырех возможных (см. рис. 1.9б) положениях. Недостаток – избыточно толстая линия при угле 45°.

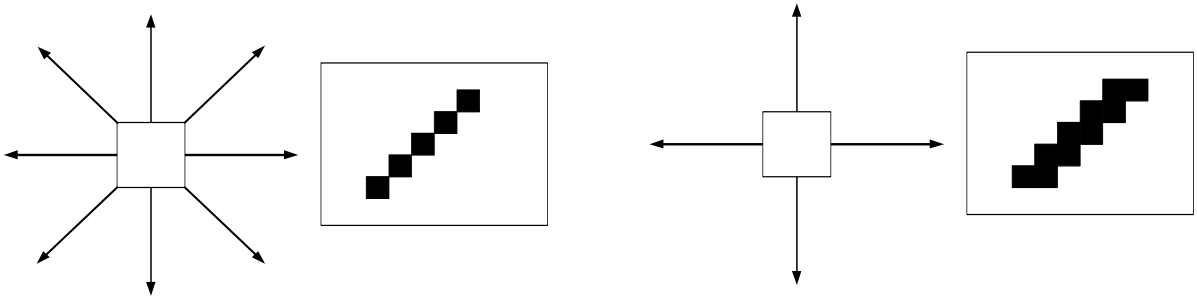

Рис. 1.9. Построение линии в прямоугольном растре

В гексагональном растре линии шестисвязные (см. рис. 1.10) такие линии более стабильны по ширине, т.е. дисперсия ширины линии меньше, чем в квадратном растре.

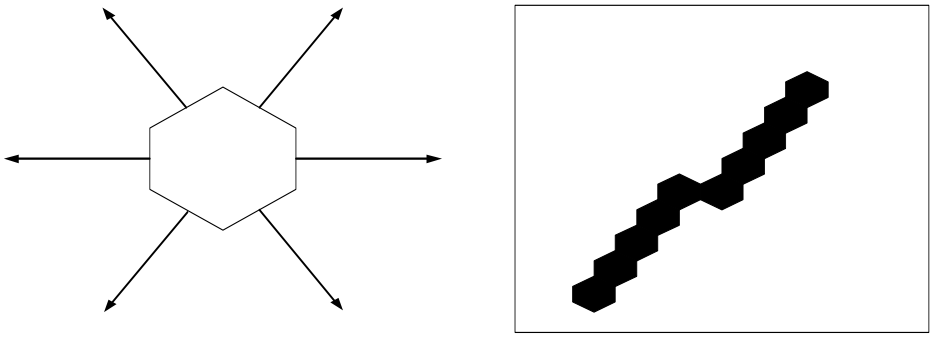

Рис. 1.10. Построение линии в гексагональном растре

Каким образом можно оценить, какой растр лучше?

Одним из способов оценки является передача по каналу связи кодированного, с учетом используемого растра, изображения с последующим восстановлением и визуальным анализом достигнутого качества. Экспериментально и математически доказано, что гексагональный растр лучше, т.к. обеспечивает наименьшее отклонение от оригинала. Но разница не велика.

1.2.2 Построение линии в квадратном растре.

Поскольку экран растрового дисплея можно рассматривать как матрицу дискретных элементов (пикселей), каждый из которых может быть подсвечен, нельзя непосредственно провести отрезок из одной точки в другую. Процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок, называется разложением в растр. В сочетании с процессом построчной визуализации изображения он известен как преобразование растровой развертки. Для горизонтальных, вертикальных и наклоненных под углом 45° отрезков выбор растровых элементов очевиден. При любой другой ориентации выбрать нужные пиксели труднее, что показано на рис. 1.11

