Алгоритмы генерации графических примитив в дискретном пространстве. Классификация методов. Метод инкрементации. Алгоритм DDA, алгоритм Брессенхам (Bresenham) для отрезков. Fлгоритм Брессенхам растеризации окружностей.

Основные алгоритмы генерирования графических примитив

При описании геометрических объектов используются такие математические объекты и их свойства как прямые и плоскости, кривые линии, двухмерные кривые, поверхности, кривизна линий на поверхности, криволинейные координаты. В компьютерной графике могут решаться системы линейных и нелинейных уравнений. Это позволяет определять точки пересечения линий, линии и поверхности, построение линий пересечения поверхностей и другие вычисления.

Одним из важнейших инструментов систем автоматизированного проектирования и программ компьютерной графики стали **сплайны**. Сплайн – гладкая кривая, которая проходит через две или более опорных точек, а также имеет расположенные вне ее управляющие точки, влияющие на форму сплайна. Наиболее общие типы сплайнов – кривые Безье и B-сплайны (B-spline curves). Сплайны состоят из вершин (Vertices) и сегментов (Segments). Каждая вершина сплайна имеет касательные векторы (Tangents), снабженные на концах управляющими точками, или маркерами (Handels). Маркеры касательных векторов управляют кривизной сегментов сплайна при входе в вершину, которой принадлежат касательные векторы, и выходе из нее.

Классификация алгоритмов компьютерной графики

Алгоритмы машинной графики можно разделить на два уровня: **нижний и верхний**.

Группа алгоритмов нижнего уровня предназначена для реализации графических примитивов (линий, окружностей, заполнений и т.п.). Эти алгоритмы воспроизведены в графических библиотеках языков высокого уровня или реализованы в графических процессорах рабочих станций.

Среди алгоритмов нижнего уровня можно выделить следующие группы:

Простейшие в смысле используемых математических методов и отличающиеся простотой реализации. Как правило, такие алгоритмы не являются наилучшими по объему выполняемых вычислений или требуемым ресурсам памяти.

Поэтому можно выделить вторую группу алгоритмов, использующих **более сложные** математические предпосылки и отличающихся большей эффективностью.

К третьей группе следует отнести алгоритмы, которые могут быть без больших затруднений реализованы аппаратно (допускающие распараллеливание, рекурсивные, реализуемые в простейших командах). В эту группу могут попасть и алгоритмы, представленные в первых двух группах.

К четвертой группе можно отнести алгоритмы со специальным назначением (например, для устранения лестничного эффекта).

К алгоритмам верхнего уровня относятся в первую очередь алгоритмы удаления невидимых линий и поверхностей. Задача удаления невидимых линий и поверхностей продолжает оставаться центральной в машинной графике. От эффективности алгоритмов, позволяющих решить эту задачу, зависят качество и скорость построения трехмерного изображения.

К задаче удаления невидимых линий и поверхностей примыкает задача построения (закрашивания) полутоновых (реалистических) изображений, т.е.

учета явлений, связанных с количеством и характером источников света, учета свойств поверхности тела (прозрачность, преломление, отражение света).

Однако при этом не следует забывать, что вывод объектов в алгоритмах верхнего уровня обеспечивается примитивами, реализующими алгоритмы нижнего уровня, поэтому нельзя игнорировать проблему выбора и разработки эффективных алгоритмов нижнего уровня.

Для разных областей применения машинной графики на первый план могут выдвигаться разные свойства алгоритмов. Для научной графики большое значение имеет универсальность алгоритма, быстродействие может отходить на второй план. Для систем моделирования, воспроизводящих движущиеся объекты, быстродействие становится главным критерием, поскольку требуется генерировать изображение практически в реальном масштабе времени.

Особенности растровой графики связаны с тем, что обычные изображения, с которыми сталкивается человек в своей деятельности (чертежи, графики, карты, художественные картины и т.п.), реализованы на плоскости, состоящей из бесконечного набора точек. Экран же растрового дисплея представляется матрицей дискретных элементов, имеющих конкретные физические размеры. При этом число их существенно ограничено. Поэтому нельзя провести точную линию из одной точки в другую, а можно выполнить только аппроксимацию этой линии с отображением ее на дискретной матрице (плоскости). Такую плоскость также называют целочисленной решеткой, растровой плоскостью или растром. Эта решетка представляется квадратной сеткой с шагом 1. Отображение любого объекта на целочисленную решетку называется разложением его в растр или просто растровым представлением.

Процесс генерации растрового образа геометрического объекта принято называть растрированием.

Перед рассмотрением конкретных алгоритмов сформулируем 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 (рис. 2);
- 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. 2).

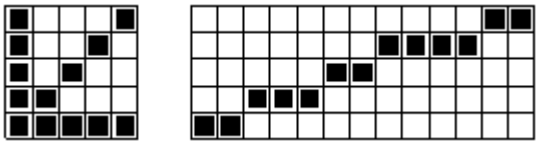


Рис. 2: Растровое представление различных векторов

Объективное улучшение аппроксимации достигается увеличением разрешения дисплея.

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.

Цифровой дифференциальный анализатор

С помощью ЦДА (Цифровой дифференциальный анализатор или DDA – Digital Differential Analyser) решается, дифференциальное уравнение отрезка, имеющее вид:

$$\frac{dy}{dx} = \frac{Px}{Py}$$

где $Py = Yk - Yn$ - приращение координат отрезка по оси Y, а

$Px = Xk - Xn$ - приращение координат отрезка по оси X.

При этом ЦДА формирует дискретную аппроксимацию непрерывного решения этого дифференциального уравнения. В обычном ЦДА, используемом, в векторных устройствах, определяется количество узлов N, используемых для аппроксимации отрезка. Затем за N циклов вычисляются координаты очередных узлов:

$$Xo = Xn; \qquad X_{i+1} = X_i + Px/N$$
$$Yo = Xn; \qquad Y_{i+1} = Y_i + Py/N$$

Получаемые значения X_i, Y_i преобразуются в целочисленные значения координаты очередного подсвечиваемого пиксела либо с округлением, либо с отбрасыванием дробной части.

