



ОГЛАВЛЕНИЕ

Предисловие редакторов перевода	5
Предисловие к русскому изданию	7
Предисловие	9

ГЛАВА 1. ВВЕДЕНИЕ В МАШИННУЮ ГРАФИКУ

13

✓ 1.1. Обзор машинной графики	13
1.2. Типы графических устройств	15
1.3. Графические дисплеи на запоминающей трубке	16
1.4. Векторные графические дисплеи с регенерацией изображения	18
1.5. Растровые графические дисплеи с регенерацией изображения	23
1.6. Устройство электронно-лучевой трубки	30
1.7. Устройство цветной растровой ЭЛТ	31
1.8. Системы с телевизионным растром	33
1.9. Диалоговые устройства	36
1.10. Резюме	46
1.11. Литература	46

ГЛАВА 2. РАСТРОВАЯ ГРАФИКА

48

✓ 2.1. Алгоритмы вычерчивания отрезков	48
✓ 2.2. Цифровой дифференциальный анализатор	50
✓ 2.3. Алгоритм Брезенхема	54
✓ 2.4. Целочисленный алгоритм Брезенхема	59
✓ 2.5. Общий алгоритм Брезенхема	60
✓ 2.6. Алгоритм Брезенхема для генерации окружности	63
✓ 2.7. Растровая развертка — способ генерация изображения	73
✓ 2.8. Растровая развертка в реальном времени	73
2.9. Групповое кодирование	80
2.10. Клеточное кодирование	83
✓ 2.11. Буферы кадра	85
✓ 2.12. Адресация раstra	87
✓ 2.13. Изображение отрезков	89
✓ 2.14. Изображение литер	91
✓ 2.15. Растровая развертка сплошных областей	92
✓ 2.16. Заполнение многоугольников	93
✓ 2.17. Растровая развертка многоугольников	94
✓ 2.18. Простой алгоритм с упорядоченным списком ребер	97

✓ 2.19. Более эффективные алгоритмы с упорядоченным списком ребер	99
✓ 2.20. Алгоритм заполнения по ребрам	105
2.21. Алгоритм со списком ребер и флагом	107
✓ 2.22. Алгоритмы заполнения с затравкой	110
✓ 2.23. Простой алгоритм заполнения с затравкой	111
✓ 2.24. Построчный алгоритм заполнения с затравкой	114
2.25. Основы методов устранения ступенчатости	119
2.26. Простой метод устранения лестничного эффекта	123
2.27. Свертка и устранение ступенчатости	127
2.28. Аппроксимация полутонами	131
2.29. Литература	139

ГЛАВА 3. ОТСЕЧЕНИЕ

✓ 3.1. Двумерное отсечение	143
✓ 3.2. Алгоритм отсечения Сазерленда — Козна, основанный на разбиении отрезка	153
✓ 3.3. Алгоритм разбиения средней точкой	158
3.4. Обобщение: отсечение двумерного отрезка выпуклым окном	166
? 3.5. Алгоритм Кируса — Бека	170
✓ 3.6. Внутреннее и внешнее отсечение	181
3.7. Определение факта выпуклости многоугольника и вычисление его внутренних нормалей	182
3.8. Разбиение невыпуклых многоугольников	187
3.9. Трехмерное отсечение	188
3.10. Трехмерный алгоритм разбиения средней точкой	192
? 3.11. Трехмерный алгоритм Кируса — Бека	194
3.12. Отсечение в однородных координатах	198
3.13. Определение выпуклости трехмерного тела и вычисление внутренних нормалей к его граням	201
3.14. Разрезание невыпуклых тел	203
✓ 3.15. Отсечение многоугольников	206
3.16. Последовательное отсечение многоугольника — алгоритм Сазерленда — Ходжмана	207
3.17. Невыпуклые отсекающие области — алгоритм Вейлера — Азертонна	220
3.18. Отсечение литер	227
3.19. Литература	228

ГЛАВА 4. УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ

4.1. Введение	230
4.2. Алгоритм плавающего горизонта	233
4.3. Алгоритм Робертса	250
4.4. Алгоритм Варнока	290
4.5. Алгоритм Вейлера — Азертонна	315
4.6. Алгоритм разбиения криволинейных поверхностей	320
4.7. Алгоритм, использующий z-буфер	321
4.8. Алгоритмы, использующие список приоритетов	329

Введение в машинную графику

Современная машинная графика — это тщательно разработанная дисциплина. Обстоятельно исследованы [1-1 — 1-3] основные элементы геометрических преобразований и описаний кривых и поверхностей. Также изучены, но все еще продолжают развиваться методы растрового сканирования, отсечение, удаление невидимых линий и поверхностей, цвет, закрашка, текстура и эффекты прозрачности. Сейчас наибольший интерес представляют именно эти разделы машинной графики.

1.1. ОБЗОР МАШИННОЙ ГРАФИКИ

Машинная графика — сложная и разнообразная дисциплина. Для изучения ее прежде всего необходимо разбить на обозримые части, приняв во внимание, что конечным продуктом машинной графики является изображение. Это изображение может, разумеется, использоваться для разнообразных целей. Например, оно может быть техническим чертежом, иллюстрацией с изображением деталей в разобранном виде в руководстве по эксплуатации, деловой диаграммой, архитектурным видом предлагаемой конструкции или проектируемого здания, рекламной иллюстрацией или кадром из мультфильма. В машинной графике фундаментальным связующим звеном является изображение; следовательно, мы должны рассмотреть, как

изображения представляются в машинной графике;
изображения готовятся для визуализации;
предварительно подготовленные изображения рисуются;
осуществляется взаимодействие с изображением.

Хотя во многих алгоритмах в качестве геометрических данных, описывающих изображения, выступают многоугольники и ребра, каждый многоугольник или ребро в свою очередь может быть представлен своими вершинами. Таким образом, точки являются фундаментальными строительными блоками для представления геометрических данных. Не меньшего внимания заслуживает алгоритм, показывающий, как организовать точки. В качестве иллюстрации рассмотрим единичный квадрат в первом квадранте. Он может быть представлен своими вершинами (рис. 1.1):

$$P_1(0, 0), P_2(1, 0), P_3(1, 1), P_4(0, 1)$$

Соответствующее алгоритмическое описание может выглядеть так:

Соединить последовательно $P_1P_2P_3P_4P_1$

Единичный квадрат также можно описать с помощью четырех ребер:

$$E_1 \equiv P_1P_2, E_2 \equiv P_2P_3, E_3 \equiv P_3P_4, E_4 \equiv P_4P_1$$

Здесь алгоритмическое описание таково:

Изобразить последовательно ребра $E_1E_2E_3E_4$

И наконец, для описания единичного квадрата как многоугольника можно использовать либо точки, либо ребра. Например,

$$S_1 = P_1P_2P_3P_4P_1 \text{ или } P_1P_4P_3P_2P_1 \\ \text{или } S_1 = E_1E_2E_3E_4$$

Основные строительные блоки (точки) в зависимости от размерности пространства можно представлять либо как пары, либо как тройки чисел. Таким образом, (x_1, y_1) или (x_1, y_1, z_1) представили

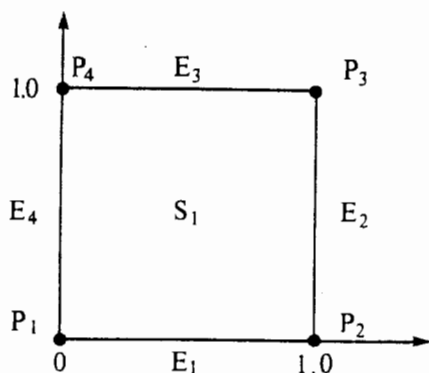


Рис. 1.1. Описания данных изображения.

бы точку в двух- или трехмерном пространстве. Две точки представляли бы отрезок, или ребро, а совокупность из трех или более точек — многоугольник. Эти точки, ребра, многоугольники накапливаются, или хранятся, в базе данных. Данные, из которых получают рисунок, редко совпадают с данными, служащими непосредственно для рисования. Данные, используемые для вывода изображения, часто называют дисплейным файлом. В нем содержится некоторая часть, вид или сцена изображения, представленного в общей базе данных. Выводимое изображение обычно формируется с помощью операций поворота, переноса, масштабирования и вычисления различных проекций данных. Как правило, эти основные видовые преобразования выполняются с помощью матричных (4×4) операций над данными, представленными в однородных координатах [1-1]. Эти операции часто реализуются аппаратно. Прежде чем рисовать окончательный результат, можно добавить удаление невидимых линий или поверхностей, произвести закраску, учесть влияние прозрачности, нанести текстуру и воспроизвести цветовые эффекты. Если не надо рисовать изображение, представленное во всей базе данных, то следует выбрать соответствующую его часть. Данный процесс называется отсечением. Отсечение может быть двух- или трехмерным. В некоторых случаях отсекающее окно или объем могут быть с дырами или иметь нерегулярную форму. Отсечение относительно стандартных областей часто реализуется аппаратно.

Почти все изображения содержат текстовую информацию. Литеры могут генерироваться либо аппаратным, либо программным образом, в последнем случае с ними можно обращаться как с любой другой частью изображения. При аппаратной генерации литеры сохраняются в виде кодов до момента непосредственного вывода. Для преобразования литер обычно предоставляются только ограниченные возможности, например некий набор углов поворота и коэффициентов масштабирования. Как правило, отсечение аппаратно генерируемых литер невозможно: либо изображается вся литера, либо она не изображается вообще.

1.2. ТИПЫ ГРАФИЧЕСКИХ УСТРОЙСТВ

Существует много разнообразных устройств для вывода изображений, построенных с помощью машинной графики. В качестве типичных примеров назовем перьевые графопостроители, точечно-матричные, электростатические и лазерные печатающие устройства, фильмирующие устройства, дисплеи на запоминающей трубке,

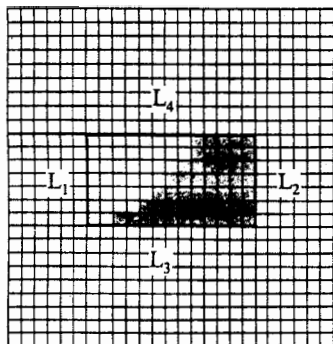


Рис. 1.15. Сплошные фигуры на растровом графическом устройстве.

на работа в реальном времени для буферов кадров размером 512×512 и 1024×1024 .

Хотя производительности, необходимой для работы в реальном масштабе времени с приемлемыми скоростями регенерации, на растровых устройствах достичь труднее, чем на векторных дисплеях с регенерацией, на них легче изображать сплошные фигуры с плавными переходами цветов. Как показано на рис. 1.15, растровое представление сплошной «полигональной» фигуры концептуально просто. Здесь представление сплошной фигуры, ограниченной отрезками L_1 , L_2 , L_3 , L_4 , достигается установкой всех пикселей внутри ограничивающего многоугольника в соответствующий цвет в буфере кадра. Алгоритмы «растровой развертки» сплошной области обсуждаются в гл. 2.

1.6. УСТРОЙСТВО ЭЛЕКТРОННО-ЛУЧЕВОЙ ТРУБКИ

Описанный выше буфер кадра сам по себе не является устройством вывода, он просто применяется для хранения рисунка. Наиболее часто в качестве устройства отображения, используемого с буфером кадра, выступает видеомонитор. Чтобы понять принципы работы растровых дисплеев и в некоторой степени векторных дисплеев с регенерацией, нужно иметь представление о конструкции ЭЛТ и методах создания видеоизображения.

На рис. 1.16 схематично показана ЭЛТ, используемая в видеомониторах. Катод (отрицательно заряженный) нагревают до тех пор, пока возбужденные электроны не создадут расширяющегося облака (электроны отталкиваются друг от друга, так как имеют одинаковый заряд). Эти электроны притягиваются к сильно заряженному положительному аноду. На внутреннюю сторону расширенного конца ЭЛТ нанесен люминофор. Если бы электронам ни-

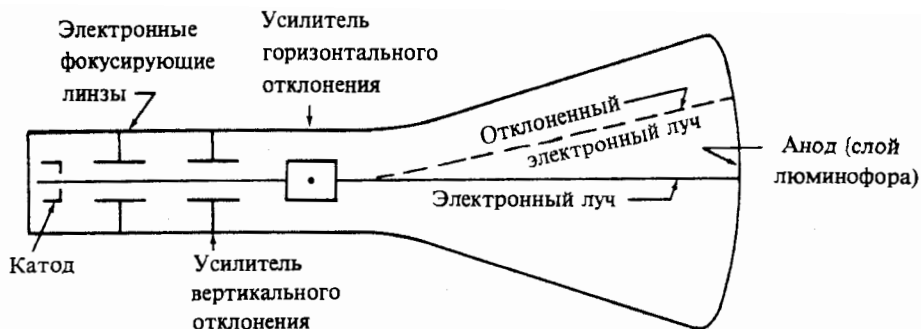


Рис. 1.16. Электронно-лучевая трубка.

что не препятствовало, то в результате их воздействия на люминофор весь экран ЭЛТ засветился бы ярким светом. Однако облако электронов с помощью электронных линз фокусируется в узкий, строго параллельный пучок. Теперь сфокусированный электронный луч дает одно яркое пятно в центре ЭЛТ. Луч отклоняется или позиционируется влево или вправо от центра и(или) выше или ниже центра с помощью усилителей горизонтального и вертикального отклонения.

Именно в данный момент проявляется отличие векторных дисплеев — как с запоминанием, так и с регенерацией — от растровых. В векторном дисплее электронный луч может быть отклонен непосредственно из любой произвольной позиции в любую другую произвольную позицию на экране ЭЛТ (аноде). Поскольку люминофорное покрытие нанесено на экран ЭЛТ сплошным слоем, в результате получается почти идеальная прямая. В отличие от этого в растровом дисплее луч может отклоняться только в строго определенные позиции на экране, образуя своеобразную мозаику. Эта мозаика составляет видеонизображение. Люминофорное покрытие на экране растровой ЭЛТ тоже не непрерывно, а представляет собой множество тесно расположенных мельчайших точек, куда может позиционироваться луч, образуя мозаику.

1.7. УСТРОЙСТВО ЦВЕТНОЙ РАСТРОВОЙ ЭЛТ

Цветная растровая ЭЛТ похожа на стандартную черно-белую ЭЛТ, описанную в предыдущем разделе. В ней находятся три электронные пушки, по одной на каждый основной цвет: красный, зеленый и синий. Электронные пушки часто объединены в треугольный блок, соответствующий подобному треугольному блоку точек крас-

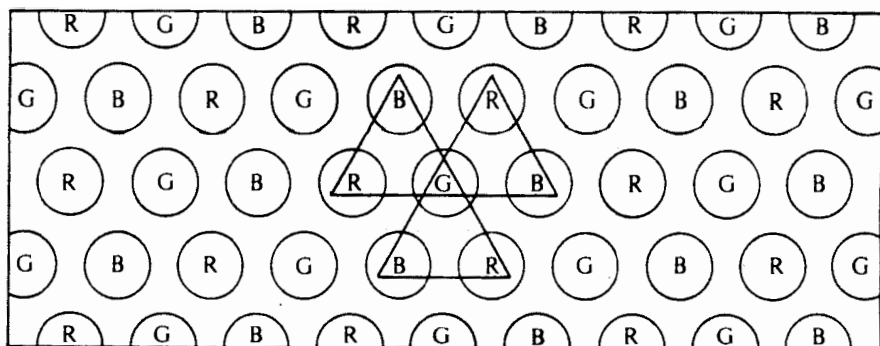


Рис. 1.17. Точечный люминофорный растр для ЭЛТ с теневой маской.