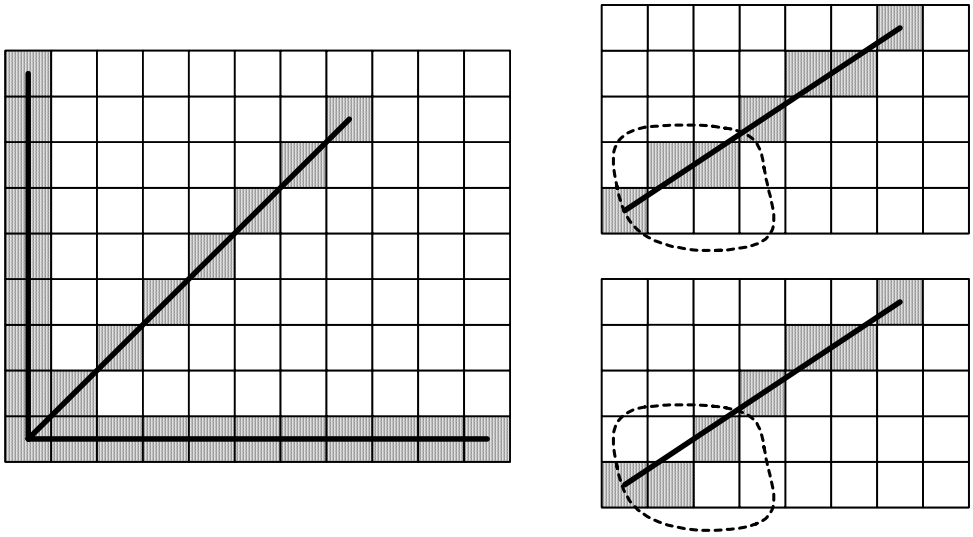

Рис. 1.11. Разложение в растр отрезков

Общие требования к изображению отрезка.

- концы отрезка должны находиться в заданных точках;
- отрезки должны выглядеть прямыми,
- яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона.

Ни одно из этих условий не может быть точно выполнено на растровом дисплее в силу того, что изображение строится из пикселов конечных размеров, а именно:

- концы отрезка в общем случае располагаются на пикселах, лишь наиболее близких к требуемым позициям и только в частных случаях координаты концов отрезка точно совпадают с координатами пикселов;
- отрезок аппроксимируется набором пикселов и лишь в частных случаях вертикальных, горизонтальных и отрезков под 45° они будут выглядеть прямыми, причем гладкими прямыми, без ступенек только для вертикальных и горизонтальных отрезков;
- яркость для различных отрезков и даже вдоль отрезка в общем случае различна, так как, например, расстояние между центрами пикселов для вертикального отрезка и отрезка под 45° различно (см. рис. 1.11).

Объективное улучшение аппроксимации достигается увеличением разрешения дисплея, но в силу существенных технологических проблем разрешение для растровых систем приемлемой скорости разрешение составляет порядка 1280×1024.

Субъективное улучшение аппроксимации основано на психофизиологических особенностях зрения и, в частности, может достигаться просто уменьшением размеров экрана. Другие способы субъективного улучшения качества аппроксимации основаны на различных программных ухищрениях по "размыванию" резких границ изображения.

1.2.3 Параметрический алгоритм рисования линии.

Необходимо провести линию из точки (x_1, y_1) в точку (x_2, y_2) с линейной интерполяцией по яркости (рис. 1.12).

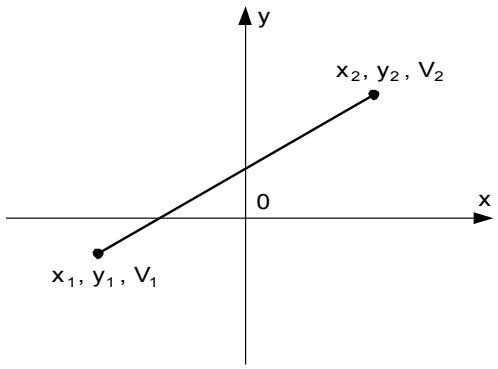

Рис. 1.12 Параметрическое рисование линии

Любую точку на этой линии можно представить в виде

$$x_{i+1} =]x_i + t \cdot (x_2 - x_1)[$$
$$y_{i+1} =]y_i + t \cdot (y_2 - y_1)[$$
$$V_{i+1} =]V_i + t \cdot (V_2 - V_1)[$$; где $t \in [0;1]$,][– знак округления до целого.

$$t = \frac{1}{\max\{|x_2 - x_1|, |y_2 - y_1|\}} = \frac{1}{N-1};$$ N – длина линии в пикселях.

Можно проводить вычисления через приращение координат.

$$\Delta x = \frac{x_2 - x_1}{N-1}; \Delta y = \frac{y_2 - y_1}{N-1}; \Delta V = \frac{V_2 - V_1}{N-1}$$

Значения приращений считаются в начале функции и не входят в цикл построения линии на экране, за счет чего повышается быстродействие.

Недостатки алгоритма:

- Необходимость работать с вещественными числами.
- В алгоритме есть операция деления, что значительно усложняет аппаратную организацию и увеличивает время работы алгоритма..

Достоинства алгоритма:

- Простота программной реализации.
- Простота реализации линейной интерполяции по яркости.

1.2.4 Алгоритм Брезенхема рисования линии.