Генератор векторов, использующий этот алгоритм, имеет недостаток – точки могут прописываться дважды, это увеличивает время построения. Кроме того из-за независимого вычисления обеих координат нет предпочтительных направлений и построенные отрезки кажутся не очень красивыми.

Субъективно лучше смотрятся вектора с единичным шагом по большей относительной координате (несимметричный ЦДА). Для Px > Py (при Px, Py > 0) это означает, что координата по X направлению должна увеличиться на 1 Px раз, а координата по Y-направлению должна также Px раз увеличиться, но на Py/Px. Т.е. количество узлов аппроксимации берется равным числу пикселов вдоль наибольшего приращения.

Для генерации отрезка из точки (x1, y1) в точку (x2, y2) в первом октанте (Px i Py i 0) алгоритм несимметричного ЦДА имеет вид:

1. Вычислить приращения координат:

Px= x2 - x1;

Py= y2 - y1;

2. Занести начальную точку отрезка

PutPixel (x1, y1);

3. Сгенерировать отрезок

while (x1 < x2) {

x1:= x1 + 1.0;

y1:= y1 + Py/Px;

PutPixel (x1, y1);

}

Пример генерации отрезка по алгоритму несимметричного ЦДА приведен на рис.3.

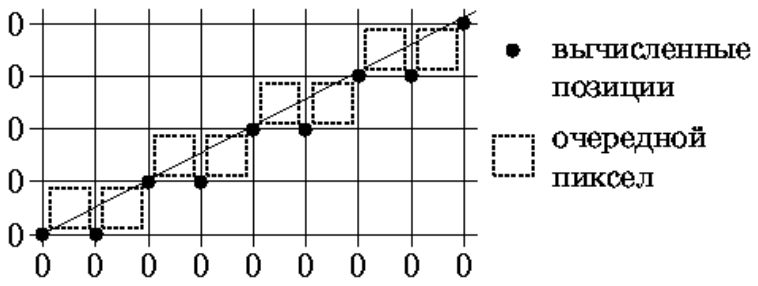


Рис. 3: Генерация отрезка несимметричным ЦДА

Алгоритм Брезенхема

Так как приращения координат, как правило, не являются целой степенью двойки, то в ЦДА-алгоритме требуется выполнение деления, что не всегда желательно, особенно при аппаратной реализации.

Брезенхем предложил алгоритм, не требующий деления, как в алгоритме несимметричного ЦДА, но обеспечивающий минимизацию отклонения сгенерированного образа от истинного отрезка, как в алгоритме обычного ЦДА.

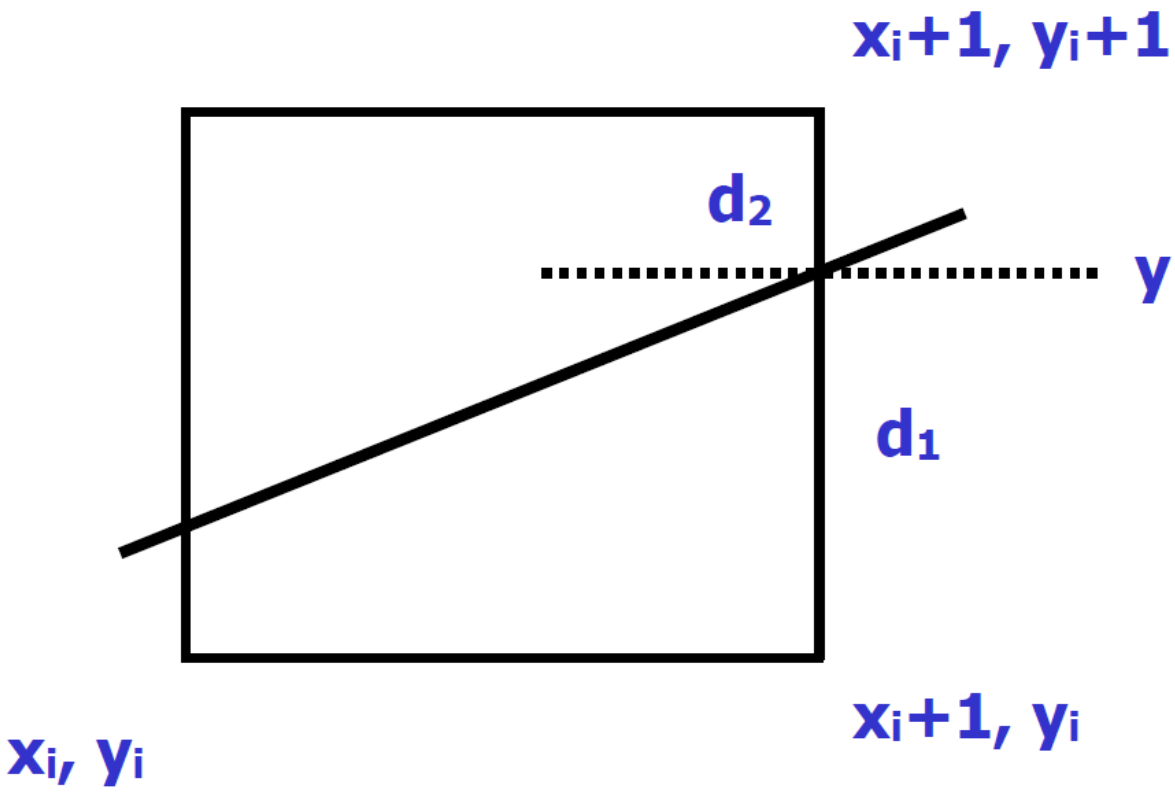
Алгоритм Брезенхэма также основан на методе инкрементации, но содержит только целочисленные операции.

Алгоритм определен для векторов с отрезком в интервале от 0 до 1.

Для каждого значения **x** определяется точку дискретного пространства, которая ближе к точке теоретического вектора.

Выбор основывается на расстояниях от двух точек-кандидатов (точка над вектором и точка под вектором) до соответствующей точки на векторе.

