Растеризация

Способы представления изображений

Изображение может быть представлено в двух видах: в векторном и в растровом.

Векторное представление — способ представления объектов и изображений в компьютерной графике, основанный на использовании геометрических примитивов, таких как точки, линии, сплайны и многоугольники.

Растровое представление — способ представления объектов и изображений в компьютерной графике, основанный на использовании сетки пикселов. Каждый пиксел при таком представлении характеризуется своим цветом.

Важными характеристиками растрового изображения являются:

- Разрешение;
- Глубина цвета (количество цветов);
- Цветовое пространство.

Векторное представление обладает несколькими неоспоримыми преимуществами:

- Размер описания сцены зависит исключительно от сложности объектов, входящих в эту сцену;
- Векторное изображение можно масштабировать без серьёзных искажений;
- Для хранения параметров объектов используются аппаратно-независимые единицы измерения, а значит, можно допиться наилучшего качества растеризации на различных устройствах;
- При увеличении или уменьшении объектов толщина линий может быть задана постоянной величиной, независимо от реального размера контура.

Но, не смотря на все преимущества векторного представления, при выводе изображений на реальные устройства возникает необходимость в растеризации. Например, для отрезка пользователь зачастую задаёт только координаты начала и конца, а внутренние точки необходимо рассчитать алгоритму растеризации. Этот алгоритм вызывается при выводе каждого кадра, а значит, должен работать максимально быстро.

Основные критерии алгоритмов растеризации приведены ниже:

- Качество работы;
- Скорость выполнения.

На сегодняшний день предпочтение отдаётся скорости выполнения, даже в ущерб качеству.

Алгоритмы отрисовки отрезков

Главной задачей алгоритмов отрисовки отрезков является вычисление координат пикселов на двумерной растровой сетке. Основная сложность заключается в том, что конечные точки отрезков должны иметь целочисленные координаты. Самый очевидный алгоритм заключается в постепенном увеличении x, вычислении $y = k * x + b$ и последующем округлении полученного результата до целого числа. В результате будет закрашена точка с координатами $(x; round(y))$. У этого алгоритма есть довольно очевидный недостаток: вычисление произведения $k * x$ и последующее округление требует много времени. Т.е. алгоритм отрисовки получается сравнительно медленным.

На практике используется один из следующих методов:

- Пошаговый алгоритм;
- Алгоритм Брезенхема.

Пошаговый алгоритм

Наиболее сложная операция в описанном алгоритме – операция умножения. Её можно устранить, если заметить, что при увеличении x на 1, значение $k = \frac{\Delta y}{\Delta x}$ сводится к $k = \Delta y$. Т.е. изменение x на 1 приводит к изменению y на k. Таким образом, если $x_{i+1} = x_i + 1$, то $y_{i+1} = y_i + k$. После этого значение y округляется до ближайшего целого. Последующие точки вычисляются на основе уже полученных значений.

Стоит заметить, что если $k > 1$, то шаг по x будет приводить к смещению по y более чем на 1 пиксел. Такое смещение приведёт к разреженности в непрерывном отрезке. Чтобы избежать этого эффекта, необходимо поменять x и y местами: увеличивать y на 1, а x на $\frac{1}{k}$.

Пошаговый алгоритм работает быстрее, так как мы избавились от умножения на вещественное число, но сохранилось ещё одно сравнительно медленное действие – округление. Алгоритм Брезенхема позволяет избавиться и от этого недостатка.

Быстрые алгоритмы

При рассмотрении быстрых алгоритмов, мы будем предполагать, что отрезок начинается в точке $(0; 0)$ и полностью лежит в первом октанте. Такой случай мы будем называть каноническим. Для построения отрезка с произвольными координатами начала и конца можно воспользоваться любым из быстрых алгоритмом, а потом выполнить операции переноса и отражения относительно осей симметрии.

Алгоритм Брезенхема

Данный алгоритм характеризуется тем, что в нём используется только целочисленная арифметика. Вещественные переменные не применяются совсем, а значит, в округлении нет необходимости.