ного, зеленого и синего люминофоров на экране ЭЛТ (рис. 1.17)¹⁾. Для того чтобы электронные пушки возбуждали только соответствующие им точки люминофора (например, красная пушка возбуждала только точку красного люминофора), между электронными пушками и поверхностью экрана помещена перфорированная металлическая решетка. Это так называемая теневая маска стандартной цветной ЭЛТ с теневой маской. Отверстия в ней образуют такие же треугольные блоки, как и точки люминофора. Расстояния между отверстиями называются шагом. Цветовые пушки расположены таким образом, что их лучи сходятся и пересекаются в плоскости теневой маски (рис. 1.18). После прохождения через отверстие красный луч, например, защищен или маскирован от пересечения с зеленой или синей точкой люминофора. Он может пересечь

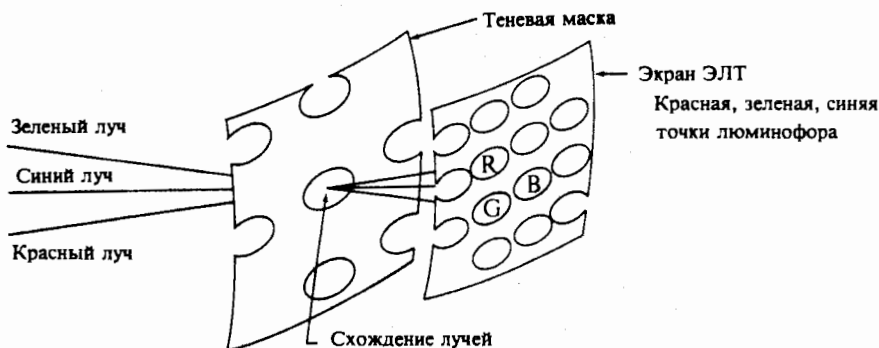


Рис. 1.18. Расположение электронной пушки и теневой маски цветной ЭЛТ.

¹⁾ На этом и следующих рисунках R — сокращение от red (красный), G — от green (зеленый), B — от blue (синий). — *Прим. ред.*

лишь красную точку. Изменяя интенсивность электронного луча для каждого основного цвета, можно получить различные оттенки. Комбинация этих оттенков дает большое количество цветов для каждого пиксела. Обычно в дисплее с высоким разрешением на каждый пиксел приходится от двух до трех цветовых триад.

1.8. СИСТЕМЫ С ТЕЛЕВИЗИОННЫМ РАСТРОМ

Процесс преобразования хранящейся в буфере кадра растровой картинки в упорядоченный набор точек на телеэкране называется растровой разверткой. Сканируемый набор точек и частота воспроизведения основаны как на особенностях визуального восприятия, так и на принципах электроники. Системе визуального восприятия человека требуется конечный интервал времени для рассмотрения элементов картины. Однако обеспечить впечатление непрерывности позволит только такой интервал, который настолько мал, что инерция зрительного восприятия перекрывает мерцание. На мерцание влияет ряд факторов, включая яркость изображения и конкретный люминофор, используемый для экрана ЭЛТ. Опыт показывает, что для практических целей минимальной скоростью вывода или изменения изображения является 25 кадров в секунду при условии, что минимальная скорость регенерации или воспроизведения в два раза больше, т. е. 50 кадр/с. Аналогичная ситуация имеет место при демонстрации кинофильма. При этом показывается 24 кадр/с, но каждый кадр показывается дважды, и в результате получается эффективная скорость воспроизведения 48 кадр/с. Таким образом, для фильма скорость изменения равна 24, а скорость регенерации — 48. В телевидении тот же самый эффект достигается с помощью метода, называемого чересстрочной разверткой.

Телевидение использует метод растрового сканирования. В американской стандартной видеосистеме используется в совокупности 525 горизонтальных строк с кадровым, или видовым, отношением 4:3, т. е. высота видовой области равна трем четвертям ее ширины. Скорость воспроизведения, или смены кадра, составляет 30 кадр/с. Однако каждый кадр делится на два поля, каждое из которых содержит по половине картинки. Эти два поля чередуются или перемежаются. Они попеременно показываются через каждые $1/60$ с. Одно поле содержит все строки с нечетными номерами (1, 3, 5,...), а другое — с четными (2, 4, 6,...). Сканирование начинается в верхнем левом углу экрана с нечетного поля. Каждая строка в поле сканируется или представляется слева направо. В то время, как электронный луч движется поперек экрана слева направо, он

также перемещается вертикально вниз, но с намного меньшей скоростью. Таким образом, «горизонтальная» сканирующая строка на самом деле слегка наклонена. При достижении правого края экрана луч делают невидимым и быстро возвращают к левому краю. Такой горизонтальный возврат луча обычно занимает около 17% времени, отведенного для одной сканирующей строки. Затем этот процесс повторяется со следующей нечетной строкой. Так как половина от 525 равна $262 \frac{1}{2}$ строки, то при завершении сканирования поля нечетных строк луч окажется в центре нижней строки экрана (рис. 1.19 и 1.20). Луч затем быстро переводится в центр верхней стороны экрана. Так производится вертикальный перевод луча для нечетного поля. Время, затрачиваемое на него, эквивалентно времени, затрачиваемому на сканирование 21 строки. Затем показывается поле четных строк, после чего луч оказывается в нижнем правом углу. Перевод луча для поля четных строк возвращает его в верхний левый угол, и весь процесс повторяется снова. Таким образом, показываются два поля для каждого кадра, т. е. 60 полей в секунду. Данный метод существенно уменьшает мерцание, так как глаз воспринимает скорость воспроизведения поля.

Хотя в принятой в США стандартной видеосистеме предусмотрено 525 строк, на самом деле видимы только 483 строки, так как время, затрачиваемое на сканирование 21 строки, уходит на вертикальный перевод луча¹⁾. В течение этого времени электронный луч

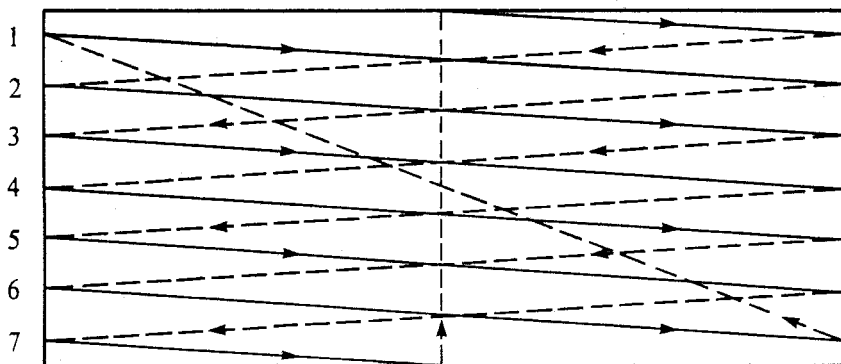


Рис. 1.19. Схема чересстрочной развертки для раstra из семи строк. Нечетное поле начинается со строки 1. Пунктиром обозначен горизонтальный обратный ход луча. Вертикальный обратный ход луча для нечетного поля начинается внизу в центре, а для четного поля — внизу справа.

¹⁾ Во многих растровых графических устройствах это время используется для обработки другой информации.



Рис. 1.20. Схема 525-строчного стандартного кадра.

невидим или выключен. Время, отпущенное на каждую сканирующую строку, легко подсчитывается для частоты воспроизведения кадра 30/с следующим образом:

$$\frac{1}{30} \frac{\text{с}}{\text{кадр}} \times \frac{1}{525} \frac{\text{кадр}}{\text{строка}} = 63.5 \frac{\text{мкс}}{\text{строка}}$$

Так как приблизительно $10 \frac{1}{2}$ мкс тратится на горизонтальный перевод луча, то вывод видимой части каждой строки должен быть завершен за 53 мкс. В строке при нормальном соотношении сторон видеообласти, равном 4:3, находится 644 пиксела. Таким образом, время, отпущенное на считывание и вывод пиксела, равно

$$53 \frac{\text{мкс}}{\text{строка}} \times \frac{1}{644} \frac{\text{строка}}{\text{пиксел}} = 82 \text{ нс}$$

Во многих растровых дисплеях, построенных на основе буфера кадра, применяется разрешение изображения 512 пикселей в строке. На считывание и вывод пиксела при таком разрешении отводится приблизительно 103 наносекунды. Аналогичные результаты получаются для 625-строчной видеосистемы с частотой воспроизведения 25 кадр/с, используемой в Великобритании и большинстве стран Европы.

Метод чересстрочной развертки не является абсолютно необходимым при выводе видеоизображения, однако изображение без чередования строк будет несовместимым со стандартным телевизионным приемником. При отсутствии чересстрочной развертки для устранения мерцания придется увеличить частоту воспроизведения до 60 кадр/с, а это, конечно, сокращает в 2 раза время для обработки пиксела. Более высокое разрешение по числу строк и количе-

ству пикселей в строке также уменьшает это время; например, при разрешении 1024×1024 на считывание и вывод пиксела отводится в 4 раза меньше времени, чем при разрешении 512×512 , — приблизительно 25 нс! В этом случае потребуются очень быстрая память для буфера кадра и такой же быстрый ЦАП.

1.9. ДИАЛоговые УСТРОЙСТВА

После того как изображение построено на экране, необходимо как-то взаимодействовать с ним или модифицировать его. Для удовлетворения этой потребности был разработан ряд диалоговых устройств. Среди них можно назвать планшет, световое перо, рычаг, мышь, ручки для ввода скалярных величин, функциональные переключатели или кнопки и, разумеется, обычную алфавитно-цифровую клавиатуру. Прежде чем перейти к обсуждению этих физических устройств, рассмотрим функциональные возможности диалоговых графических устройств. Обычно насчитывают четыре или пять таких классов [1-3 — 1-6]. Логическими диалоговыми устройствами являются локатор, валуатор, селектор и кнопка. Из-за широкой распространенности алфавитно-цифровой клавиатуры ее часто выделяют в пятый класс, называемый клавиатурой. На самом деле клавиатуру можно концептуально и функционально считать набором кнопок.

Функцией локатора является выдача координатной информации в двух или трех измерениях. Обычно выдаваемые значения координат находятся в пространстве устройства (концептуальном пространстве) и могут быть как относительными, так и абсолютными. Валуатор применяется для ввода одиночной величины. Обычно это вещественное число между нулем и некоторым вещественным максимумом. Кнопка используется для выбора и активирования событий или процедур, управляющих ходом диалога. Кнопка обычно предоставляет двоичную («включено» или «выключено») цифровую информацию. В функцию селектора входит идентификация или выбор объектов или подкартинок в выведенном изображении. Логическая клавиатура обрабатывает текстовую информацию. На рис. 1.21 показана типичная клавиатура.

Наиболее общим устройством класса локаторов является планшет (рис. 1.22). Планшеты можно использовать либо отдельно, либо в комбинации с графическим дисплеем на ЭЛТ. В первом случае их часто называют оцифровывателями. Сам по себе планшет состоит из плоской поверхности и карандаша (похожего на

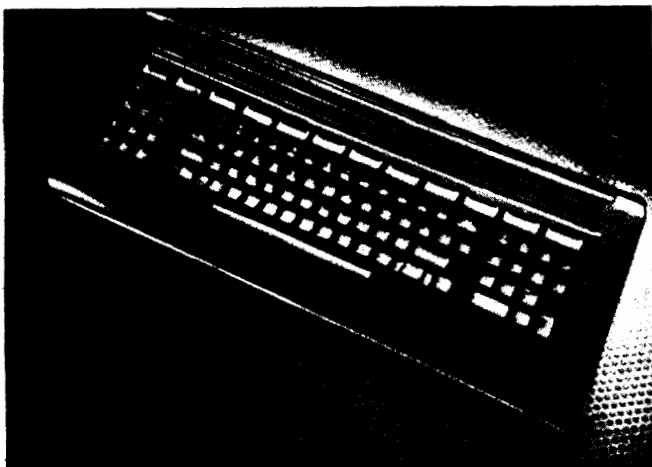


Рис. 1.21. Алфавитно-цифровая клавиатура. (С разрешения Evans & Sutherland Computer Corp.)

обычный карандаш), который служит для указания точки на поверхности планшета. Часто существует возможность узнать некоторую информацию о расстоянии между карандашом и планшетом: близко или далеко от поверхности находится карандаш. При использовании вместе с дисплеем на ЭЛТ обратная связь на экране обеспечивается с помощью небольшого символа (курсора), отслеживающего перемещение карандаша по поверхности планшета.

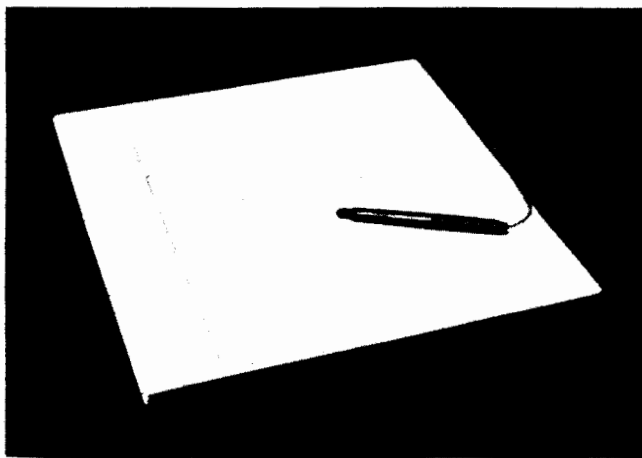


Рис. 1.22. Типичный планшет. (С разрешения фирмы Adage)

Растровая графика

Для работы с устройствами растровой графики нужны специальные методы генерации изображения, вычерчивания прямых и кривых линий, закраски многоугольников, создающей впечатление сплошных объектов. Эти методы рассматриваются в данной главе.

2.1. АЛГОРИТМЫ ВЫЧЕРЧИВАНИЯ ОТРЕЗКОВ

Поскольку экран растрового дисплея с электронно-лучевой трубкой (ЭЛТ) 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. 2.1.

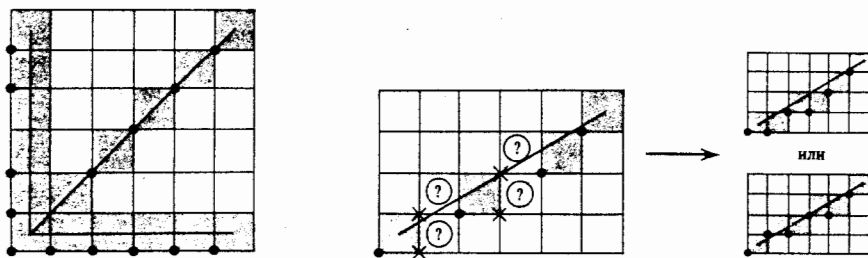


Рис. 2.1. Разложение в растр отрезков прямых.