Пусть m = (y2-y1) / (x2-x1) - наклон вектора, и (xi, yi) - последняя точка дискретного пространства, выбранного в процессе генерации вектора



Обозначаем:

d1 - расстояние от теоретического вектора до точки O (xi + 1, yi)

d2 - расстояние от теоретического вектора до D (xi + 1, yi + 1)

Следующей выбранной точкой будет **O**, если **d1 <d2**, или точка **D** в противном случае. Если **d1 = d2**, вы можете выбрать любую из двух точек.

Выражаем разницу d1-d2:

$$y = m * (x_i + 1) + b$$

является координатой точки на теоретическом векторе.

$$d_1 = y - y_i = m * (x_i + 1) + b - y_i$$

$$d_2 = y_i + 1 - y = y_i + 1 - m * (x_i + 1) - b$$

$$d_1 - d_2 = +2 * m * (x_i + 1) - 2 * y_i + 2 * b - 1$$

Заменяем m на dy / dx, затем умножается на dx, и получаем:

$$t_i = (d_1 - d_2) * d_x = 2 * d_y * (x_i + 1) - 2 * d_x * y_i + 2 * b * d_x - d_x$$

t_i - представляет ошибку аппроксимации на шаге ***i***

Значение $c = 2 * b * d_x - d_x + 2 * d_y$ одинаково для любого шага, поэтому: $t_i = 2 * d_y * x_i - 2 * d_x * y_i + c$

Обозначаем (xi + 1, yi + 1) точку, выбранную на текущем шаге.

Тогда выражение ошибки аппроксимации для следующего шага будет:

$$t_{i+1} = 2 * d_y * x_{i+1} - 2 * d_x * y_{i+1} + c$$

Если $t_i <= 0$ выбираем точку O, то есть $x_{i+1} = x_i + 1$ и $y_{i+1} = y_i$. Получаем:

$$t_{i+1} = 2 * d_y * (x_i + 1) - 2 * d_x * y_i + c$$

или:

$$t_{i+1} = t_i + 2 * d_y$$

Если $t_i > 0$ выбираем точку D, то есть $x_{i+1} = x_i + 1$ и $y_{i+1} = y_i + 1$. Получаем:

$$t_{i+1} = 2 * d_y * (x_i + 1) - 2 * d_x * (y_i + 1) + c$$

или:

$$t_{i+1} = t_{i+2} * d_y - 2 * d_x$$

Ошибка аппроксимации для первого шага вычисляется путем замены в выражение:

$$t_1 = 2 * d_y * x_1 - 2 * d_x * y_1 + 2 * d_y - d_x + 2 * d_x(y_1 - \left({d_y}/{d_x}\right) * x_1)$$

$$t_1 = 2 * d_y - d_x$$

ГЕНЕРАЦИЯ ОКРУЖНОСТИ

Во многих областях кроме прямых отрезков и строк текстов, являются окружности и эллипсы, параболы и гиперболы. Наиболее используемым примитивом, является окружность. Один из наиболее простых и эффективных алгоритмов генерации окружности является алгоритм Брезенхемом.

Алгоритм Брезенхема для растеризации окружности

Рассмотрим генерацию 1/8 окружности, центр которой лежит в начале координат. Остальные части окружности могут быть получены последовательными отражениями (использованием симметрии точек на окружности относительно центра и осей координат).

Окружность с центром в начале координат описывается уравнением:

$$X^2 + Y^2 = R^2$$

Алгоритм Брезенхема пошагово генерирует очередные точки окружности, выбирая на каждом шаге для занесения пиксела точку растра $P_i(X_i, Y_i)$, ближайшую к истинной окружности, так чтобы ошибка:

$$Ei(Pi) = ({X_i}^2 + {Y_i}^2) - R^2$$

была минимальной. Причем, как и в алгоритме Брезенхема для генерации отрезков, выбор ближайшей точки выполняется с помощью анализа значений управляющих переменных, для вычисления которых не требуется вещественной арифметики. Для выбора очередной точки достаточно проанализировать знаки.

Рассмотрим генерацию 1/8 окружности по часовой стрелке, начиная от точки X=0, Y=R.

Проанализируем возможные варианты занесения i+1-й точки, после занесения i-й.

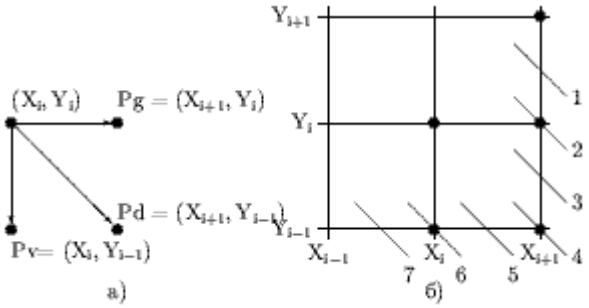


Рис. 5: Варианты расположения очередного пиксела окружности

При генерации окружности по часовой стрелке после занесения точки (X_i, Y_i) следующая точка может быть (см. рис. 5 а) либо Pg = (X_{i+1}, Y_i) - перемещение по горизонтали, либо Pd = (X_{i+1}, Y_{i-1}) - перемещение по диагонали, либо Pv = (X_i, Y_{i-1}) - перемещение по вертикали.

Для этих возможных точек вычислим и сравним абсолютные значения разностей квадратов расстояний от центра окружности до точки и окружности:

$$|Dg| = |(X+1)^2 + Y^2 - R^2|$$
$$|Dd| = |(X+1)^2 + (Y-1)^2 - R^2|$$
$$|Dg| = |X^2 + (Y-1)^2 - R^2|$$

Выбирается и заносится та точка, для которой это значение минимально.

Выбор способа расчета определяется по значению Dd.

Если:

Dd < 0, то диагональная точка внутри окружности. Это варианты 1-3 (см. рис. 5 б).

Dd > 0, то диагональная точка вне окружности. Это варианты 5-7.

Dd = 0, то диагональная точка лежит точно на окружности. Это вариант 4.

Рассмотрим случаи различных значений Dd в только что приведенной последовательности.

ОТСЕЧЕНИЕ ОТРЕЗКОВ

Если изображение выходит за пределы экрана, то на части дисплеев увеличивается время построения за счет того, что изображение строится в "уме". В некоторых дисплеях выход за пределы экрана приводит к искажению картины, так как координаты просто ограничиваются при достижении ими граничных значений, а не выполняется точный расчет координат пересечения (эффект "стягивания" изображения). Простые дисплеи не допускают выхода за пределы экрана. Все это, особенно в связи с широким использованием технологии просмотра окнами, требует выполнения отсечения сцены по границам окна видимости.