В 1965 году Брезенхеймом был предложен простой целочисленный алгоритм для растрового построения отрезка. В алгоритме используется управляющая переменная d_i, которая на каждом шаге пропорциональна разности между s и t (см. рис. 1.13). На рис.1.18 приведен i-ый шаг, когда пиксел P_{i-1} уже найден как ближайший к реальному изображаемому отрезку, и теперь требуется определить, какой из пикселов должен быть установлен следующим: T_i или S_i.

Если s<t, то S_i ближе к отрезку и необходимо выбрать его; в противном случае ближе будет T_i. Другими словами, если s-t<0, то выбирается S_i; в противном случае выбирается T_i.

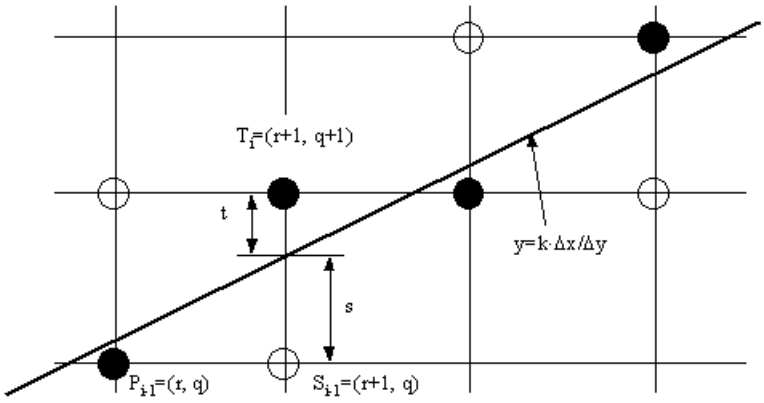

Рис. 1.13. Алгоритм Брезенхейма

Изображаемый отрезок проводится из точки (x_1, y_1) в точку (x_2, y_2). Пусть первая точка находится ближе к началу координат, тогда перенесем обе точки, T(x_1, y_1) так, чтобы начальная точка отрезка оказалась в начале координат, тогда конечная окажется в (Δ x, Δ y), где Δ x= x_2- x_1 , Δ y= x_2- x_1. Уравнение прямой теперь имеет вид y=x· Δ y/Δx. Из рисунка следует, что $s = \frac{\Delta y}{\Delta x}(r+1) - q; t = q + 1 - \frac{\Delta y}{\Delta x}(r+1)$.

поэтому $s - t = 2\frac{\Delta y}{\Delta x}(r+1) - 2q - 1$.

помножим на Δ x: $\Delta x(s-t) = 2(\Delta y \cdot r - q \cdot \Delta x) + 2\Delta y - \Delta x$

так как Δ x>0, величину Δ x(s-t) можно использовать в качестве критерия для выбора пиксела. Обозначим эту величину d_i : $d_i = 2(\Delta y \cdot x_{i-1} - y_{i-1} \cdot \Delta x) + 2\Delta y - \Delta x$

так как r = x_{i-1}, q = y_{i-1}, получаем: $\Delta_i = d_{i+1} - d_i = 2\Delta y(x_i - x_{i-1}) - 2\Delta x(y_i - y_{i-1})$

Известно, что x_I - x_{i-1}=1.

Если d_i >= 0, то выбираем T_i, тогда $\Delta_i = 2(\Delta y - \Delta x)$

Если d_i <0, то выбираем S_i, тогда $\Delta_i = 2\Delta y$

Таким образом, мы получили итеративную формулу для вычисления критерия d_i.

Начальное значение d_1=2Δ y-Δ x.

Можно построить блок схему алгоритма:

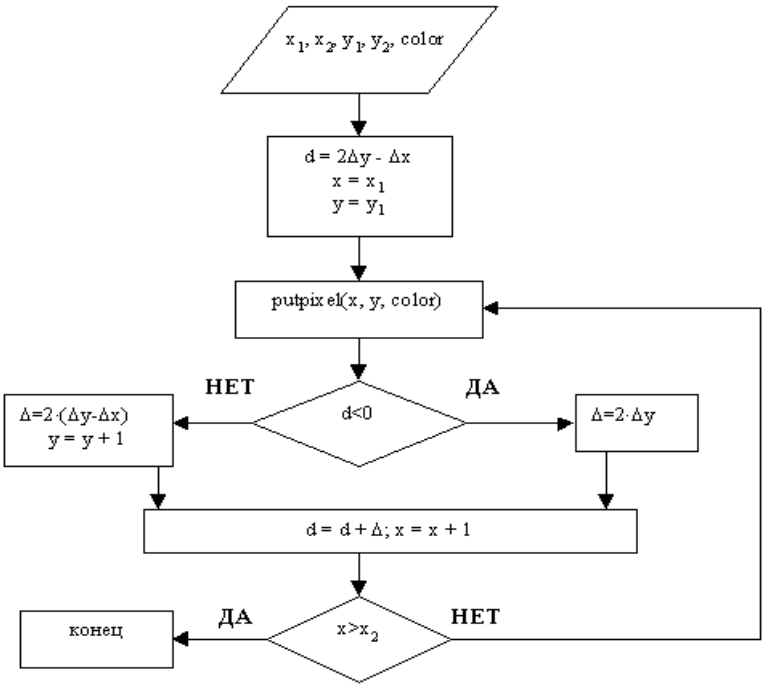

Рис. 1.14 Структурная схема алгоритма

Линейная интерполяция по яркости при построении отрезка по алгоритму Брезенхема получается параметрическим способом, т.е.

$V = V_{i-1} + \Delta V; \Delta V = \frac{V_2 - V_1}{N - 1}$, где N – длина отрезка в пикселях.

1.2.5 Алгоритмы построения окружности.

Рассмотрим окружность с центром в начале координат, для которой $x^2+y^2=R^2$, или в параметрической форме:

x=R· cos(a);
y=R· sin(a).

То есть легко написать программу рисования окружности:

```
void Circle (int x, int y, int R, int color)
{
    int a;
    int x1;
    int x2;
    int y1;
    int y2;
    x2=x+R;
    y2=y;
    for ( int a=1; a<=360; a++)
    {
        x1=x2; y1=y2;
        x2=round(R*cos(a))+x;
        y2=round(R*sin(a))+x;
        Line (x1, y1, x2, y2, color);
    }
}
```

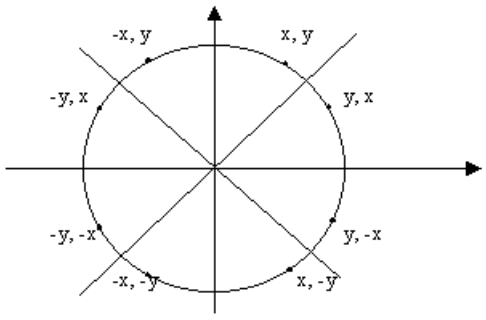

Рис. 1.15. Параметрический алгоритм рисования окружности

Если воспользоваться симметрией окружности, то можно построить более эффективный алгоритм. Если точка (x, y) лежит на окружности, то легко вычислить семь точек, принадлежащих окружности, симметричных этой. То есть, имея функцию вычисления значения y по *x=0..R/SQRT(2)* для построения дуги от 0^0 до 45^0. Построим процедуру, которая будет по одной координате ставить восемь точек, симметричных центру окружности.

```
void Circle_Pixel(int x0, int y0, int x, int y, int color);
{
    putpixel(x0 + x, y0 + y, color);
    putpixel(x0 + y, y0 + x, color);
    putpixel(x0 + y, y0 - x, color);
    putpixel(x0 + x, y0 - y, color);
    putpixel(x0 - x, y0 - y, color);
    putpixel(x0 - y, y0 - x, color);
```

```
    putpixel(x0 - y, y0 + x, color);
    putpixel(x0 - x, y0 + y, color);
}
```

Таким образом можно написать программу рисование окружности по точкам:

```
void Circle (int x0, int y0, int R, int color)
{
    for ( int x=0; x<=R/sqrt(2); x++)
    {
        int y = (int)(sqrt(sqr(R)-sqr(x)));
        Circle_Pixel (x0, y0, x, y, color);
    }
}
```

1.2.6 Алгоритм Брезенхема генерации окружности

Брезенхем разработал алгоритм более эффективный, чем каждый из рассмотренных выше. Здесь используется тот же принцип, что и для рисования линии.