Прежде чем приступить к обсуждению конкретных алгоритмов рисования отрезков, полезно рассмотреть общие требования к таким алгоритмам и ответить на вопрос, каковы желаемые характеристики изображения. Очевидно, что отрезки должны выглядеть прямыми, начинаться и заканчиваться в заданных точках. Далее, яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона. Наконец, рисовать нужно быстро. Как это часто бывает, не все из перечисленных критериев могут быть полностью удовлетворены. Сама природа растрового дисплея исключает генерацию абсолютно прямых линий (кроме ряда специальных случаев), равно как и точное совпадение начала и конца отрезка с заданными точками. Тем не менее при достаточно высоком разрешении дисплея можно получить приемлемую аппроксимацию.

Постоянная вдоль всего отрезка яркость достигается лишь при проведении горизонтальных, вертикальных и наклоненных под углом 45° прямых. Для всех других ориентаций разложение в растр приведет к неравномерной яркости, как это показано на рис. 2.1. Даже для частных случаев яркость зависит от наклона: заметим, например, что расстояние между соседними пикселями для отрезка под углом 45° больше, чем для вертикальных и горизонтальных прямых. Поэтому вертикальные и горизонтальные отрезки будут выглядеть ярче, чем наклонные. Обеспечение одинаковой яркости вдоль отрезков разных длин и ориентаций требует извлечения квадратного корня, а это замедлит вычисления. Обычным компромиссом является нахождение приближенной длины отрезка, сведение вычислений к минимуму, предпочтительное использование целой арифметики, а также реализация алгоритмов на аппаратном или микропрограммном уровне.

В большинстве алгоритмов вычерчивания отрезков для упрощения вычислений используется пошаговый метод. Приведем пример подобного алгоритма:

Простой пошаговый алгоритм

позиция = начало

шаг = приращение

- 1 If позиция — конец < точность then 4
If позиция > конец then 2
If позиция < конец then 3
- 2 позиция = позиция — шаг
go to 1

- 3 позиция = позиция + шаг
 go to 1
 4 finish

Простой алгоритм разложения отрезка в растр, описанный в следующем разделе, иллюстрирует применение пошаговых методов.

2.2. ЦИФРОВОЙ ДИФФЕРЕНЦИАЛЬНЫЙ АНАЛИЗАТОР

Один из методов разложения отрезка в растр состоит в решении дифференциального уравнения, описывающего этот процесс. Для прямой линии имеем

$$\frac{dy}{dx} = \text{const} \quad \text{или} \quad \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Решение представляется в виде

$$\begin{aligned} y_{i+1} &= y_i + \Delta y \\ y_{i+1} &= y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x \end{aligned} \quad (2.1)$$

где x_1, y_1 и x_2, y_2 — концы разлагаемого отрезка и y_i — начальное значение для очередного шага вдоль отрезка. Фактически уравнение (2.1) представляет собой рекуррентное соотношение для последовательных значений y вдоль нужного отрезка. Этот метод, используемый для разложения в растр отрезков, называется цифровым дифференциальным анализатором (ЦДА). В простом ЦДА либо Δx , либо Δy (большее из приращений) выбирается в качестве единицы растра. Ниже приводится простой алгоритм, работающий во всех квадрантах:

Процедура разложения в растр отрезка по методу цифрового дифференциального анализатора (ЦДА)

предполагается, что концы отрезка (x_1, y_1) и (x_2, y_2) не совпадают

Integer — функция преобразования вещественного числа в целое.

Примечание: во многих реализациях функция Integer означает взятие целой части, т. е. $\text{Integer}(-8.5) = -9$, а не -8 . В алгоритме используется именно такая функция.

Sign — функция, возвращающая $-1, 0, 1$ для отрицательного, нулевого и положительного аргумента соответственно

аппроксимируем длину отрезка

if $\text{abs}(x_2 - x_1) \geq \text{abs}(y_2 - y_1)$ **then**

 Длина = $\text{abs}(x_2 - x_1)$

else

 Длина = $\text{abs}(y_2 - y_1)$

end if

полагаем большее из приращений Δx или Δy равным единице растра

$\Delta x = (x_2 - x_1) / \text{Длина}$

$\Delta y = (y_2 - y_1) / \text{Длина}$

округляем величины, а не отбрасываем дробную часть

использование знаковой функции делает алгоритм пригодным для всех квадрантов

$x = x_1 + 0.5 \cdot \text{Sign}(\Delta x)$

$y = y_1 + 0.5 \cdot \text{Sign}(\Delta y)$

начало основного цикла

$i = 1$

while ($i \leq \text{Длина}$)

Plot (**Integer**(x), **Integer**(y))

$x = x + \Delta x$

$y = y + \Delta y$

$i = i + 1$

end while

finish

Приведем пример, иллюстрирующий работу алгоритма:

Пример 2.1. Простой ЦДА в первом квадранте

Рассмотрим отрезок из точки (0, 0) в точку (5, 5). Используем ЦДА для разложения этого отрезка в растр. Результаты работы алгоритма таковы:

начальные установки

$x_1 = 0$

$y_1 = 0$

$x_2 = 5$

$y_2 = 5$

Длина = 5

$\Delta x = 1$

$\Delta y = 1$

$x = 0.5$

$y = 0.5$

результаты пошагового выполнения основного цикла

i	Plot	x	y
		0.5	0.5
1	(0, 0)		
		1.5	1.5
2	(1, 1)		
		2.5	2.5
3	(2, 2)		
		3.5	3.5
4	(3, 3)		
		4.5	4.5
5	(4, 4)		
		5.5	5.5

Полученное растровое представление отрезка изображено на рис. 2.2. Заметим, что концевые точки определены точно и выбранные пиксели равномерно распределены вдоль отрезка. Внешний вид прямой вполне удовлетворителен. Однако если начальным значением переменной i сделать нуль вместо единицы, то окажется активированным пиксел с координатами (5, 5), что нежелательно. Если адрес пикселя задан целыми координатами левого нижнего угла, то активация этого пикселя даст явно неверную конечную точку отрезка (рис. 2.2). Вдобавок при вычерчивании серии последовательных отрезков пиксел (5, 5) будет активирован дважды: в конце данного отрезка и в начале следующего. Такой пиксел может выглядеть как более яркий или иметь иной (быть может, неестественный) цвет. Следующий пример иллюстрирует работу алгоритма в третьем квадранте.

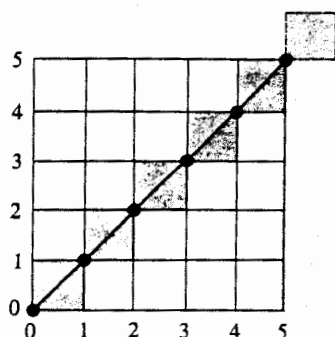


Рис. 2.2. Результаты работы простого ЦДА в первом квадранте.

Пример 2.2. Простой ЦДА в третьем квадранте

Рассмотрим отрезок из точки (0, 0) в точку (−8, −4) в третьем квадранте. Результаты работы алгоритма таковы:

начальные установки

$$\begin{aligned}x_1 &= 0 \\y_1 &= 0 \\x_2 &= -8 \\y_2 &= -4\end{aligned}$$

Длина = 8

$$\begin{aligned}\Delta x &= -1 \\ \Delta y &= -0.5 \\ x &= -0.5 \\ y &= -0.5\end{aligned}$$

результаты пошагового выполнения основного цикла в предположении, что используется функция округления, таковы:

i	Plot	x	y
		−0.5	−0.5
1	(−1, −1)	−1.5	−1.0
2	(−2, −1)	−2.5	−1.5
3	(−3, −2)	−3.5	−2.0
4	(−4, −2)	−4.5	−2.5
5	(−5, −3)	−5.5	−3.0
6	(−6, −3)	−6.5	−3.5
7	(−7, −4)	−7.5	−4.0
8	(−8, −4)	−8.5	−4.5

Несмотря на то что результаты, представленные на рис. 2.3, выглядят вполне приемлемыми, анализ отрезков, проведенных из точки (0, 0) в точку (−8, 4) и (8, −4), показывает, что разложенный в растр отрезок лежит по одну сторону от реального и что на одном из концов отрезка появляется лишняя точка, т. е. результат работы алгоритма зависит от ориентации. Следовательно, точность в концевых точках ухудшается. Далее, если вместо взятия целой части использовать округление до ближайшего целого, то ре-

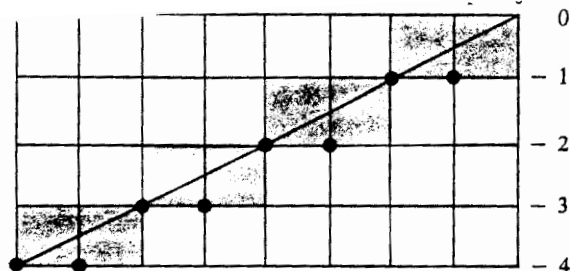


Рис. 2.3. Результаты работы простого ЦДА в третьем квадранте.

зультаты снова получатся разными. Таким образом, либо нужно использовать более сложный и более медленный алгоритм, либо надо отступить от требования максимально точной аппроксимации. Вдобавок предложенный алгоритм имеет тот недостаток, что он использует вещественную арифметику. В следующем разделе описан более приемлемый алгоритм.

2.3. АЛГОРИТМ БРЕЗЕНХЕМА

Хотя алгоритм Брезенхема [2-1] был первоначально разработан для цифровых графопостроителей, однако он в равной степени подходит и для использования растровыми устройствами с ЭЛТ. Алгоритм выбирает оптимальные растровые координаты для представления отрезка. В процессе работы одна из координат — либо x , либо y (в зависимости от углового коэффициента) — изменяется на единицу. Изменение другой координаты (либо на нуль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние мы назовем ошибкой.

Алгоритм построен так, что требуется проверять лишь знак этой ошибки. На рис. 2.4 это иллюстрируется для отрезка в первом октанте, т. е. для отрезка с угловым коэффициентом, лежащим в диапазоне от нуля до единицы. Из рисунка можно заметить, что если угловой коэффициент отрезка из точки $(0, 0)$ больше чем $1/2$, то его пересечение с прямой $x = 1$ будет расположено ближе к прямой $y = 1$, чем к прямой $y = 0$. Следовательно, точка растра $(1, 1)$ лучше аппроксимирует ход отрезка, чем точка $(1, 0)$. Если угловой коэффициент меньше $1/2$, то верно обратное. Для углового коэффи-

представлении отрезка дискретными пикселями. Так как желательно проверять только знак ошибки, то она первоначально устанавливается равной $-1/2$. Таким образом, если угловой коэффициент отрезка больше или равен $1/2$, то величина ошибки в следующей точке растра с координатами $(1, 0)$ может быть вычислена как

$$e = e + m$$

где m — угловой коэффициент. В нашем случае при начальном значении ошибки $-1/2$

$$e = -1/2 + 3/8 = -1/8$$

Так как e отрицательно, отрезок пройдет ниже середины пикселя. Следовательно, пиксел на том же самом горизонтальном уровне лучше аппроксимирует положение отрезка, поэтому y не увеличивается. Аналогично вычисляем ошибку

$$e = -1/8 + 3/8 = 1/4$$

в следующей точке растра $(2, 0)$. Теперь e положительно, а значит, отрезок пройдет выше средней точки. Растровый элемент $(2, 1)$ со следующей по величине координатой y лучше аппроксимирует положение отрезка. Следовательно, y увеличивается на единицу. Прежде чем рассматривать следующий пиксел, необходимо откорректировать ошибку вычитанием из нее единицы. Имеем

$$e = 1/4 - 1 = -3/4$$

Заметим, что пересечение вертикальной прямой $x = 2$ с заданным отрезком лежит на $1/4$ ниже прямой $y = 1$. Если же перенести отрезок $1/2$ вниз, мы получим как раз величину $-3/4$. Продолжение вычислений для следующего пикселя дает

$$e = -3/4 + 3/8 = -3/8$$

Так как e отрицательно, то y не увеличивается. Из всего сказанного следует, что ошибка — это интервал, отсекаемый по оси y рассматриваемым отрезком в каждом растровом элементе (относительно $-1/2$).

Приведем алгоритм Брезенхема для первого октанта, т. е. для случая $0 \leq \Delta y \leq \Delta x$.

Алгоритм Брезенхема разложения в растр отрезка для первого октанта

предполагается, что концы отрезка (x_1, y_1) и (x_2, y_2) не совпадают

Integer — функция преобразования в целое

$x, y, \Delta x, \Delta y$ — целые

e — вещественное

инициализация переменных

$x = x_1$

$y = y_1$

$\Delta x = x_2 - x_1$

$\Delta y = y_2 - y_1$

инициализация e с поправкой на половину пиксела

$e = \Delta y / \Delta x - 1/2$

начало основного цикла

for $i = 1$ **to** Δx

Plot (x, y)

while ($e \geq 0$)

$y = y + 1$

$e = e - 1$

end while

$x = x + 1$

$e = e + \Delta y / \Delta x$

next i

finish

Блок-схема алгоритма приводится на рис. 2.6. Пример дан ниже.

Пример 2.3. Алгоритм Брезенхема

Рассмотрим отрезок, проведенный из точки (0, 0) в точку (5, 5). Разложение отрезка в растр по алгоритму Брезенхема приводит к такому результату:

начальные установки

$x = 0$

$y = 0$

$\Delta x = 5$

$\Delta y = 5$

$e = 1 - 1/2 = 1/2$

Результаты пошагового выполнения основного цикла

i	Plot	e	x	y
1	(0, 0)	$1/2$	0	0
		$-1/2$	0	1
		$1/2$	1	1
2	(1, 1)	$-1/2$	1	2
		$1/2$	2	2

3	(2, 2)	- 1/2	2	3
		1/2	3	3
4	(3, 3)	- 1/2	3	4
		1/2	4	4
5	(4, 4)	- 1/2	4	5
		1/2	5	5

Результат показан на рис. 2.7 и совпадает с ожидаемым. Заметим, что точка раstra с координатами (5, 5) не активирована. Эту точку можно активировать путем изменения условия цикла **for-next**

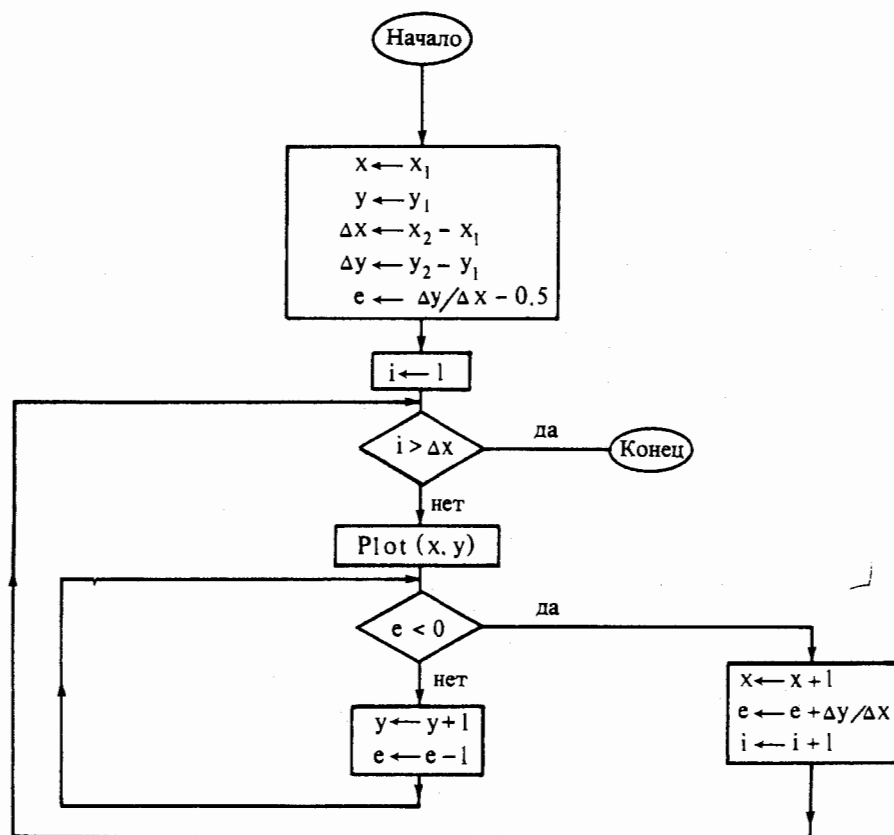


Рис. 2.6. Блок-схема алгоритма Брезенхема.