В простых графических системах достаточно двумерного отсечения, в трехмерных пакетах используется трех и четырехмерное отсечение. Последнее выполняется в ранее рассмотренных однородных координатах, позволяющих единым образом выполнять аффинные и перспективные преобразования.

Программное отсечение достаточно медленный процесс, поэтому, в мощные дисплеи встраивается соответствующая аппаратура.

Отсекаемые отрезки могут быть трех классов - целиком видимые, целиком невидимые и пересекающие окно. По способу выбора решения отбрасывания невидимого отрезка целиком или принятия его существует два основных типа алгоритмов отсечения - алгоритмы, использующие кодирование концов отрезка или всего отрезка и алгоритмы, использующие параметрическое представление отсекаемых отрезков и окна отсечения. Представители первого типа алгоритмов - алгоритм Коэна-Сазерленда (Cohen-Sutherland, CS-алгоритм) и FC-алгоритм (Fast Clipping - алгоритм). Алгоритмы второго типа - алгоритм Кируса-Бека (Curus-Beck, CB - алгоритм) и более поздний алгоритм Лианга-Барски (Liang-Barsky, LB-алгоритм).

Алгоритмы с кодированием применимы для прямоугольного окна, стороны которого параллельны осям координат, в то время как алгоритмы с параметрическим представлением применимы для произвольного окна.

Алгоритм Коэна-Сазерленда, являющийся стандартом алгоритма отсечения линий и обладающий одним из лучших быстродействий при компактной реализации. Наиболее быстрый, но и чрезвычайно громоздкий алгоритм FC. Алгоритм Лианга-Барски отсекает прямоугольное окном с использованием параметрического представления. Быстродействие этого алгоритма сравнимо с быстродействием алгоритма Коэна-Сазерленда. Алгоритм Кируса-Бека использует параметрическое представление и позволяет отсекать произвольным выпуклым окном.

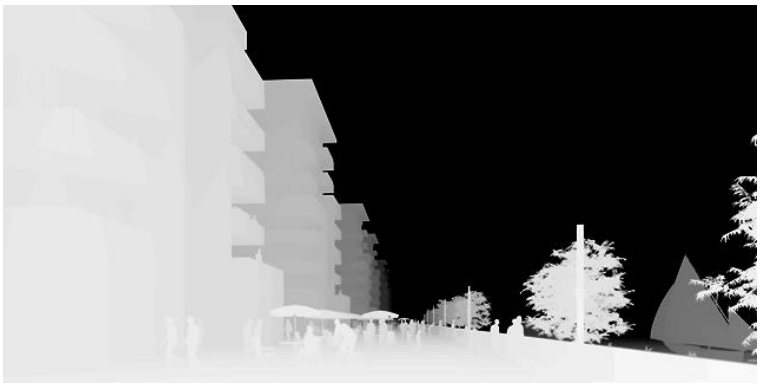
Z-буфер

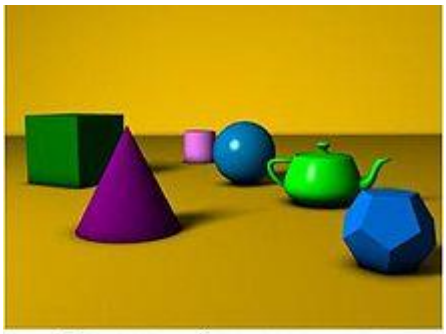
В компьютерной трёхмерной графике способом учёта удалённости элемента изображения – является алгоритм **Z-буфер**. Представляет собой один из вариантов решения «проблемы видимости». Очень эффективен и практически не имеет недостатков, если реализуется аппаратно. Программно же существуют другие методы, способные конкурировать с ним: Z-сортировка («алгоритм художника») и двоичное разбиение пространства (BSP), но они также имеют свои достоинства и недостатки. Основной недостаток Z-буферизации состоит в потреблении большого объёма памяти: в работе используется так называемый **буфер глубины** или **Z-буфер**.

Z-буфер представляет собой двумерный массив, каждый элемент которого соответствует пикселю на экране. Когда видеокарта рисует пиксель, его удалённость просчитывается и записывается в ячейку Z-буфера. Если пиксели двух рисуемых объектов перекрываются, то их значения глубины

сравниваются, и рисуется тот, который ближе, а его значение удалённости сохраняется в буфер. Получаемое при этом графическое изображение носит название z-depth (глубина) карта, представляющая собой полутоновое графическое изображение, каждый пиксель которого может принимать до 256 значений серого. По ним определяется удалённость от зрителя того или иного объекта трехмерной сцены. Карта широко применяется в постобработке для придания объёмности и реалистичности и создаёт такие эффекты, как глубина резкости, атмосферная дымка и т. д. Также карта используется в 3д-пакетах для текстурирования, делая поверхность рельефной.