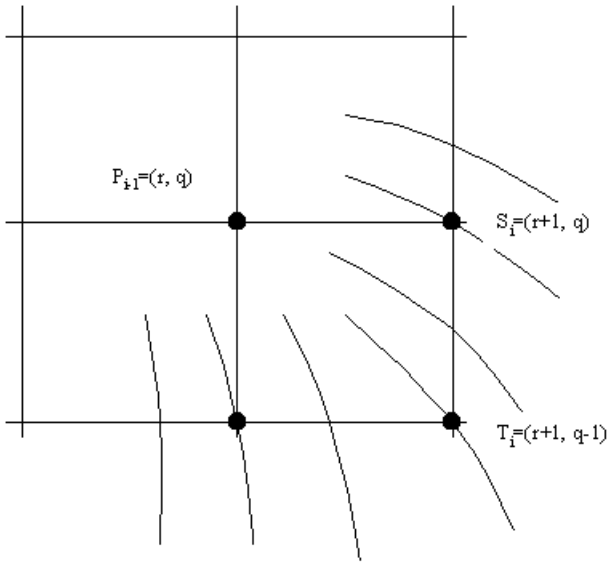

Рис. 1.16. Алгоритм Брезенхейма рисования окружности

Будем рассматривать сегмент окружности, соответствующий $x=x_0.. x_0+R/sqrt(2)$. На каждом шаге выбираем точку, ближайшую к реальной окружности. В качестве ошибки возьмем величину $D(P_i)=(x^2_i+ y^2_i) – R^2$.

Рассмотрим рис. 2.6.1 , на котором показаны различные возможные способы прохождения истинной окружности через сетку пикселей. Пусть пиксел P_{i-1} уже найден как ближайший к реальной изображенной окружности, и теперь требуется определить, какой из пикселей должен быть установлен следующим: T_i или S_i. Для этого определим точку, которой соответствует минимальная ошибка:

$$D(S_i)=((q+1)^2+ p^2) – R^2,\ D(T_i)=((q+1)^2+ (p+1)^2) – R^2.$$

Если $|D(S_i)| < |D(T_i)|$, то выбираем S_i, иначе - T_i. Введем величину $d_i = |D(S_i)| - |D(T_i)|$, тогда S_i выбирается при $d_i < 0$, иначе выбирается T_i. Если рассматривать только часть окружности, дугу от 0^0 до 45^0, то $D(S_i) > 0$ так как точка S_i лежит за пределами окружности, а $D(T_i) < 0$, так как T_i находится внутри окружности, поэтому $d_i = D(S_i) + D(T_i)$.

Алгебраические вычисления, аналогичные тем, которые проводились для линии, приводят к результату:

$$d_1=3-2R.$$

Если выбираем S_i (когда $d_i < 0$),

$$\Delta_i=4x_{i-1}+6;$$

если выбираем T_i (когда $d_i >=0$),

$$\Delta_i=4(x_{i-1} * y_{i-1}) +10.$$

Блок-схема этого алгоритма:

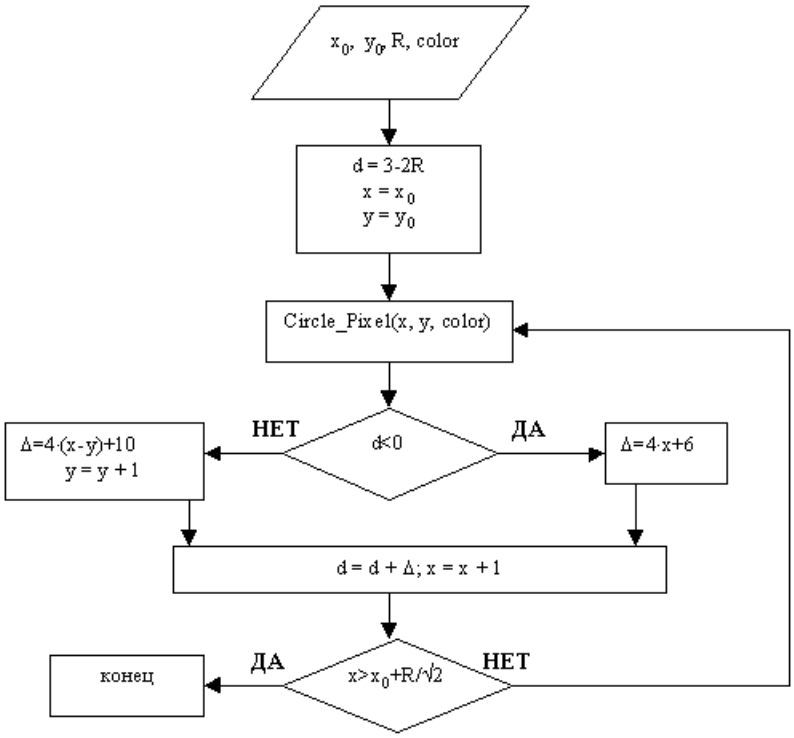

Рис. 1.17. Структурная схема алгоритма

Алгоритм хорош тем что отсутствуют операции с плавающей точкой, а также операции деления и извлечения корня.