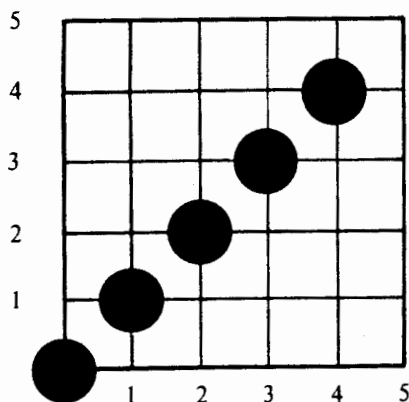


Рис. 2.7. Результат работы алгоритма Брезенхема в первом октанте.

на 0 to Δx . Активацию точки (0, 0) можно устранить, если поставить оператора **Plot** непосредственно перед строкой **next i**.

2.4. ЦЕЛОЧИСЛЕННЫЙ АЛГОРИТМ БРЕЗЕНХЕМА

Алгоритм Брезенхема в том виде, как он представлен выше, требует использования арифметики с плавающей точкой и деления (для вычисления углового коэффициента и оценки ошибки). Быстродействие алгоритма можно увеличить, если использовать только целочисленную арифметику и исключить деление. Так как важен лишь знак ошибки, то простое преобразование

$$\bar{e} = 2e \Delta x$$

превратит предыдущий алгоритм в целочисленный и позволит эффективно реализовать его на аппаратном или микропрограммном уровне. Модифицированный целочисленный алгоритм для первого октанта, т. е. для $0 \leq \Delta y \leq \Delta x$, таков:

Целочисленный алгоритм Брезенхема для первого октанта

предполагается, что концы отрезка (x_1, y_1) и (x_2, y_2) не совпадают и все переменные — целые

$$x = x_1$$

$$y = y_1$$

$$\Delta x = x_2 - x_1$$

$$\Delta y = y_2 - y_1$$

инициализируем \bar{e} с поправкой на половину пиксела

$$\bar{e} = 2 * \Delta y - \Delta x$$

начало основного цикла

for $i = 1$ **to** Δx

Plot (x, y)

while ($\bar{e} \geq 0$)

$\bar{y} = \bar{y} + 1$

$\bar{e} = \bar{e} - 2 * \Delta x$

end while

$\bar{x} = \bar{x} + 1$

$\bar{e} = \bar{e} + 2 * \Delta y$

next i

finish

Блок-схема, помещенная на рис. 2.6, применима и в данном случае с соответствующими изменениями в вычислении ошибки.

2.5. ОБЩИЙ АЛГОРИТМ БРЕЗЕНХЕМА

Чтобы реализация алгоритма Брезенхема была полной, необходимо обрабатывать отрезки во всех октантах. Модификацию легко сделать, учитывая в алгоритме номер квадранта, в котором лежит отрезок и его угловой коэффициент. Когда абсолютная величина углового коэффициента больше 1, y постоянно изменяется на единицу, а критерий ошибки Брезенхема используется для принятия решения об изменении величины x . Выбор постоянно изменяющейся (на $+1$ или -1) координаты зависит от квадранта (рис. 2.8). Общий алгоритм может быть оформлен в следующем виде:

Обобщенный целочисленный алгоритм Брезенхема квадрантов

*предполагается, что концы отрезка (x_1, y_1) и (x_2, y_2) не совпадают
все переменные считаются целыми*

*функция **Sign** возвращает $-1, 0, 1$ для отрицательного, нулевого и положительного аргумента соответственно*

инициализация переменных

$x = x_1$

$y = y_1$

$\Delta x = \text{abs}(x_2 - x_1)$

$\Delta y = \text{abs}(y_2 - y_1)$

$s_1 = \text{Sign}(x_2 - x_1)$

$s_2 = \text{Sign}(y_2 - y_1)$

обмен значений Δx и Δy в зависимости от углового коэффици-

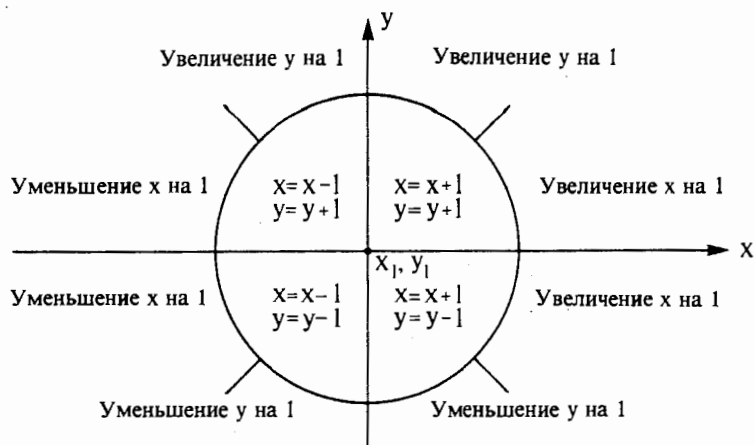


Рис. 2.8. Разбор случаев для обобщенного алгоритма Брезенхема.

ента наклона отрезка

if $\Delta y > \Delta x$ **then**

Врем = Δx

$\Delta x = \Delta y$

$\Delta y = \text{Врем}$

Обмен = 1

else

Обмен = 0

end if

инициализация \bar{e} с поправкой на половину пиксела

$\bar{e} = 2 * \Delta y - \Delta x$

основной цикл

for $i = 1$ **to** Δx

Plot(x, y)

while ($\bar{e} \geq 0$)

if Обмен = 1 **then**

$x = x + s_1$

else

$y = y + s_2$

end if

$\bar{e} = \bar{e} - 2 * \Delta x$

end while

if Обмен = 1 **then**

$y = y + s_2$

else

$x = x + s_1$

```

end if
 $\bar{e} = \bar{e} + 2 * \Delta y$ 
next i
finish

```

Пример 2.4. Обобщенный алгоритм Брезенхема

Для иллюстрации общего алгоритма Брезенхема рассмотрим отрезок из точки (0, 0) в точку (-8, -4). В примере 2.2 этот отрезок был обработан с помощью простого БРА:

начальные установки

```

x = 0
y = 0
 $\Delta x = 8$ 
 $\Delta y = 4$ 
 $s_1 = -1$ 
 $s_2 = -1$ 
Обмен = 0
e = 0

```

пошаговое выполнение основного цикла

i	Plot	e	x	y
		0	0	0
1	(0, 0)	-16	0	-1
		-8	-1	-1
2	(-1, -1)	0	-2	-1
3	(-2, -1)	-16	-2	-2
		-8	-3	-2
4	(-3, -2)	0	-4	-2
5	(-4, -2)	-16	-4	-3
		-8	-5	-3
6	(-5, -3)	0	-6	-3
7	(-6, -3)	-16	-6	-4
		-8	-7	-4
8	(-7, -4)	0	-8	-4

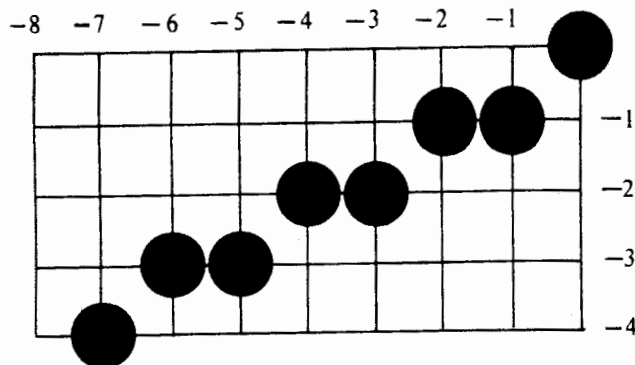


Рис. 2.9. Результат работы обобщенного алгоритма Брезенхема в третьем квадранте.

На рис. 2.9 продемонстрирован результат. Сравнение с рис. 2.3 показывает, что результаты работы двух алгоритмов отличаются.

2.6. АЛГОРИТМ БРЕЗЕНХЕМА ДЛЯ ГЕНЕРАЦИИ ОКРУЖНОСТИ

В растр нужно разлагать не только линейные, но и другие, более сложные функции. Разложению конических сечений, т. е. окружностей, эллипсов, парабол, гипербол, было посвящено значительное число работ [2-2 — 2-5]. Наибольшее внимание, разумеется, уделено окружности [2-6 — 2-9]. Один из наиболее эффективных и простых для понимания алгоритмов генерации окружности принадлежит Брезенхему [2-10]. Для начала заметим, что необходимо сгенерировать только одну восьмую часть окружности. Остальные ее части могут быть получены последовательными отражениями, как это показано на рис. 2.10. Если сгенерирован первый октант (от 0 до 45° против часовой стрелки), то второй октант можно получить зеркальным отражением относительно прямой $y = x$, что дает в совокупности первый квадрант. Первый квадрант отражается относительно прямой $x = 0$ для получения соответствующей части окружности во втором квадранте. Верхняя полуокружность отражается относительно прямой $y = 0$ для завершения построения. На рис. 2.10 приведены двумерные матрицы соответствующих преобразований.

Для вывода алгоритма рассмотрим первую четверть окружности с центром в начале координат. Заметим, что если работа алгоритма начинается в точке $x = 0, y = R$, то при генерации окружности по часовой стрелке в первом квадранте y является монотон-

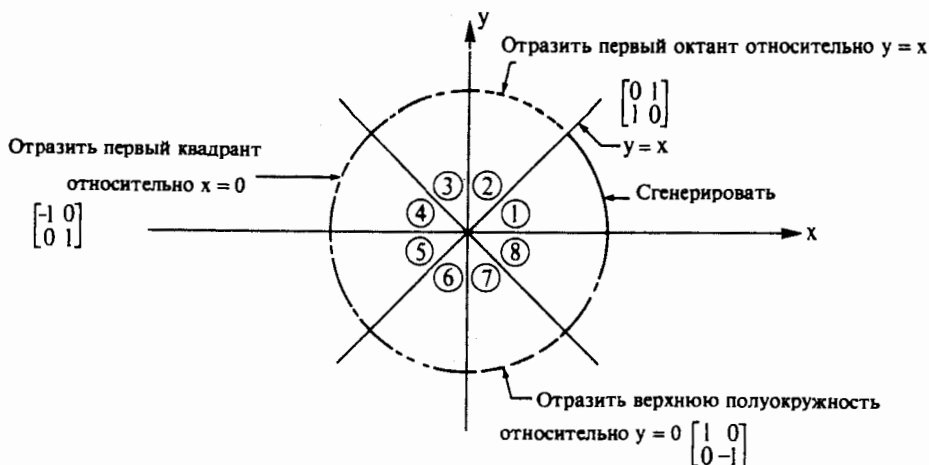


Рис. 2.10. Генерация полной окружности из дуги в первом октанте.

но убывающей функцией аргумента x (рис. 2.11). Аналогично, если исходной точкой является $y = 0, x = R$, то при генерации окружности против часовой стрелки x будет монотонно убывающей функцией аргумента y . В нашем случае выбирается генерация по часовой стрелке с началом в точке $x = 0, y = R$. Предполагается, что центр окружности и начальная точка находятся точно в точках раstra.

Для любой заданной точки на окружности при генерации по часовой стрелке существует только три возможности выбрать следующий пиксел, наилучшим образом приближающий окружность: горизонтально вправо, по диагонали вниз и вправо, вертикально вниз. На рис. 2.12 эти направления обозначены соответственно m_H , m_D , m_V . Алгоритм выбирает пиксел, для которого минимален квадрат расстояния между одним из этих пикселов и окружностью, т. е. минимум из¹⁾

$$m_H = |(x_i + 1)^2 + (y_i)^2 - R^2|$$

$$m_D = |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

$$m_V = |(x_i)^2 + (y_i - 1)^2 - R^2|$$

Вычисления можно упростить, если заметить, что в окрестности точки (x_i, y_i) возможны только пять типов пересечений окружности и сетки раstra, приведенных на рис. 2.13.

¹⁾ Здесь минимизируется не квадрат расстояния, а абсолютное значение разности квадратов расстояний от центра окружности до пиксела и до окружности. — *Прим. перев.*

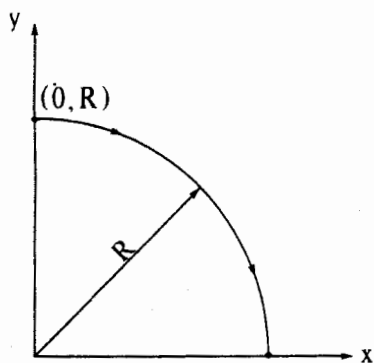


Рис. 2.11. Окружность в первом квадранте.

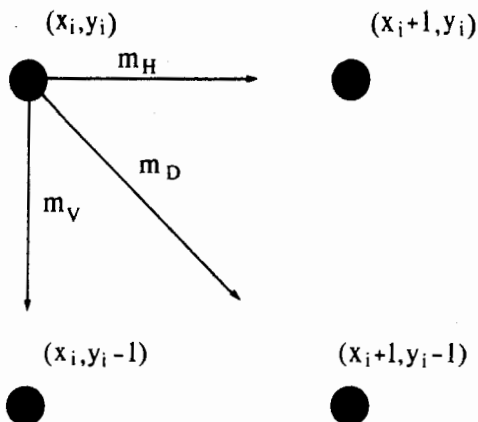


Рис. 2.12. Выбор пикселей в первом квадранте.

Разность между квадратами расстояний от центра окружности до диагонального пиксела $(x_i + 1, y_i - 1)$ и от центра до точки на окружности R^2 равна

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Как и в алгоритме Брезенхема для отрезка, для выбора соответствующего пиксела желательно использовать только знак ошибки, а не ее величину.