- Примеры карт





Простая трёхмерная сцена

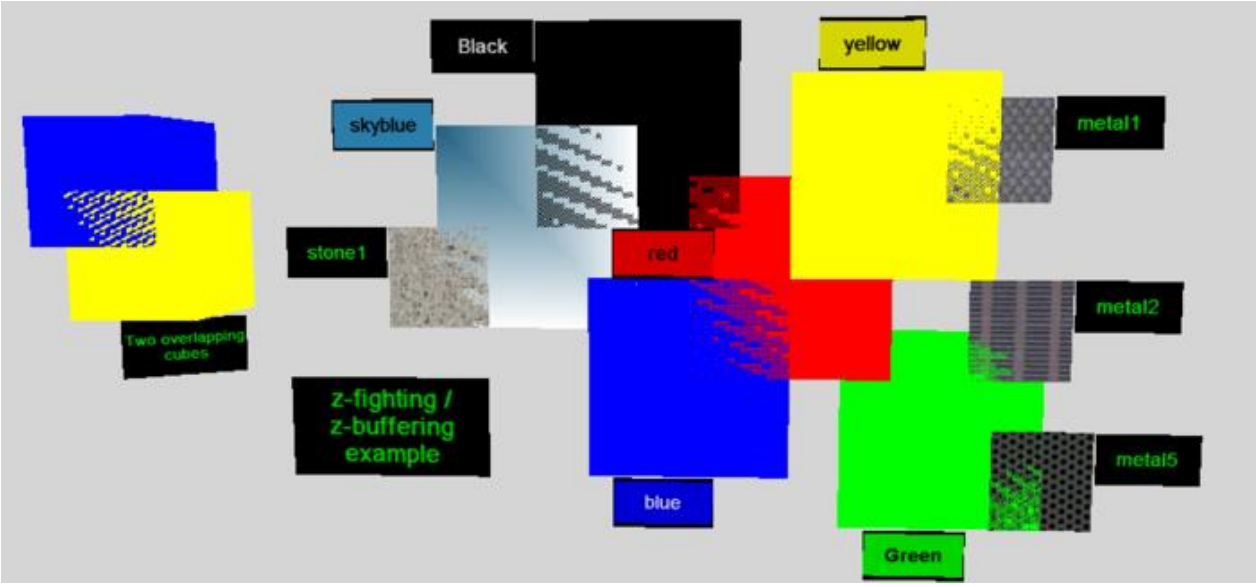
Представление в Z-буфере

Разрядность буфера глубины оказывает сильное влияние на качество визуализации: использование 16-битного буфера может привести к геометрическим искажениям, например, эффекту «борьбы», если два объекта находятся близко друг к другу. 24, 32-разрядные буферы хорошо справляются со своей задачей. 8-битные почти никогда не используются из-за низкой точности.

Z-конфликт

Если два объекта имеют близкую Z-координату, иногда, в зависимости от точки обзора, показывается то один, то другой, то оба полосатым узором. Это называется *Z-конфликт* (англ. *Z fighting*). Чаще всего конфликты присущи спецэффектам (декалям), накладывающимся на основную текстуру, например, дырам от пуль.

Решаются Z-конфликты сдвигом одного объекта относительно другого на величину, превышающую погрешность Z-буфера.



Двумерный алгоритм Коэна-Сазерленда

Этот алгоритм позволяет быстро выявить отрезки, которые могут быть или приняты, или отброшены целиком. Вычисление пересечений требуется, когда отрезок не попадает ни в один из этих классов. Этот алгоритм особенно эффективен в двух случаях:

- большинство примитивов содержится целиком в большом окне,
- большинство примитивов лежит целиком вне относительно маленького окна.

Идея алгоритма состоит в следующем:

Окно отсечения и прилегающие к нему части плоскости вместе образуют 9 областей (рис. 1). Каждой из областей присвоен 4-х разрядный код.

Две конечные точки отрезка получают 4-х разрядные коды, соответствующие областям, в которые они попали. Смысл разрядов кода:

1 рр = 1 - точка над верхним краем окна;

2 рр = 1 - точка под нижним краем окна;

3 рр = 1 - точка справа от правого края окна;

4 рр = 1 - точка слева от левого края окна.

Определение того лежит ли отрезок целиком внутри окна или целиком вне окна выполняется следующим оыбразом:

- если коды обоих концов отрезка равны 0 то отрезок целиком внутри окна, отсечение не нужно, отрезок принимается как тривиально видимый (отрезок AB на рис. 1);

- если логическое & кодов обоих концов отрезка не равно нулю, то отрезок целиком вне окна, отсечение не нужно, отрезок отбрасывается как тривиально невидимый (отрезок KL на рис. 1);

- если логическое & кодов обоих концов отрезка равно нулю, то отрезок подозрительный, он может быть частично видимым (отрезки CD, EF, GH) или целиком невидимым (отрезок IJ); для него нужно определить координаты пересечений со сторонами окна и для каждой полученной части определить тривиальную видимость или невидимость. При этом для отрезков CD и IJ потребуется вычисление одного пересечения, для остальных (EF и GH) - двух.

При расчете пересечения используется горизонтальность либо вертикальность сторон окна, что позволяет определить координату X или Y точки пересечения без вычислений.

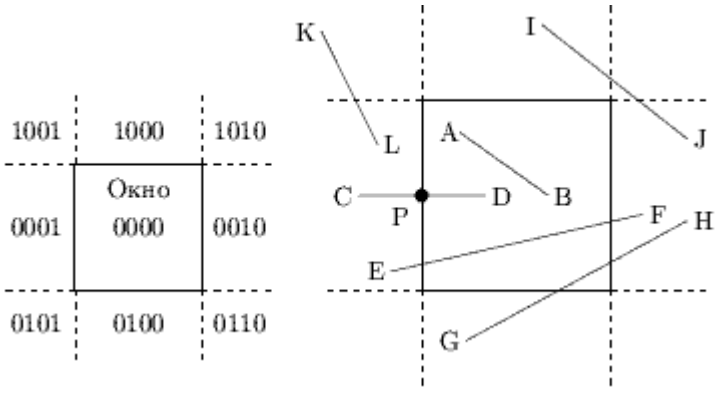


Рис. 1: Отсечение по методу Коэна-Сазерленда

При непосредственном использовании описанного выше способа отбора, целиком видимого или целиком невидимого отрезка после расчета пересечения потребовалось бы вычисление кода расположения точки пересечения. Для примера рассмотрим отрезок CD. Точка пересечения обозначена как P. В силу того, что граница окна считается принадлежащей окну, то можно просто принять только часть отрезка PD, попавшую в окно. Часть же отрезка CP, на самом деле оказавшаяся вне окна, потребует дальнейшего рассмотрения, так как логическое И кодов точек C и P даст 0, т.е. отрезок CP нельзя просто отбросить. Для решения этой проблемы Коэн и Сазерленд предложили заменять конечную точку с ненулевым кодом конца на точку, лежащую на стороне окна, либо на ее продолжении.

В целом схема алгоритма Коэна-Сазерленда следующая:

1. Рассчитать коды конечных точек отсекаемого отрезка.