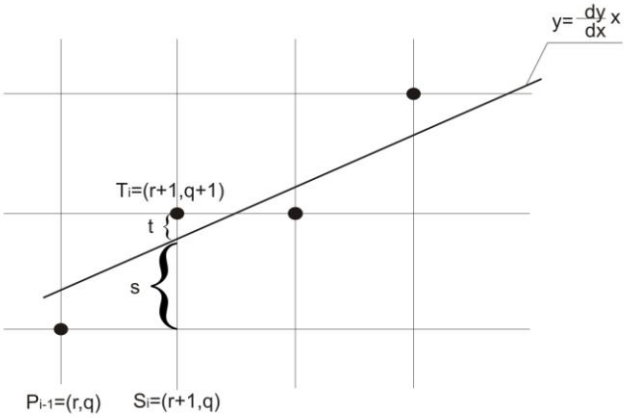

Рисунок 1 Схема алгоритма Брезенхема

В алгоритме используется специальная переменная d_i, которая на каждом шаге пропорциональная разности между s и t, где s - расстояние от идеальной линии до нижнего пиксела, t – расстояние от идеальной линии до верхнего пиксела (при заданном x). Для точки P_i нужно определить какой из пикселов будет выбран: T_i или S_i. Пиксе выбирается на основе критерия близости к идеальной линии. Если $s < t$, то будет выбрана точка S_i, в противном случае T_i.

Рассмотрим работу алгоритма подробнее. В качестве входных данных поступает отрезок с координатами начала и конца $(x_1; y_1) - (x_2; y_2)$ соответственно. Предположим, что точка $(x_1; y_1)$ находится ближе к началу координат, чем точка $(x_2; y_2)$. Применим к обеим точкам операцию переноса на вектор $(-x_1; -y_1)$. Тогда координаты начала отрезка будут равны $(0; 0)$, а координаты конца $(x_2 - x_1; y_2 - y_1) = (dx; dy)$. После такого преобразования уравнение прямой, на которой лежит отрезок, будет иметь вид $y = \frac{dy}{dx} * x$.

Обозначим координаты точки P_i после переноса через $(r; q)$. Тогда мы получим следующие значения для координат и отрезков:

$$S_i = (r + 1; q)$$

$$T_i = (r + 1; q + 1)$$

$$s = \frac{dy}{dx}(r + 1) - q$$

$$t = q + 1 - \frac{dy}{dx}(r + 1)$$

Найдём разность s и t:

$$s - t = 2\frac{dy}{dx}(r + 1) - 2q - 1$$

$$dx(s - t) = 2(r * dy - q * dx) + 2dy - dx$$

Величина dx всегда положительна, поэтому мы можем использовать значение $dx(s - t)$ для определения того, какой пиксел будет закрашен. Если эта величина отрицательная, выбирается точка S_i в противном случае, выбирается T_i.

Обозначив $d_i = dx(s-t)$, имеем:

$$d_i = 2(r * dy - q * dx) + 2dy - dx$$

Так как $r = x_{i-1}; q = y_{i-1}$, то

$$d_i = 2 * x_{i-1} * dy - 2 * y_{i-1} * dx + 2dy - dx$$

Прибавив единицу ко всем индексам, получим:

$$d_{i+1} = 2 * x_i * dy - 2 * y_i * dx + 2dy - dx$$

Вычтем полученные выражения друг из друга:

$$d_{i+1} - d_i = 2dy(x_i - x_{i-1}) - 2dx(y_i - y_{i-1})$$

Вспомним, что $x_i - x_{i-1} = 1$. Тогда получаем следующее выражение:

$$d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1})$$

$\boldsymbol{d_i < 0}$	$\boldsymbol{d_i \geq 0}$
Выбирается точка S_i $x_i = x_{i-1} + 1$ $y_i = y_{i-1}$ $d_{i+1} = d_i + 2dy$	Выбирается точка T_i $x_i = x_{i-1} + 1$ $y_i = y_{i-1} + 1$ $d_{i+1} = d_i + 2(dy - dx)$

Т.о. мы только что получили способ вычисления d_{i+1} на основе предыдущего значения и способ выбора очередной точки для закраски.