При $\Delta_i < 0$ диагональная точка $(x_i + 1, y_i - 1)$ находится внутри реальной окружности, т. е. это случаи 1 или 2 на рис. 2.13. Ясно, что в этой ситуации следует выбрать либо пиксел $(x_i + 1, y_i)$, т. е. m_H , либо пиксел $(x_i + 1, y_i - 1)$, т. е. m_D . Для этого сначала рассмотрим случай 1 и проверим разность квадратов расстояний от

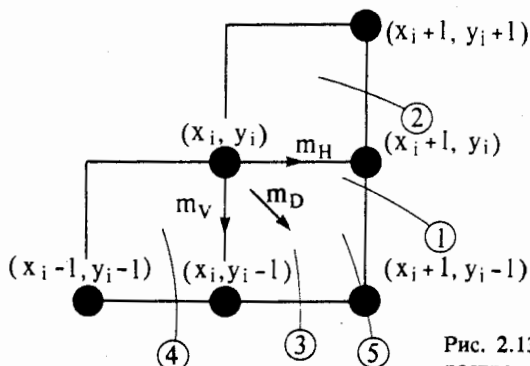


Рис. 2.13. Пересечение окружности и сетки раstra.

окружности до пикселей в горизонтальном и диагональном направлениях:

$$\delta = |(x_i + 1)^2 + (y_i)^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

При $\delta < 0$ расстояние от окружности до диагонального пикселя (m_D) больше, чем до горизонтального (m_H). Напротив, если $\delta > 0$, расстояние до горизонтального пикселя (m_H) больше. Таким образом,

при $\delta \leq 0$ выбираем m_H в $(x_i + 1, y_i)$

при $\delta > 0$ выбираем m_D в $(x_i + 1, y_i - 1)$

При $\delta = 0$, когда расстояние от окружности до обоих пикселей одинаковы, выбираем горизонтальный шаг.

Количество вычислений, необходимых для оценки величины δ , можно сократить, если заметить, что в случае 1

$$(x_i + 1)^2 + (y_i)^2 - R^2 \geq 0$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

так как диагональный пиксел $(x_i + 1, y_i - 1)$ всегда лежит внутри окружности, а горизонтальный $(x_i + 1, y_i)$ — вне ее. Таким образом, δ можно вычислить по формуле

$$\delta = (x_i + 1)^2 + (y_i)^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Дополнение до полного квадрата члена $(y_i)^2$ с помощью добавления и вычитания $-2y_i + 1$ дает

$$\delta = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] + 2y_i - 1$$

В квадратных скобках стоит по определению Δ_i и его подстановка

$$\delta = 2(\Delta_i + y_i) - 1$$

существенно упрощает выражение.

Рассмотрим случай 2 на рис. 2.13 и заметим, что здесь должен быть выбран горизонтальный пиксел $(x_i + 1, y_i)$, так как y является монотонно убывающей функцией. Проверка компонент δ показывает, что

$$(x_i + 1)^2 + (y_i)^2 - R^2 < 0$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

поскольку в случае 2 горизонтальный $(x_i + 1, y_i)$ и диагональный $(x_i + 1, y_i - 1)$ пиксели лежат внутри окружности. Следовательно,

$\delta < 0$, и при использовании того же самого критерия, что и в случае 1, выбирается пиксел $(x_i + 1, y_i)$.

Если $\Delta_i > 0$, то диагональная точка $(x_i + 1, y_i - 1)$ находится вне окружности, т. е. это случаи 3 и 4 на рис. 2.13. В данной ситуации ясно, что должен быть выбран либо пиксел $(x_i + 1, y_i - 1)$, т. е. m_D , либо $(x_i, y_i - 1)$, т. е. m_V . Аналогично разбору предыдущего случая критерий выбора можно получить, рассматривая сначала случай 3 и проверяя разность между квадратами расстояний от окружности до диагонального m_D и вертикального m_V пикселей, т. е.

$$\delta' = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |(x_i)^2 + (y_i - 1)^2 - R^2|$$

При $\delta' < 0$ расстояние от окружности до вертикального пикселя $(x_i, y_i - 1)$ больше и следует выбрать диагональный шаг m_D , к пикселу $(x_i + 1, y_i - 1)$. Напротив, в случае $\delta' > 0$ расстояние от окружности до диагонального пикселя больше и следует выбрать вертикальное движение к пикселу $(x_i, y_i - 1)$. Таким образом,

при $\delta' \leq 0$ выбираем m_D в $(x_i + 1, y_i - 1)$

при $\delta' > 0$ выбираем m_V в $(x_i, y_i - 1)$

Здесь в случае $\delta' = 0$, т. е. когда расстояния равны, выбран диагональный шаг.

Проверка компонент δ' показывает, что

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 \geq 0$$

$$(x_i)^2 + (y_i - 1)^2 - R^2 < 0$$

поскольку для случая 3 диагональный пиксел $(x_i + 1, y_i - 1)$ находится вне окружности, тогда как вертикальный пиксел $(x_i, y_i - 1)$ лежит внутри ее. Это позволяет записать δ' в виде

$$\delta' = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + (x_i)^2 + (y_i - 1)^2 - R^2$$

Дополнение до полного квадрата члена $(x_i)^2$ с помощью добавления и вычитания $2x_i + 1$ дает

$$\delta' = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] - 2x_i - 1$$

Использование определения Δ_i приводит выражение к виду

$$\delta' = 2(\Delta_i - x_i) - 1$$

Теперь, рассматривая случай 4, снова заметим, что следует выбрать вертикальный пиксел $(x_i, y_i - 1)$, так как y является моно-

тонно убывающей функцией при возрастании x .

Проверка компонент δ' для случая 4 показывает, что

$$\begin{aligned}(x_i + 1)^2 + (y_i - 1)^2 - R^2 &> 0 \\ (x_i)^2 + (y_i - 1)^2 - R^2 &> 0\end{aligned}$$

поскольку оба пиксела находятся вне окружности. Следовательно, $\delta' > 0$ и при использовании критерия, разработанного для случая 3, происходит верный выбор m_V .

Осталось проверить только случай 5 на рис. 2.13, который встречается, когда диагональный пиксел $(x_i + 1, y_i - 1)$ лежит на окружности, т. е. $\Delta_i = 0$. Проверка компонент δ показывает, что

$$\begin{aligned}(x_i + 1)^2 + (y_i)^2 - R^2 &> 0 \\ (x_i + 1)^2 + (y_i - 1)^2 - R^2 &= 0\end{aligned}$$

Следовательно, $\delta > 0$ и выбирается диагональный пиксел $(x_i + 1, y_i - 1)$. Аналогичным образом оцениваем компоненты δ' :

$$\begin{aligned}(x_i + 1)^2 + (y_i - 1)^2 - R^2 &= 0 \\ (x_i)^2 + (y_i - 1)^2 - R^2 &< 0\end{aligned}$$

и $\delta' < 0$, что является условием выбора правильного диагонального шага к $(x_i + 1, y_i - 1)$. Таким образом, случай $\Delta_i = 0$ подчиняется тому же критерию, что и случай $\Delta_i < 0$ или $\Delta_i > 0$.

Подведем итог полученных результатов:

$$\Delta_i < 0$$

$\delta \leq 0$ выбираем пиксел $(x_i + 1, y_i) \rightarrow m_H$

$\delta > 0$ выбираем пиксел $(x_i + 1, y_i - 1) \rightarrow m_D$

$$\Delta_i > 0$$

$\delta' \leq 0$ выбираем пиксел $(x_i + 1, y_i - 1) \rightarrow m_D$

$\delta' > 0$ выбираем пиксел $(x_i, y_i - 1) \rightarrow m_V$

$\Delta_i = 0$ выбираем пиксел $(x_i + 1, y_i - 1) \rightarrow m_D$

Легко разработать простые рекуррентные соотношения для реализации пошагового алгоритма. Сначала рассмотрим горизонтальный шаг m_H к пикселу $(x_i + 1, y_i)$. Обозначим это новое положение пиксела как $(i + 1)$. Тогда координаты нового пиксела и значение Δ_i равны

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i$$

$$\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2$$

$$\begin{aligned}
&= (x_{i+1})^2 + 2x_{i+1} + 1 + (y_i - 1)^2 - R^2 \\
&= (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2x_{i+1} + 1 \\
&= \Delta_i + 2x_{i+1} + 1
\end{aligned}$$

Аналогично координаты нового пиксела и значение Δ_i для шага m_D к пикселу $(x_i + 1, y_i - 1)$ таковы:

$$\begin{aligned}
x_{i+1} &= x_i + 1 \\
y_{i+1} &= y_i - 1 \\
\Delta_{i+1} &= \Delta_i + 2x_{i+1} - 2y_{i+1} + 2
\end{aligned}$$

То же самое для шага m_V к $(x_i, y_i - 1)$

$$\begin{aligned}
x_{i+1} &= x_i \\
y_{i+1} &= y_i - 1 \\
\Delta_{i+1} &= \Delta_i - 2y_{i+1} + 1
\end{aligned}$$

Реализация алгоритма Брезенхема на псевдокоде для окружности приводится ниже.

Пошаговый алгоритм Брезенхема для генерации окружности в первом квадранте

все переменные — целые

инициализация переменных

$$x_i = 0$$

$$y_i = R$$

$$\Delta_i = 2(1 - R)$$

$$\text{Предел} = 0$$

1 **Plot**(x_i, y_i)

If $y_i \leq \text{Предел}$ **then** 4

Выделение случая 1 или 2, 4 или 5, или 3

If $\Delta_i < 0$ **then** 2

If $\Delta_i > 0$ **then** 3

If $\Delta_i = 0$ **then** 20

определение случая 1 или 2

2 $\delta = 2\Delta_i + 2y_i - 1$

If $\delta \leq 0$ **then** 10

If $\delta > 0$ **then** 20

определение случая 4 или 5

3 $\delta' = 2\Delta_i + 2x_i - 1$

If $\delta' \leq 0$ **then** 20

If $\delta' > 0$ **then** 30

выполнение шагов

шаг m_H

10 $x_i = x_i + 1$
 $\Delta_i = \Delta_i + 2x_i + 1$
go to 1

шаг m_D

20 $x_i = x_i + 1$
 $y_i = y_i - 1$
 $\Delta_i = \Delta_i + 2x_i - 2y_i + 2$
go to 1

шаг m_V

30 $y_i = y_i - 1$
 $\Delta_i = \Delta_i - 2y_i + 1$
go to 1

4 **finish**

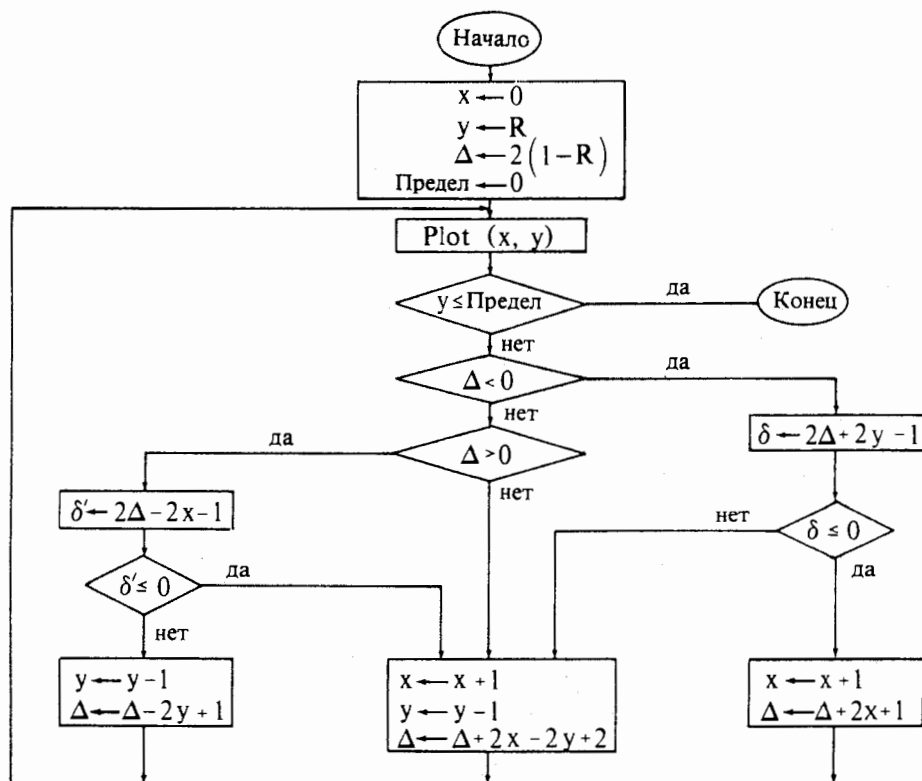


Рис. 2.14. Блок-схема пошагового алгоритма Брезенхема генерации окружности в первом квадранте.

Переменная предела устанавливается в нуль для окончания работы алгоритма на горизонтальной оси, в результате генерируется окружность в первом квадранте. Если необходим лишь один из октантов, то второй октант можно получить с помощью установки $\text{Предел} = \text{Integer}(R/\sqrt{2})$, а первый — с помощью отражения второго октанта относительно прямой $y = x$ (рис. 2.10). Блок-схема алгоритма приводится на рис. 2.14.

Пример 2.5. Алгоритм Брезенхема для окружности

Для иллюстрации работы алгоритма генерации окружности рассмотрим окружность радиуса 8 с центром в начале координат. Генерируется только первый квадрант.

начальные установки

```
x = 0
y = 8
Δi = 2(1 - 8) = -14
Предел = 0
```

пошаговое выполнение основного цикла

```
1 Plot(0,8)
  yi > Предел           продолжать
  Δi < 0 go to 2
2 δ = 2(-14) + 2(8) - 1 = -13 < 0 go to 10
10 x = 0 + 1 = 1
   Δi = -14 + 2 + 1 = -11
   go to 1
```

```
1 Plot(1,8)
  yi > Предел           продолжать
  Δi < 0 go to 2
2 δ = 2(-11) + 2(8) - 1 = -7 < 0 go to 10
10 x = 1 + 1 = 2
   Δi = -11 + 2(2) + 1 = -6
   go to 1
```

```
1 Plot(2,8)
```

```

.
.
.
```

продолжать

Результаты всех последовательных проходов алгоритма сведены в таблицу. Список пикселей, выбранных алгоритмом, состоит из (0, 8), (1, 8), (2, 8), (3, 7), (4, 7), (5, 6), (6, 5), (7, 4), (7, 3), (8, 2), (8, 1), (8, 0).

Plot	Δ _i	δ	δ'	x	y
	-14			0	8
(0, 8)	-11	-13		1	8

2.7. РАСТРОВАЯ РАЗВЕРТКА — СПОСОБ ГЕНЕРАЦИИ ИЗОБРАЖЕНИЯ

Для вывода на видеомонитор разложенный в растр образ необходимо представить в виде того шаблона, который требует дисплей (см. разд. 1.8). Это преобразование называется растровой разверткой. В отличие от дисплейного списка для векторного дисплея, содержащего информацию только об отрезках или литерах, в данном случае дисплейный список должен содержать информацию о каждом пикселе на экране. Необходимо, кроме того, чтобы эта информация организовывалась и выводилась со скоростью видеогенерации в порядке сканирования строк, т. е. сверху вниз и слева направо. Существует четыре способа достижения такого результата — растровая развертка в реальном времени, групповое кодирование, клеточная организация и память буфера кадра.