В цикле повторять пункты 2-6:

2. Если логическое И кодов конечных точек не равно 0, то отрезок целиком вне окна. Он отбрасывается и отсечение закончено.

3. Если оба кода равны 0, то отрезок целиком видим. Он принимается и отсечение закончено.

4. Если начальная точка внутри окна, то она меняется местами с конечной точкой.

5. Анализируется код начальной точки для определения стороны окна с которой есть пересечение и выполняется расчет пересечения. При этом вычисленная точка пересечения заменяет начальную точку.

6. Определение нового кода начальной точки.

Тема 5 Основные алгоритмы генерирования графических примитив

При описании геометрических объектов используются такие математические объекты и их свойства как прямые и плоскости, кривые линии, двухмерные кривые, поверхности, кривизна линий на поверхности, криволинейные координаты. В компьютерной графике могут решаться системы линейных и нелинейных уравнений. Это позволяет определять точки пересечения линий, линии и поверхности, построение линий пересечения поверхностей и другие вычисления.

Одним из важнейших инструментов систем автоматизированного проектирования и программ компьютерной графики стали **сплайны**. Сплайн – гладкая кривая, которая проходит через две или более опорных точек, а также имеет расположенные вне ее управляющие точки, влияющие на форму сплайна. Наиболее общие типы сплайнов – кривые Безье и B-сплайны (B-spline curves). Сплайны состоят из вершин (Vertices) и сегментов (Segments). Каждая вершина сплайна имеет касательные векторы (Tangents), снабженные на концах управляющими точками, или маркерами (Handels). Маркеры касательных векторов управляют кривизной сегментов сплайна при входе в вершину, которой принадлежат касательные векторы, и выходе из нее.

Классификация алгоритмов компьютерной графики

Алгоритмы машинной графики можно разделить на два уровня: **нижний и верхний**.

Группа алгоритмов нижнего уровня предназначена для реализации графических примитивов (линий, окружностей, заполнений и т.п.). Эти

алгоритмы воспроизведены в графических библиотеках языков высокого уровня или реализованы в графических процессорах рабочих станций.

Среди алгоритмов нижнего уровня можно выделить следующие группы:

Простейшие в смысле используемых математических методов и отличающиеся простотой реализации. Как правило, такие алгоритмы не являются наилучшими по объему выполняемых вычислений или требуемым ресурсам памяти.

Поэтому можно выделить вторую группу алгоритмов, использующих **более сложные** математические предпосылки и отличающихся большей эффективностью.

К третьей группе следует отнести алгоритмы, которые могут быть без больших затруднений реализованы аппаратно (допускающие распараллеливание, рекурсивные, реализуемые в простейших командах). В эту группу могут попасть и алгоритмы, представленные в первых двух группах.

К четвертой группе можно отнести алгоритмы со специальным назначением (например, для устранения лестничного эффекта).

К алгоритмам верхнего уровня относятся в первую очередь алгоритмы удаления невидимых линий и поверхностей. Задача удаления невидимых линий и поверхностей продолжает оставаться центральной в машинной графике. От эффективности алгоритмов, позволяющих решить эту задачу, зависят качество и скорость построения трехмерного изображения.

К задаче удаления невидимых линий и поверхностей примыкает задача построения (закрашивания) полутоновых (реалистических) изображений, т.е. учета явлений, связанных с количеством и характером источников света, учета свойств поверхности тела (прозрачность, преломление, отражение света).

Однако при этом не следует забывать, что вывод объектов в алгоритмах верхнего уровня обеспечивается примитивами, реализующими алгоритмы нижнего уровня, поэтому нельзя игнорировать проблему выбора и разработки эффективных алгоритмов нижнего уровня.

Для разных областей применения машинной графики на первый план могут выдвигаться разные свойства алгоритмов. Для научной графики большое значение имеет универсальность алгоритма, быстродействие может отходить на второй план. Для систем моделирования, воспроизводящих движущиеся объекты, быстродействие становится главным критерием, поскольку требуется генерировать изображение практически в реальном масштабе времени.

Особенности растровой графики связаны с тем, что обычные изображения, с которыми сталкивается человек в своей деятельности (чертежи, графики, карты, художественные картины и т.п.), реализованы на плоскости, состоящей из бесконечного набора точек. Экран же растрового дисплея представляется матрицей дискретных элементов, имеющих конкретные физические размеры. При этом число их существенно ограничено. Поэтому нельзя провести точную линию из одной точки в другую, а можно выполнить только аппроксимацию этой линии с отображением ее на дискретной матрице (плоскости). Такую плоскость также называют целочисленной решеткой, растровой плоскостью или растром. Эта решетка представляется квадратной сеткой с шагом 1. Отображение любого объекта на целочисленную решетку называется разложением его в растр или просто растровым представлением.

Процесс генерации растрового образа геометрического объекта принято называть растрированием.

Перед рассмотрением конкретных алгоритмов сформулируем 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 (рис. 2);
- 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. 2).

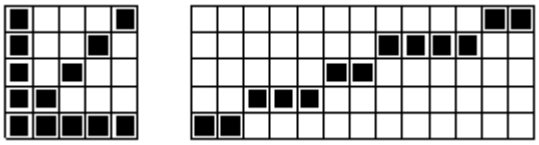


Рис. 2: Растровое представление различных векторов

Объективное улучшение аппроксимации достигается увеличением разрешения дисплея.

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.

Цифровой дифференциальный анализатор

С помощью ЦДА (Цифровой дифференциальный анализатор или DDA – Digital Differential Analyser) решается, дифференциальное уравнение отрезка, имеющее вид:

$$\frac{dy}{dx} = \frac{Px}{Py}$$

где $Py = Yk - Yn$ - приращение координат отрезка по оси Y, а

$Px = Xk - Xn$ - приращение координат отрезка по оси X.

При этом ЦДА формирует дискретную аппроксимацию непрерывного решения этого дифференциального уравнения. В обычном ЦДА, используемом, в векторных устройствах, определяется количество узлов N, используемых для аппроксимации отрезка. Затем за N циклов вычисляются координаты очередных узлов:

$$Xo = Xn; \qquad X_{i+1} = X_i + Px/N$$
$$Yo = Xn; \qquad Y_{i+1} = Y_i + Py/N$$

Получаемые значения X_i, Y_i преобразуются в целочисленные значения координаты очередного подсвечиваемого пиксела либо с округлением, либо с отбрасыванием дробной части.