Начальное значение d_1 находится при $i = 1; (x_0; y_0) = (0; 0)$. Т.е. $d_1 = 2dy - dx$.

Алгоритм Кастла-Питвея

Введём две управляющие команды для плоттера: s - сдвиг пера вправо на один пиксел, d – сдвиг пера вправо и вверх (по диагонали) на один пиксел. Алгоритм Кастла-Питвея позволяет строить отрезок, оперируя двумя этими командами. Этот алгоритм гораздо медленнее алгоритма Брезенхема, но представляет интерес с чисто алгоритмической точки зрения.

Введём две команды для работы со строками:

- $\oplus$ - конкатенация (объединение) строк. Например, $dss \oplus ssd = dssssd$;
- $\sim$ - переворот строки. Например, $\sim dssds = sdssd$.

Тогда алгоритм растеризации отрезка из точки $(0; 0)$ в точку $(a; b)$ будет выглядеть следующим образом:

```
y:=b;
x:=a-b;

m1:="s";
m2:="d";
while x<>y do
begin
  if x>y then
  begin
```

```
   x:=x-y;
   m2:=m1⊕~m2;
  end
  else
  begin
   y:=y-x;
   m1:=m2⊕~m1;
  end;
end;
```

После завершения работы алгоритма нужная последовательность сдвигов определяется, как $m = m2 \oplus \sim m1$.

Алгоритм Ву (растеризация отрезков со сглаживанием)

Зачастую линии, построенные с помощью алгоритма Брезенхема, выглядят неестественно. Для того, чтобы линии выглядели более естественно, предложены алгоритмы со сглаживанием. Один из них – алгоритм Ву. Этот алгоритм обеспечивает хорошее сглаживание и достаточно высокую скорость.

Необходимости в сглаживании при отрисовке вертикальных и горизонтальных линий нет. Алгоритм Ву применяется исключительно для наклонных линий. Координаты по неосновной оси рассчитываются методом, подобным алгоритму Брезенхема. Отличие состоит в том, что в алгоритме Ву на каждом шаге высвечивается не один, а два пиксела. Интенсивность этих пикселов обратно пропорциональна их расстоянию от идеальной линии, но в сумме эти интенсивности дают 1 (т.е. яркость пиксела на идеальной линии). Это придаёт линии одинаковую интенсивность на всём её протяжении.

Рисунок 2 Линия, построенная алгоритмом Ву

Алгоритмы отрисовки окружностей

Растеризация окружности является гораздо более сложной задачей, чем растеризация отрезка. Но стоит помнить, что окружность обладает осевой симметрией, так что можно строить только малую её часть, а для построения цельной окружности воспользоваться операциями поворота и отражения.

Алгоритмы построения на основе четырёхсторонней симметрии

Окружность с центром в начале координат представляется следующим уравнением: $x^2 + y^2 = R^2$. Получить окружность можно следующим образом: сначала строим $\frac{1}{4}$ окружности (т.е. полностью

строим один квадрант), а остальные части получаем отражением и переносом. Для построения четверти окружности можно изменять x от 0 до R, при этом на каждом шаге необходимо вычислять значение y.

Этот метод крайне неэффективен, так как содержит не только операции умножения, но и операции извлечения корня. Кроме того, при x близких к R в дуге будут появляться незаполненные участки. Т.е. дуга в нижней части получится более разреженной.

От разреженности можно избавиться, рассчитывая окружность, используя уравнения в полярных координатах. Тогда точки окружности вычислялись бы как $(R * cos(\alpha); R * sin(\alpha))$, путём пошагового изменения α от $0°$ до $90°$. Этот метод позволит улучшить качество изображения, но время вычисления останется непропорционально большим.

Алгоритм Брезенхема для окружностей

Брезенхем предложил эффективный алгоритм генерации дуг. Для построения окружности достаточно построить $\frac{1}{8}$ её часть (второй октант), остальные части можно получить с помощью соответсвующих матриц отражения:

- Первый октант – отражение относительно оси $y = x$ второго октанта. Матрица отражения: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$;
- Второй квадрант – отражение относительно оси $x = 0$ первого квадранта. Матрица отражения $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$;
- Нижняя полуокружность – отражение относительно оси верхней полуокружности $y = 0$. Матрица отражения: $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

Построение окружности начинается в точке $(0; R)$. Окружность строится по часовой стрелке, при этом x изменяется в диапазоне от 0 до $\frac{R}{\sqrt{2}}$.