2.8. РАСТРОВАЯ РАЗВЕРТКА В РЕАЛЬНОМ ВРЕМЕНИ

При развертке в реальном времени или «на лету» сцена произвольно представляется в терминах визуальных атрибутов и геометрических характеристик. Типичными визуальными атрибутами являются цвет, оттенок и интенсивность, тогда как координаты x , y , углы наклона и текст относятся к геометрическим характеристикам. Последние упорядочены по координате y . Во время воспроизведения каждого кадра процессор сканирует эту информацию и вычисляет интенсивность каждого пиксела на экране. При такой развертке не нужны большие количества памяти. Требования на память обычно ограничиваются необходимостью хранить дисплейный список плюс одну сканирующую строку. Более того, поскольку информация о сцене хранится в произвольно организованном дисплейном списке, добавление или удаление информации из списка осуществляется легко, а это удобно для динамических приложений. Однако сложность выводимого изображения ограничивается скоростью дисплейного процессора. Обычно это означает, что ограничено число отрезков или многоугольников в картине, количество пересечений со сканирующей строкой или число цветов или полутонов серого.

Для получения пересечений (если они есть) каждого отрезка дисплейного списка со сканирующей строкой в простейшей реализации метода всякий раз при изображении строки обрабатывается весь дисплейный список. При регенерации видеоизображения на каждую сканирующую строку, а значит, и на обработку всего списка приходится только 63.5 микросекунды. Столь малое время позволяет ис-

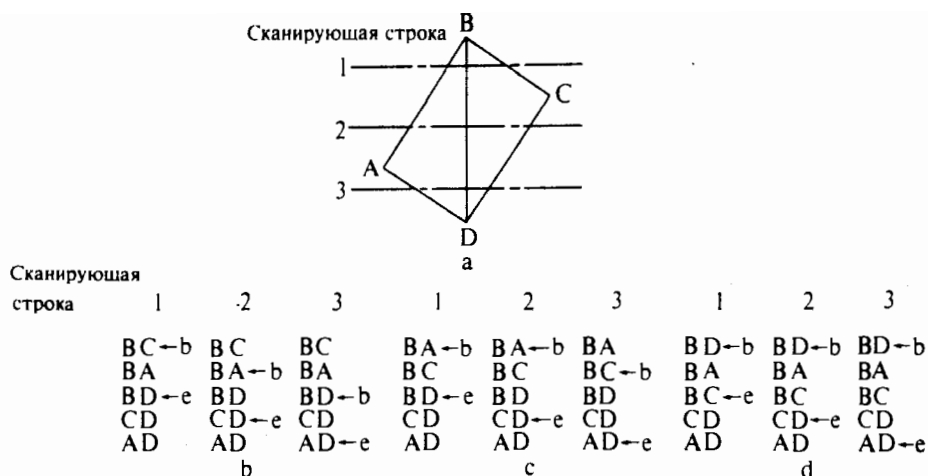


Рис. 2.16. Простой список активных ребер.

пользовать данный метод только для рисования несложных чертежей, не более. Так как в общем случае не каждый отрезок в сцене пересекает каждую сканирующую строку, то количество вычислений может быть сокращено путем введения списка активных ребер (САР). Этот список содержит те отрезки изображения, которые пересекают сканирующую строку.

Для организации и управления САР можно использовать ряд методов. Сначала отрезки изображения сортируются по наибольшей координате y . В одном из простых методов такой сортировки используются два перемещающихся указателя в отсортированном списке. Указатель начала используется для обозначения начала списка активных ребер, а указатель конца — для обозначения конца этого списка. На рис. 2.16,а представлена сцена из нескольких отрезков с тремя характерными сканирующими строками. На рис. 2.16,б показан типичный отсортированный список отрезков фигуры. Указатель начала в исходном положении устанавливается на начало этого списка, т. е. на отрезок BC . Указатель конца установлен на тот последний отрезок в списке, который начинается *выше* рассматриваемой сканирующей строки, т. е. на отрезок BD . При сканировании изображения необходимо корректировать САР, при этом указатель конца передвигают вниз, чтобы включить в список новые отрезки, начинающиеся на текущей сканирующей строке или выше нее. В то же самое время указатель начала передвигают вниз, чтобы исключить отрезки, кончающиеся выше текущей сканирующей строки. Это изображено на рис. 2.16 для скани-

рующих строк, помеченных цифрами 2 и 3 на рис. 2.16,а. Рисунки 2.16,с и d иллюстрируют проблему, возникающую в этом простом алгоритме. Порядок сортировки отрезков, начинающихся с одной и той же координаты y , влияет на размер списка активных ребер. Например, отрезок BC на рис. 2.16,d никогда не покинет этот список. В результате может обрабатываться больше информации, чем на самом деле необходимо.

Эту и аналогичные проблемы можно устранить путем введения дополнительной структуры данных. При этом можно упростить также вычисление пересечения каждого отрезка изображения со сканирующими строками. Сначала выполняется групповая сортировка по y всех отрезков изображения. При групповой сортировке по y ¹⁾ (рис. 2.17,b) просто создаются области памяти или группы для каждой сканирующей строки. Если, например, применяется 512 сканирующих строк, то используется 512 групп. При просмотре отрезков в дисплейном списке информация о каждом отрезке помещается в группу, соответствующую наибольшей величине координаты y для отрезка. Для простого черно-белого контурного изображения необходимо записывать только координату x точки пересечения с групповой сканирующей строкой, Δx — изменение этой координаты x при переходе от одной сканирующей строки к другой, и Δy — число сканирующих строк, пересекаемых отрезком. Для простых изображений большинство из y -групп будет пусто.

Список активных ребер для текущей сканирующей строки формируется добавлением информации из y -группы, соответствующей этой строке. Координаты x точек пересечения сортируются в порядке сканирования, и ребра из CAP преобразуются в растровую форму. После этого для каждого отрезка из CAP Δy уменьшается на единицу. Если $\Delta y < 0$, то отрезок исключается из списка. И наконец, для каждого отрезка координата x точки пересечения для новой сканирующей строки получается добавлением к прежнему значению величины Δx . Этот процесс повторяется для всех сканирующих строк. На рис. 2.17,с приводится CAP для сканирующих строк 3, 5 и 7 из простой сцены на рис. 2.17,а.

Если используется фиксированный размер y -групп, то для пересечений с каждой сканирующей строкой выделяется фиксированное количество памяти. Таким образом, максимальное число пересечений с произвольной сканирующей строкой предопределено заранее и, следовательно, сложность изображения ограничена. Одним из

¹⁾ Групповая сортировка является одной из форм распределяющей сортировки, где основание счисления равно числу групп (сканирующих строк). См. Кнут [2-11].

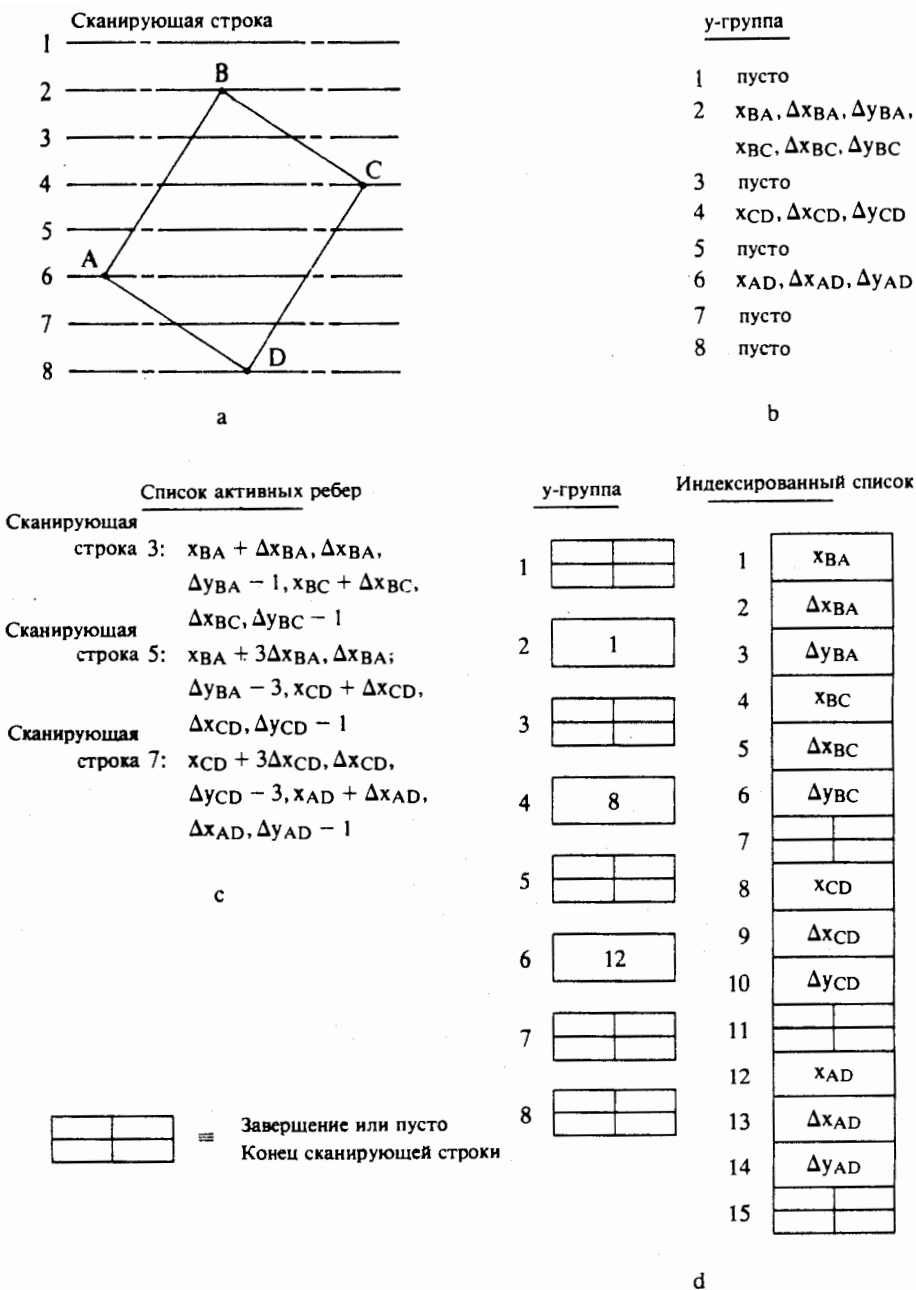


Рис. 2.17. Групповая сортировка по y , список активных ребер и структура последовательного индексированного списка.

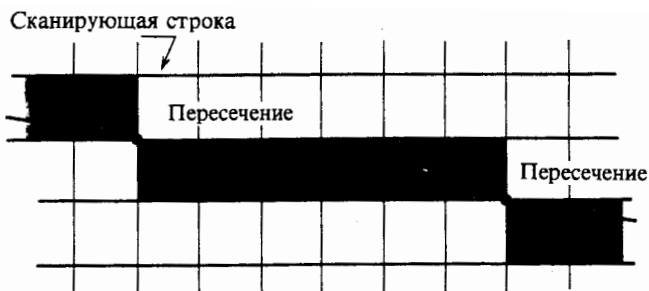


Рис. 2.18. Простой метод растровой развертки для почти горизонтальных отрезков.

методов, позволяющих преодолеть это ограничение, может служить использование в качестве структуры данных последовательного индексированного списка. В этом случае каждая у-группа содержит только указатель на расположение в структуре данных информации для первого отрезка из группы (т. е. начинающегося на этой сканирующей строке). На рис. 2.17, d показаны последовательный индексированный список и структура данных для рис. 2.17, а. В этой конкретной ситуации предполагается, что данные для текущей сканирующей строки выбираются группами по три до тех пор, пока не встретится пустая ссылка или знак завершения.

Метод определения пересечений отрезков со сканирующими строками дает хорошие результаты для вертикальных и почти вертикальных отрезков. Однако для почти горизонтальных отрезков будет вычислено очень мало точек пересечения, что приведет к неприемлемому изображению отрезка. В качестве простого решения можно предложить определять пересечения на двух последовательных сканирующих строках и активировать все пиксели между точками пересечений, как это показано на рис. 2.18. Для горизонтальных отрезков используются концевые точки.

Так как все изображение обрабатывается для каждого видеокадра, развертка в реальном времени применима для высоко интерактивной графики. При использовании групповой сортировки по у отрезки могут быть добавлены или удалены из дисплейного списка простым добавлением или удалением их из соответствующей у-группы и связанной с ней структуры данных. Как показано выше на рис. 2.17, b, это легче всего сделать для у-групп фиксированной длины. Для удобства добавления и удаления отрезков в сцене используется структура данных в виде связного списка (рис. 2.19). Заметим, что для связного списка на рис. 2.19, b, требуются признаки конца каждой группы данных и ссылка на следующую группу для рассматриваемой сканирующей строки, например элемент 4, а так-

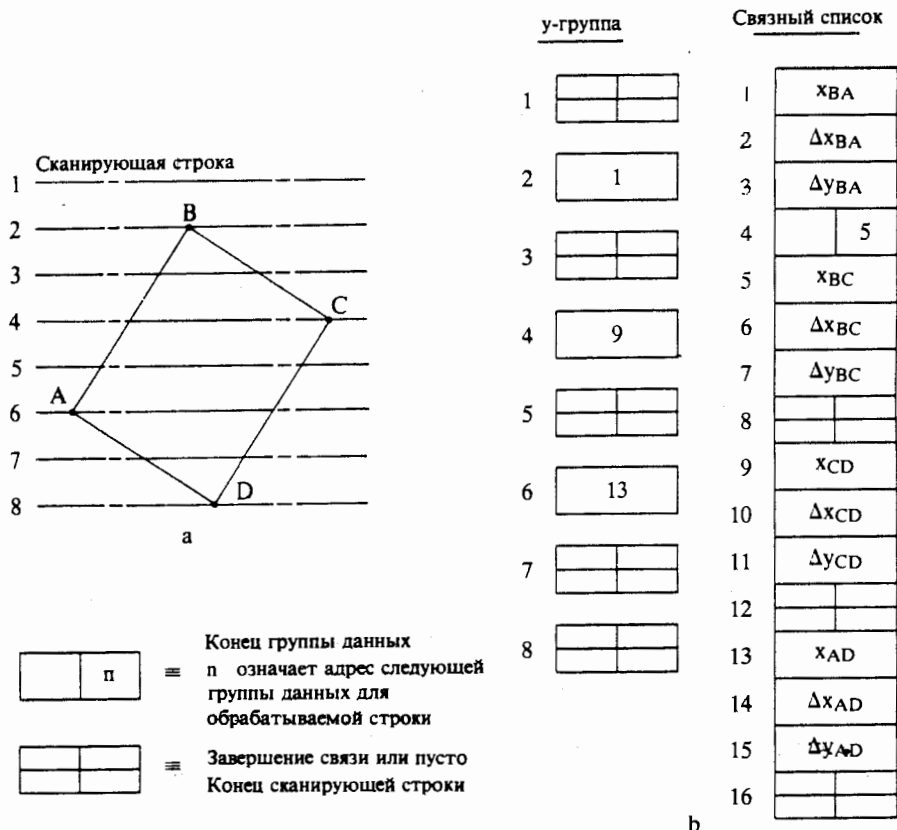


Рис. 2.19. Групповая сортировка по у и связный список для случая интерактивной графики.