Генератор векторов, использующий этот алгоритм, имеет недостаток – точки могут прописываться дважды, это увеличивает время построения. Кроме того из-за независимого вычисления обеих координат нет предпочтительных направлений и построенные отрезки кажутся не очень красивыми.

Субъективно лучше смотрятся вектора с единичным шагом по большей относительной координате (несимметричный ЦДА). Для Px > Py (при Px, Py >

0) это означает, что координата по X направлению должна увеличиться на 1 Px раз, а координата по Y-направлению должна также Px раз увеличиться, но на Py/Px. Т.е. количество узлов аппроксимации берется равным числу пикселов вдоль наибольшего приращения.

Для генерации отрезка из точки (x1, y1) в точку (x2, y2) в первом октанте (Px і Py і 0) алгоритм несимметричного ЦДА имеет вид:

4. Вычислить приращения координат:

Px= x2 - x1;

Py= y2 - y1;

5. Занести начальную точку отрезка

PutPixel (x1, y1);

6. Сгенерировать отрезок

while (x1 < x2) {

x1:= x1 + 1.0;

y1:= y1 + Py/Px;

PutPixel (x1, y1);

}

Пример генерации отрезка по алгоритму несимметричного ЦДА приведен на рис.3.

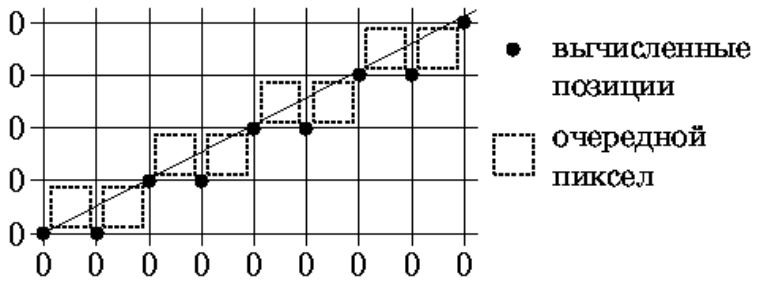


Рис. 3: Генерация отрезка несимметричным ЦДА

Алгоритм Брезенхема

Так как приращения координат, как правило, не являются целой степенью двойки, то в ЦДА-алгоритме требуется выполнение деления, что не всегда желательно, особенно при аппаратной реализации.

Брезенхем предложил алгоритм, не требующий деления, как в алгоритме несимметричного ЦДА, но обеспечивающий минимизацию отклонения сгенерированного образа от истинного отрезка, как в алгоритме обычного ЦДА.

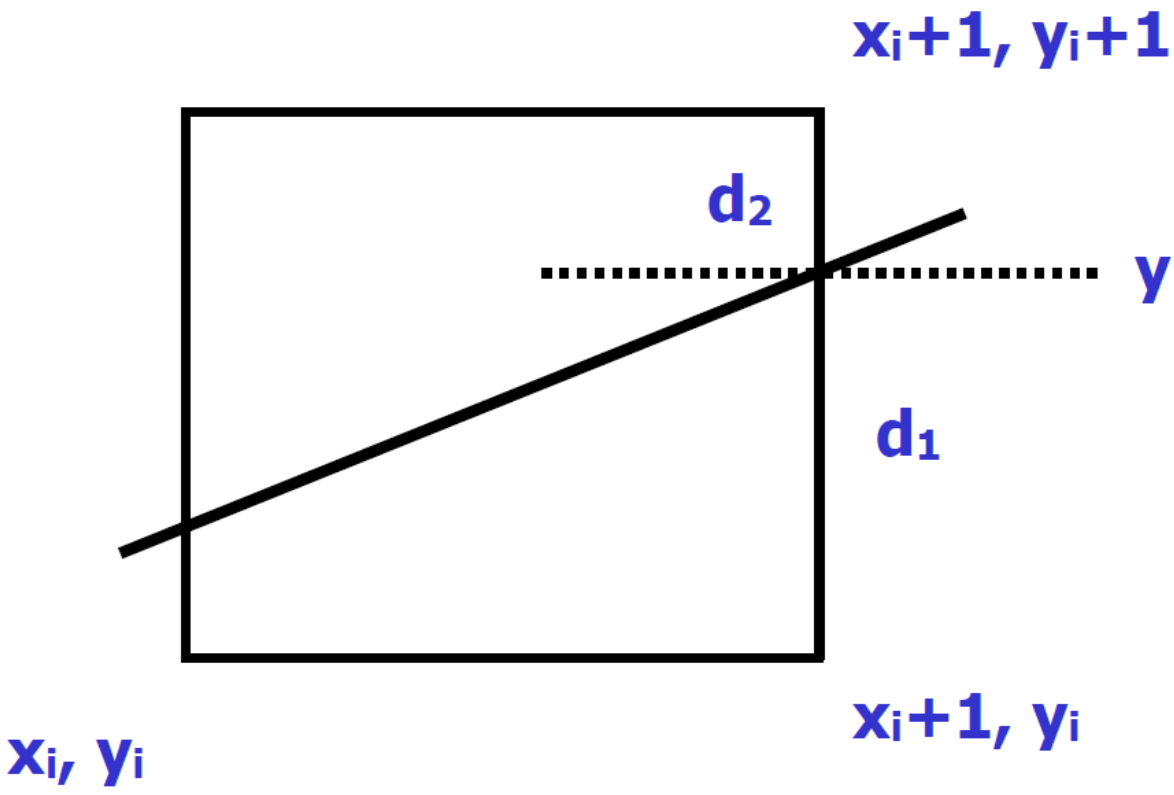
Алгоритм Брезенхэма также основан на методе инкрементации, но содержит только целочисленные операции.

Алгоритм определен для векторов с отрезком в интервале от 0 до 1.

Для каждого значения **x** определяется точку дискретного пространства, которая ближе к точке теоретического вектора.

Выбор основывается на расстояниях от двух точек-кандидатов (точка над вектором и точка под вектором) до соответствующей точки на векторе.