Для заданной точки на такой дуге существует только две возможности выбрать следующий пиксел, наилучшим образом приближающий окружность (либо пиксел с координатами $(x + 1; y)$ или $(x + 1; y - 1)$). Алгоритм выбирает пиксел, для которого минимален квадрат расстояния между ним и окружностью.

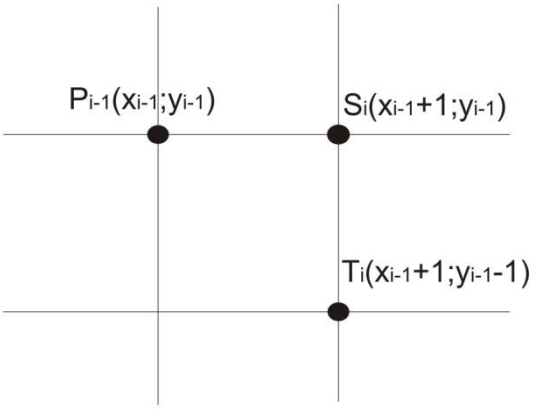

Рисунок 3 Возможные направления для построения окружности

Расстояния вычисляются по следующим формулам:

$$D(S_i) = (x_{i-1} + 1)^2 + y_{i-1}^2 - R^2$$

$$D(T_i) = (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - R^2$$

Введём дополнительную переменную $d_i = |D(Si)| - |D(T_i)|$. Если $d_i \geq 0$ выбирается пиксел T_i, в противном случае выбирается пиксел S_i.

Идеальная окружность может располагаться относительно пикселов S_i, T_i одним из 5 возможных способов:

1. Над пикселом S_i;
2. Через пиксел S_i;
3. Под пикселом S_i и над пикселом T_i;
4. Через пиксел T_i;
5. Под пикселом T_i;

Для случаев 1,2: $D(T_i) < 0; D(S_i) \leq 0; D(S_i) < D(T_i)$. Так как $d_i < 0$, выбирается пиксел S_i.

Для случая 3: $D(T_i) < 0; D(S_i) > 0$. В зависимости от знака d_i выбирается либо пиксел T_i, либо S_i.

Для случаев 4,5: $D(T_i) \geq 0; D(S_i) > 0; D(S_i) > D(T_i)$. Так как $d_i \geq 0$ выбирается пиксел T_i.

С помощью некоторых преобразований можно получить, что $d_1 = 3 - 2R$.

$\boldsymbol{d_i < 0}$	$\boldsymbol{d_i \geq 0}$
Выбирается точка S_i $x_i = x_{i-1} + 1$ $y_i = y_{i-1}$ $d_{i+1} = d_i + 4 * x_{i-1} + 6$	Выбирается точка T_i $x_i = x_{i-1} + 1$ $y_i = y_{i-1} - 1$ $d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10$

Таким итеративным способом строится восьмая часть окружности. Все остальные части получаются с помощью описанных выше матриц отражения.

Отрисовка эллипсов

Эллипс представляет собой деформированную окружность. Он имеет два фокуса, и сумма расстояний от любой точки эллипса до фокусов – постоянная величина. Можно рассматривать окружность, как частный случай эллипса, когда фокусы совпадают. В каноническом виде уравнение эллипса имеет вид $\frac{X^2}{A^2} + \frac{Y^2}{B^2} = 1$, где A, B – радиусы эллипса. Но это уравнение можно переписать в виде $B^2x^2 + A^2y^2 - A^2B^2 = 0$. Тогда это уравнение можно будет использовать для сравнения двух возможных пикселов на очередном шаге алгоритма, и выбирать тот пиксел, который внесёт в изображение эллипса наименьшие искажения. Важно помнить, что эллипс обладает только двумя осями симметрии, а значит, для его построения нужно построить целый квадрант (остальные три квадранта получаются отражением относительно осей).