же признак завершения связи. При добавлении отрезка *BD* список модифицируется так, как показано на рис. 2.19,d. Информация об отрезке *BD* добавляется к концу списка данных. Дисплейный процессор направляется в эту ячейку с помощью модифицированной ссылки из ячейки 8. Если теперь отрезок *BC* удаляется из фигуры, список модифицируется так, как показано на рис. 2.19,f. Заметим, что ссылка в ячейке 4 модифицирована для того, чтобы обойти ячейки, содержащие информацию об отрезке *BC*.

Этот простой пример иллюстрирует основные идеи для модификации связного списка в интерактивных графических системах. Однако здесь не приведены все необходимые подробности. Например, должно быть очевидно, что длина списка будет постоянно



с

у-группа

Связный список

1	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					1	x _{BA}	
2	<table><tr><td colspan="2">1</td></tr></table>	1		2	Δx_{BA}			
1								
3	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					3	Δy_{BA}	
4	<table><tr><td colspan="2">9</td></tr></table>	9		4		5		
9								
5	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					5	x _{BC}	
6	<table><tr><td colspan="2">13</td></tr></table>	13		6	Δx_{BC}			
13								
7	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					7	Δy_{BC}	
8	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					8		17
		9	x _{CD}					
		10	Δx_{CD}					
		11	Δy_{CD}					
		12						
		13	x _{AD}					
		14	Δx_{AD}					
		15	Δy_{AD}					
		16						
		17	x _{BD}					
		18	Δx_{BD}					
		19	Δy_{BD}					
		20						

d

д



е

у-группа

Связный список

1	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					1	xBA	
2	<table><tr><td colspan="2">1</td></tr></table>	1		2	Δ xBA			
1								
3	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					3	Δ yBA	
4	<table><tr><td colspan="2">9</td></tr></table>	9		4		17		
9								
5	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					5	xBc	
6	<table><tr><td colspan="2">13</td></tr></table>	13		6	Δ xBc			
13								
7	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					7	Δ yBc	
8	<table><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>					8		17
		9	xCD					
		10	Δ xCD					
		11	Δ yCD					
		12						
		13	xAD					
		14	Δ xAD					
		15	Δ yAD					
		16						
		17	xBd					
		18	Δ xBd					
		19	Δ yBd					
		20						

f

ф

Рис. 2.19. Продолжение.

расти, если только «потерянные» ячейки (с 5 по 8 на рис. 2.19, f) не будут снова использованы или список не будет сжат. Дополнительную информацию о связанных списках и структурах данных можно найти, например, в [2-12].

Так как алгоритм, работающий в таких жестких ограничениях на время обработки одного видеокadra программно реализовать трудно, то успешные программные реализации используются главным образом в имитационных системах, таких, как летные тренажеры, навигационные тренажеры для кораблей и т. п.

2.9. ГРУППОВОЕ КОДИРОВАНИЕ

В методе группового кодирования сделана попытка воспользоваться тем, что большие области изображения имеют одинаковую интенсивность или цвет. При простейшем групповом кодировании определяется только интенсивность и количество последовательных пикселей с этой интенсивностью на данной сканирующей строке. На рис. 2.20, а показан простой черно-белый чертеж на растре 30×30 и соответствующие кодирующие последовательности для сканирующих строк с номерами 1, 15 и 30. Кодирующие данные следует рассматривать группами по два. Первое число — интенсивность, второе — число последовательных пикселей на сканирующей строке с этой интенсивностью:

Интенсивность	Длина участка
---------------	---------------

Таким образом, в строке 1 на рис. 2.20, а имеется 30 пикселей нулевой интенсивности, т. е. черных или фоновых. Все изображение можно закодировать с помощью 208 чисел. Попиксельное хранение, т. е. одна порция информации на каждый пиксел (битовая карта), потребовало бы 900 значений интенсивности для раstra 30×30 . В этом случае сжатие данных с помощью группового кодирования составляет $4.33 : 1$.

Легко обрабатываются с помощью данного метода сплошные фигуры, как это продемонстрировано на рис. 2.20, b с кодированием 1, 15 и 30 строк. Особый интерес представляет 15-я сканирующая строка. Все изображенные на рис. 2.20, b может быть закодировано с использованием 136 чисел при сжатии данных $6.62 : 1$. Большая степень сжатия изображений со сплошными фигурами по сравнению с сеточными рисунками объясняется тем, что два ребра покрываются одной парой «интенсивность — длина».

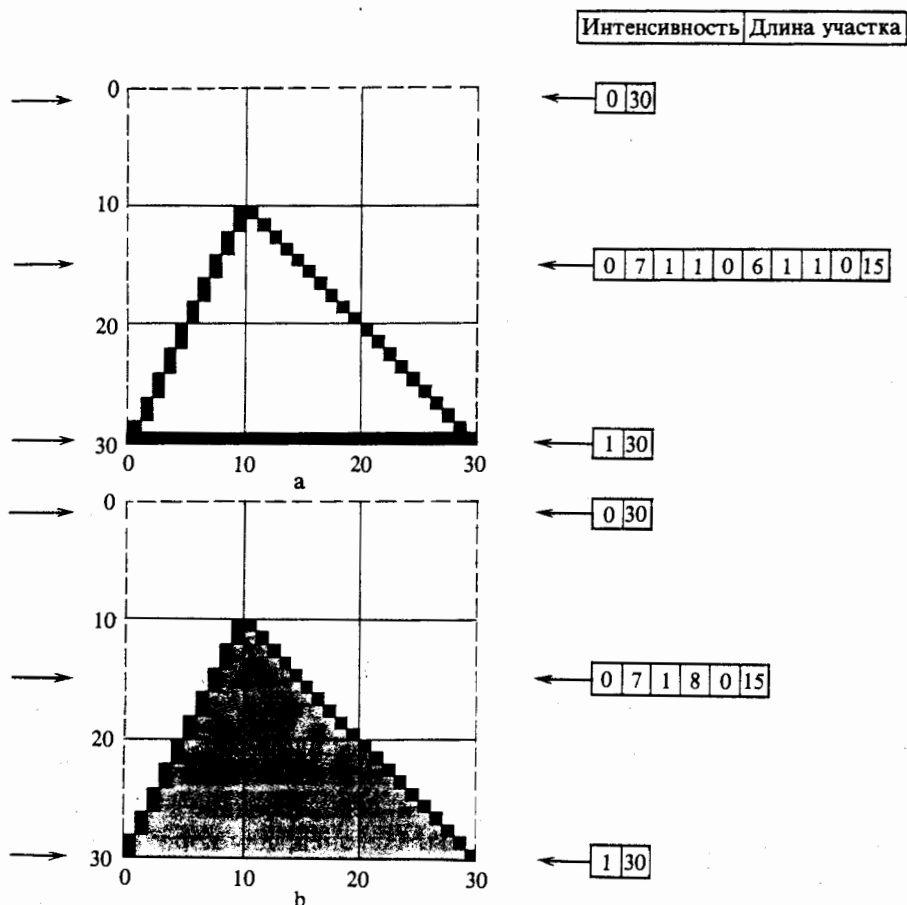


Рис. 2.20. Примеры группового кодирования.

Для добавления цвета эта простая схема группового кодирования может быть легко расширена. На данной сканирующей строке для цвета приводятся интенсивности красной, зеленой и синей цветных пушек, а за ними — количество последовательных пикселей с этим цветом; например,

Интенсивность красного	Интенсивность зеленого	Интенсивность синего	Длина
---------------------------	---------------------------	-------------------------	-------

Для простого цветного дисплея, в котром цветовая пушка либо выключена (0), либо включена (1), кодирование 15 строки

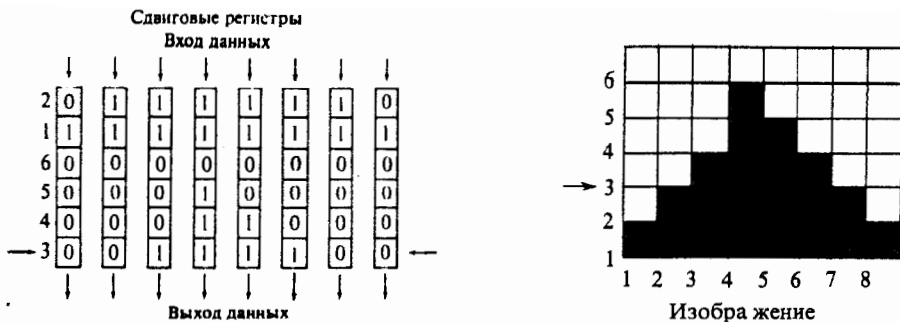


Рис. 2.25. Буфер кадра на сдвиговых регистрах.

равной числу пикселей в сканирующей строке, умноженному на число строк.

На рис. 2.25 показан простой шестистрочный дисплей по восемь пикселей на строке. На рисунке изображен также буфер кадра на сдвиговых регистрах, в который входят восемь регистров по 6 бит каждый. В буфере находится набор битов изображения. Биты для строки 3 показаны в момент выталкивания со дна сдвиговых регистров. Для согласованности со скоростью видеогенерации следует тщательно управлять последовательностью вывода сдвиговых регистров.

Для буферов кадра на вторичной памяти и на сдвиговых регистрах уровень интерактивности не высок. Для вторичной памяти причина заключается в большом времени доступа, а для сдвиговых регистров снижение эффективности интерактивной работы связано с тем, что изменения могут быть сделаны только при добавлении битов в регистр.

Как показано на рис. 2.26, схема графической системы с буфером кадра похожа на схему для векторного дисплея с регенерацией. При необходимости прикладная программа на главном компьютере модифицирует буфер кадра. Дисплейный контроллер периодически обрабатывает его в порядке сканирования строк и передает видеомонитору информацию, необходимую для регенерации изображения. Буфер кадра можно реализовать либо как часть памяти главного компьютера, либо как отдельную память. На рис. 2.27 показаны две эти схемы, реализованные со структурой общей шины. Хо-



Рис. 2.26. Графическая система с буфером кадра.

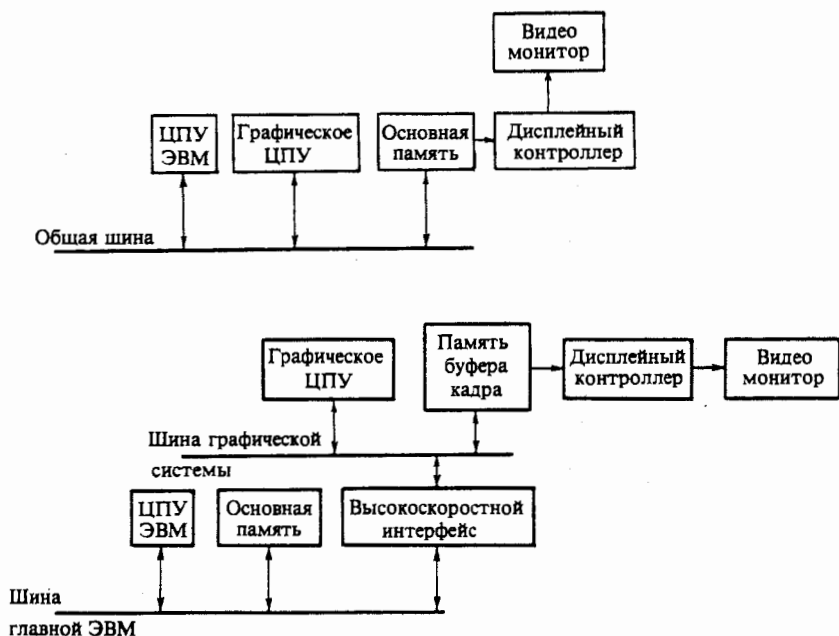


Рис. 2.27. Архитектура графических систем с буфером кадра.

Первая схема позволяет процессору ЭВМ самому манипулировать буфером кадра (рис. 2.27, а), обычно более эффективно добавив к системе специализированный графический процессор. При получении команд от главного процессора графический процессор управляет детальной обработкой буфера кадра. При двух процессорах на общей шине и одной памяти на шине может произойти конфликтная ситуация, что сокращает среднюю производительность системы. Таким образом, для высокопроизводительных систем более предпочтительна архитектура, показанная на рис. 2.27, б. В этом случае память буфера кадра отделена от основной, что исключает конфликт на шине. Более того, графическую подсистему можно оптимизировать для улучшения характеристик модификации буфера кадров и, следовательно, для увеличения производительности системы.

2.12. АДРЕСАЦИЯ РАСТРА

Для простоты изложения будем считать, что пиксел в растре или буфере кадра имеет двумерные координаты x и y , как это показано на рис. 2.28. Цифровая память, однако, организована в один линейный список адресатов, и необходимо, таким образом, преобразова-

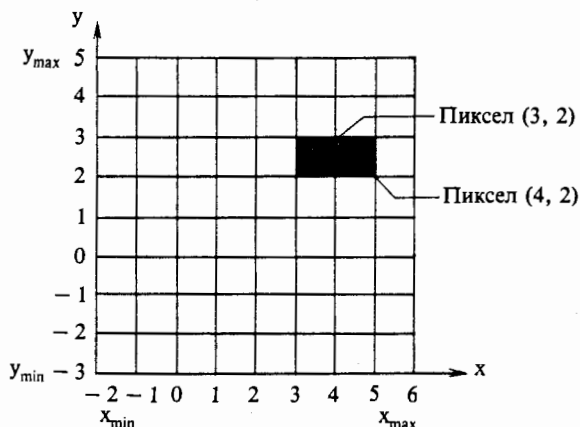


Рис. 2.28. Система координат растра.

ние координатного представления в линейное. Предположим, что начальный адрес в памяти не равен нулю, тогда преобразование задается формулой

$$\text{Адрес} = (x_{\max} - x_{\min})(y - y_{\min}) + (x - x_{\min}) + \text{базовый адрес}$$

В вычислении первого члена участвует число строк. Второй член добавляет адрес в строке, а последний — начальный адрес. Пиксел идентифицируется координатами своего левого нижнего угла.

Пример 2.6. Адресация растра.

Рассмотрим пиксел с координатами (3,2) в небольшом растре на рис. 2.28. Здесь $x_{\max} = 6$, $x_{\min} = -2$, $y_{\max} = 5$, $y_{\min} = -3$, причем первый пиксел из левого нижнего угла хранится в первой ячейке памяти; база или начальный адрес равен 1. Следовательно, адрес пиксела вычисляется по формуле

$$\begin{aligned} \text{Адрес} &= [6 - (-2)] \times [2 - (-3)] + [3 - (-2)] + 1 = (8) \times (5) + 5 + 1 = \\ &= 40 + 6 = 46 \end{aligned}$$

Данный результат можно проверить непосредственным подсчетом квадратов на рисунке.

Эта же схема работает и в случае, когда положительная ось x направлена вправо, а положительная ось y — вниз, при условии адресации пиксела относительно левого верхнего угла.