Пусть m = (y2-y1) / (x2-x1) - наклон вектора, и (xi, yi) - последняя точка дискретного пространства, выбранного в процессе генерации вектора



Обозначаем:

d1 - расстояние от теоретического вектора до точки O (xi + 1, yi)

d2 - расстояние от теоретического вектора до D (xi + 1, yi + 1)

Следующей выбранной точкой будет **O**, если **d1 <d2**, или точка **D** в противном случае. Если **d1 = d2**, вы можете выбрать любую из двух точек.

Выражаем разницу d1-d2:

$$y = m * (x_i + 1) + b$$

является координатой точки на теоретическом векторе.

$$d_1 = y - y_i = m * (x_i + 1) + b - y_i$$

$$d_2 = y_i + 1 - y = y_i + 1 - m * (x_i + 1) - b$$

$$d_1 - d_2 = +2 * m * (x_i + 1) - 2 * y_i + 2 * b - 1$$

Заменяем m на dy / dx, затем умножается на dx, и получаем:

$$t_i = (d_1 - d_2) * d_x = 2 * d_y * (x_i + 1) - 2 * d_x * y_i + 2 * b * d_x - d_x$$

t_i - представляет ошибку аппроксимации на шаге ***i***

Значение $c = 2 * b * d_x - d_x + 2 * d_y$ одинаково для любого шага, поэтому: $t_i = 2 * d_y * x_i - 2 * d_x * y_i + c$

Обозначаем (xi + 1, yi + 1) точку, выбранную на текущем шаге.

Тогда выражение ошибки аппроксимации для следующего шага будет:

$$t_{i+1} = 2 * d_y * x_{i+1} - 2 * d_x * y_{i+1} + c$$

Если $t_i <= 0$ выбираем точку O, то есть $x_{i+1} = x_i + 1$ и $y_{i+1} = y_i$. Получаем:

$$t_{i+1} = 2 * d_y * (x_i + 1) - 2 * d_x * y_i + c$$

или:

$$t_{i+1} = t_i + 2 * d_y$$

Если $t_i > 0$ выбираем точку D, то есть $x_{i+1} = x_i + 1$ и $y_{i+1} = y_i + 1$. Получаем:

$$t_{i+1} = 2 * d_y * (x_i + 1) - 2 * d_x * (y_i + 1) + c$$

или:

$$t_{i+1} = t_{i+2} * d_y - 2 * d_x$$

Ошибка аппроксимации для первого шага вычисляется путем замены в выражение:

$$t_1 = 2 * d_y * x_1 - 2 * d_x * y_1 + 2 * d_y - d_x + 2 * d_x(y_1 - \left({d_y}/{d_x}\right) * x_1)$$

$$t_1 = 2 * d_y - d_x$$

ГЕНЕРАЦИЯ ОКРУЖНОСТИ

Во многих областях кроме прямых отрезков и строк текстов, являются окружности и эллипсы, параболы и гиперболы. Наиболее используемым примитивом, является окружность. Один из наиболее простых и эффективных алгоритмов генерации окружности является алгоритм Брезенхемом.

Алгоритм Брезенхема для растеризации окружности

Рассмотрим генерацию 1/8 окружности, центр которой лежит в начале координат. Остальные части окружности могут быть получены последовательными отражениями (использованием симметрии точек на окружности относительно центра и осей координат).

Окружность с центром в начале координат описывается уравнением:

$$X^2 + Y^2 = R^2$$

Алгоритм Брезенхема пошагово генерирует очередные точки окружности, выбирая на каждом шаге для занесения пиксела точку растра $P_i(X_i, Y_i)$, ближайшую к истинной окружности, так чтобы ошибка:

$$Ei(Pi) = ({X_i}^2 + {Y_i}^2) - R^2$$

была минимальной. Причем, как и в алгоритме Брезенхема для генерации отрезков, выбор ближайшей точки выполняется с помощью анализа значений управляющих переменных, для вычисления которых не требуется вещественной арифметики. Для выбора очередной точки достаточно проанализировать знаки.

Рассмотрим генерацию 1/8 окружности по часовой стрелке, начиная от точки X=0, Y=R.

Проанализируем возможные варианты занесения i+1-й точки, после занесения i-й.

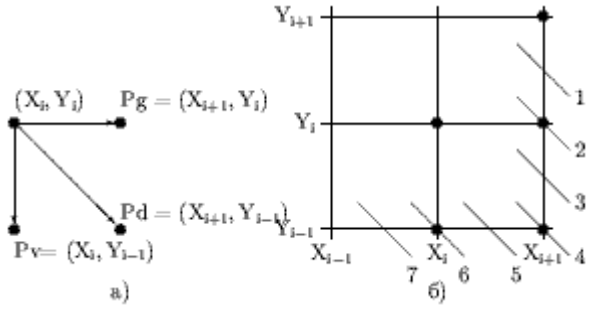


Рис. 5: Варианты расположения очередного пиксела окружности

При генерации окружности по часовой стрелке после занесения точки (X_i, Y_i) следующая точка может быть (см. рис. 5 а) либо Pg = (X_{i+1}, Y_i) - перемещение по горизонтали, либо Pd = (X_{i+1}, Y_{i-1}) - перемещение по диагонали, либо Pv = (X_i, Y_{i-1}) - перемещение по вертикали.

Для этих возможных точек вычислим и сравним абсолютные значения разностей квадратов расстояний от центра окружности до точки и окружности:

$$|Dg| = |(X+1)^2 + Y^2 - R^2|$$
$$|Dd| = |(X+1)^2 + (Y-1)^2 - R^2|$$
$$|Dg| = |X^2 + (Y-1)^2 - R^2|$$

Выбирается и заносится та точка, для которой это значение минимально.

Выбор способа расчета определяется по значению Dd.

Если:

Dd < 0, то диагональная точка внутри окружности. Это варианты 1-3 (см. рис. 5 б).

Dd > 0, то диагональная точка вне окружности. Это варианты 5-7.

Dd = 0, то диагональная точка лежит точно на окружности. Это вариант 4.

Рассмотрим случаи различных значений Dd в только что приведенной последовательности.