Как правило, для заданного буфера кадра величины x_{\max} , x_{\min} , y_{\min} и базовый адрес постоянны. Уравнение можно переписать в виде

$$\text{Адрес} = K_1 + K_2 y + x$$

где

$$K_1 = \text{базовый адрес} - K_2 y_{\min} - x_{\min}$$

$$K_2 = x_{\max} - x_{\min}$$

Вычисление адреса пиксела, следовательно, требует только двух сложений и одного умножения. При последовательной адресации пикселов для дальнейшего уменьшения работы, связанной с определением адреса, можно использовать пошаговые вычисления. В частности,

$$\text{Адрес}(x \pm 1, y) = K_1 + K_2 y + x \pm 1 = \text{Адрес}(x, y) \pm 1$$

$$\text{Адрес}(x, y \pm 1) = K_1 + K_2(y \pm 1) + x = \text{Адрес}(x, y) \pm K_2$$

$$\text{Адрес}(x \pm 1, y \pm 1) = K_1 + K_2(y \pm 1) + x \pm 1 = \text{Адрес}(x, y) \pm K_2 \pm 1$$

Здесь для горизонтального или вертикального приращения в растре требуется только одно сложение или вычитание, а для диагонального приращения — только два сложения или вычитания. Операция умножения полностью исключена из вычислений.

Пример 2.7. Пошаговая адресация раstra

Рассмотрим пиксел с координатами (4, 2) раstra на рис. 2.28. Здесь

$$K_2 = 6 - (-2) = 8$$

$$K_1 = 1 - (-8)(-3) - (-2) = 27$$

$$\text{Адрес} = 27 + (8)(2) + 4 = 47$$

Вспомнив результат для пиксела (3, 2) в предыдущем примере и используя пошаговые вычисления, получим

$$\text{Адрес}(x + 1, y) = \text{Адрес}(x, y) + 1$$

$$\text{Адрес}(4, 2) = 46 + 1 = 47$$

2.13. ИЗОБРАЖЕНИЕ ОТРЕЗКОВ

Подобная адресация буфера кадра позволяет обращаться с ним как с графическим дисплеем на запоминающей трубке. Сначала буфер кадра очищается или устанавливается в фоновую интенсивность или цвет. Вместо того чтобы записывать векторы прямо на экран дисплея, для разложения в растр отрезка применяется либо алгоритм Брезенхема, либо ЦДА и соответствующие пикселы записываются в буфер кадра. Когда изображение или кадр построены, дисплейный контроллер читает буфер кадра в порядке сканирования строк и выводит результат на видеомонитор.

Выборочное стирание отрезков можно реализовать с помощью повторного использования алгоритма разложения в растр и записи

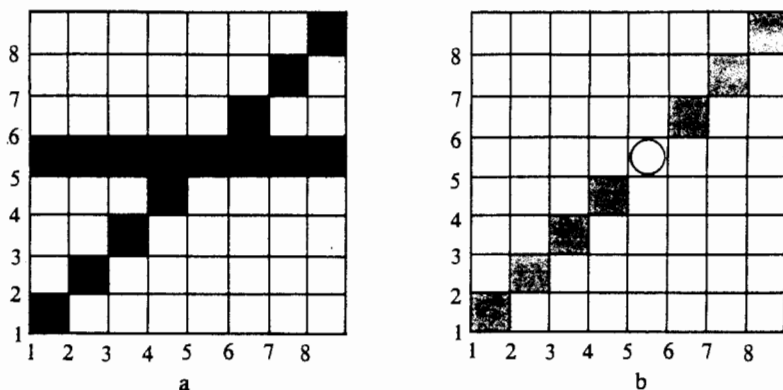


Рис. 2.29. Выборочное стирание отрезков в буфере кадра.

соответствующих пикселей с фоновой интенсивностью или цветом. Проблема, возникающая при использовании данного метода, иллюстрируется на рис. 2.29. Если удаляемый отрезок пересекает другой отрезок, то в последнем появится разрыв. На рис. 2.29, а показаны два пересекающихся отрезка. Если горизонтальный отрезок $y = 5$ стирается с помощью записи пикселей с фоновой интенсивностью или цветом в буфер кадра, то в результате в другом отрезке появится разрыв в пикселе (5, 5). Обнаружить и заполнить разрывы не составляет труда, надо только определить пересечение удаляемого отрезка со всеми другими отрезками в изображении. Данная операция для сложного изображения может занять много времени.

Для уменьшения затрат можно использовать оболочечный или минимаксный текст. Этот метод проиллюстрирован на рис. 2.30. Отрезок ab могут пересекать только те отрезки, которые проходят через нарисованную пунктиром прямоугольную оболочку, сформированную из минимальных и максимальных значений координат x ,

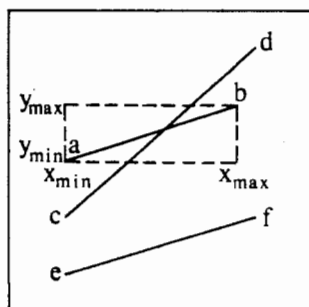


Рис. 2.30. Оболочечный или минимаксный тест.

у отрезка ab . Тесты для каждого отрезка выглядят следующим образом:

Минимаксный или оболочечный тест

```
if (Хотрmax < Хоболmin) or
   (Хотрmin > Хоболmax) or
   (Yотрmax < Yоболmin) or
   (Yотрmin > Yоболmax)
then
    пересечения нет
else
    вычислить пересечение
finish
```

2.14. ИЗОБРАЖЕНИЕ ЛИТЕР

Алфавитно-цифровые символы (литеры) записываются в буфер кадра с помощью маски. Литерная маска — это маленький растр, содержащий относительные адреса пикселей, используемых для представления литеры (см. рис. 2.22). С помощью литерной маски также можно представить и специальные символы, специфичные в конкретной прикладной области, например резисторы, конденсаторы или математические символы. Сама маска просто содержит двоичные величины, обозначающие, используется или нет конкретный пиксел в маске для представления формы буквы или символа. Для простых черно-белых изображений 1 обычно означает, что пиксел используется в представлении, а 0 — не используется. Для цветных изображений применяются дополнительные биты в качестве индексов в таблице цветов.

Литеру можно вставить в буфер кадра, указав адрес (x_0, y_0) начала маски в буфере. Каждый пиксел в маске смещается на величины x_0, y_0 . Простой алгоритм для бинарной маски приводится ниже.

Вставка маски в буфер кадра

$X_{min}, X_{max}, Y_{min}, Y_{max}$ — пределы маски

x_0, y_0 — адрес в буфере кадра

```
for j = Ymin to Ymax - 1
    for i = Xmin to Xmax - 1
```

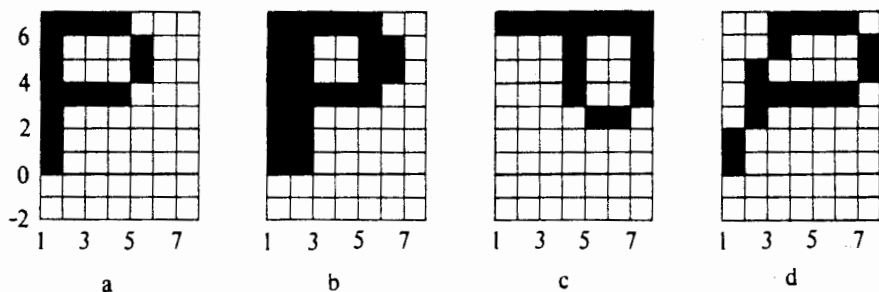


Рис. 2.31. Трансформированные маски литер.

```

if Маска(i, j) < > 0 then
    записать Маска(i, j) в буфер кадра в ( $x_0 + i$ ,  $y_0 + j$ )
else
end if
next i
next j
finish

```

Стереть литеру в буфере кадра можно, перезаписав ее с фоновой интенсивностью или цветом.

Для создания литер различных шрифтов или ориентаций перед записью в буфер маску можно модифицировать. Некоторые из таких простых модификаций показаны на рис. 2.31. На рис. 2.31, а изображена исходная литерная маска. Записывая ее в две последовательные ячейки x_0 и $x_0 + 1$, получим жирную литеру (рис. 2.31, б). Литеру можно повернуть (рис. 2.31, в) или наклонить, в результате последней операции получим курсив (рис. 2.31, д).

2.15. РАСТРОВАЯ РАЗВЕРТКА СПЛОШНЫХ ОБЛАСТЕЙ

Для сих пор речь шла о представлении на растровом графическом устройстве отрезков прямых линий. Однако одной из уникальных характеристик такого устройства является возможность представления сплошных областей. Генерацию сплошных областей из простых описаний ребер или вершин будем называть растровой разверткой сплошных областей, заполнением многоугольников или заполнением контуров. Для этого можно использовать несколько методов, которые обычно делятся на две широкие категории: растровая развертка и затравочное заполнение.

В методах растровой развертки пытаются определить в порядке

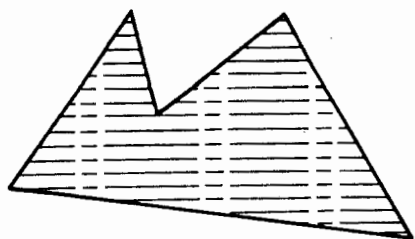


Рис. 2.32. Штриховка или закрашка контура.

сканирования строк, лежит ли точка внутри многоугольника или контура. Эти алгоритмы обычно идут от «верха» многоугольника или контура к «низу». Методы развертки также применимы и к векторным дисплеям, в которых они используются для штриховки или закрашки контуров, как показано на рис. 2.32.

В методах затравочного заполнения предполагается, что известна некоторая точка (затравка) внутри замкнутого контура. В алгоритмах ищут точки, соседние с затравочной и расположенные внутри контура. Если соседняя точка расположена не внутри, значит, обнаружена граница контура. Если же точка оказалась внутри контура, то она становится новой затравочной точкой и поиск продолжается рекурсивно. Подобные алгоритмы применимы только к растровым устройствам.

2.16. ЗАПОЛНЕНИЕ МНОГОУГОЛЬНИКОВ

Многие замкнутые контуры являются простыми многоугольниками. Если контур состоит из кривых линий, то его можно аппроксимировать подходящим многоугольником или многоугольниками. Простейший метод заполнения многоугольника состоит в проверке

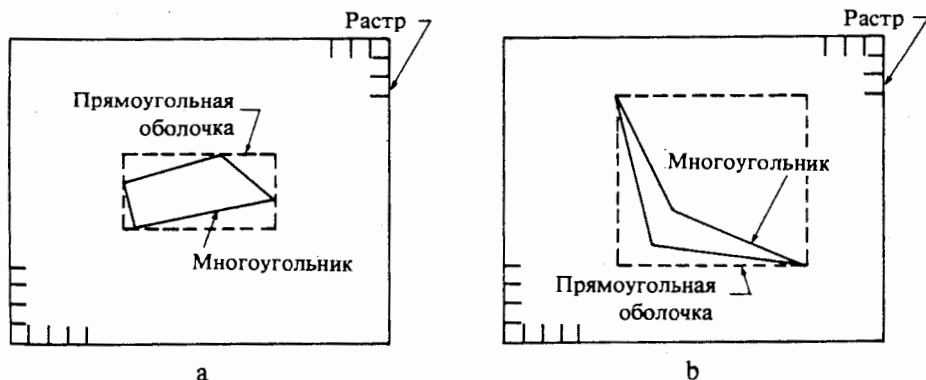


Рис. 2.33. Прямоугольная оболочка многоугольника.

на принадлежность внутренности многоугольника каждого пиксела в растре. Так как обычно большинство пикселей лежит вне многоугольника, то данный метод слишком расточителен. Затраты можно уменьшить путем вычисления для многоугольника прямоугольной оболочки — наименьшего прямоугольника, содержащего внутри себя многоугольник. Как показано на рис. 2.33, проверяются только внутренние точки этой оболочки. Использование прямоугольной оболочки для многоугольника, изображенного на рис. 2.33,а, намного сокращает число проверяемых пикселей. В то же время для многоугольника, представленного на рис. 2.33,б, сокращение существенно меньше.

2.17. РАСТРОВАЯ РАЗВЕРТКА МНОГОУГОЛЬНИКОВ

Можно разработать более эффективный метод, чем тест на принадлежность внутренней части, если воспользоваться тем фактом, что соседние пиксели, вероятно, имеют одинаковые характеристики (кроме пикселей граничных ребер). Это свойство называется пространственной когерентностью. Для растровых графических устройств соседние пиксели на сканирующей строке, вероятно, имеют одинаковые характеристики. Это когерентность растровых строк.

Характеристики пикселей на данной строке изменяются только там, где ребро многоугольника пересекает строку. Эти пересечения делят сканирующую строку на области.

Для простого многоугольника на рис. 2.34 строка 2 пересекает многоугольник при $x = 1$ и $x = 8$. Получаем три области:

$x < 1$	вне многоугольника
$1 \leq x \leq 8$	внутри многоугольника
$x > 8$	вне многоугольника

Строка 4 делится на пять областей:

$x < 1$	вне многоугольника
$1 \leq x \leq 4$	внутри многоугольника
$4 < x < 6$	вне многоугольника
$6 \leq x \leq 8$	внутри многоугольника
$x > 8$	вне многоугольника

Совсем необязательно, чтобы точки пересечения для строки 4 сразу определялись в фиксированном порядке (слева направо). Например, если многоугольник задается списком вершин $P_1 P_2 P_3 P_4 P_5$,

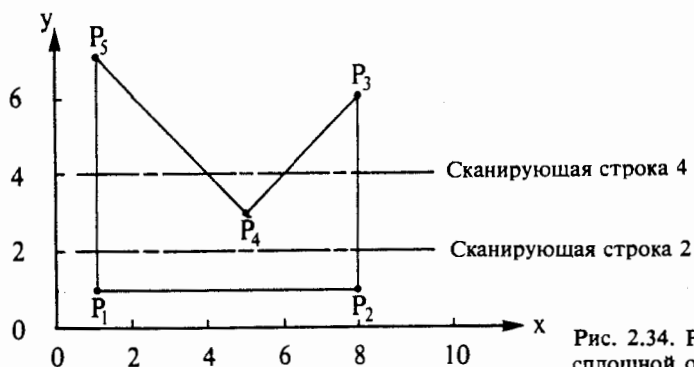


Рис. 2.34. Растровая развертка сплошной области.

а список ребер — последовательными парами вершин P_1P_2 , P_2P_3 , P_3P_4 , P_4P_5 , P_5P_1 , то для строки 4 будут найдены следующие точки пересечения с ребрами многоугольника: 8, 6, 4, 1. Эти точки надо отсортировать в возрастающем порядке по x , т. е. получить 1, 4, 6, 8.

При определении интенсивности, цвета и оттенка пикселей на сканирующей строке рассматриваются пары отсортированных точек пересечений. Для каждого интервала, задаваемого парой пересечений, используется интенсивность или цвет заполняемого многоугольника. Для интервалов между парами пересечений и крайних (от начала строки до первой точки пересечения и от последней точки пересечения до конца строки) используется фоновая интенсивность или цвет. На рис. 2.34 для строки 4 в фоновый цвет установлены пиксеты: от 0 до 1, от 4 до 6, от 8 до 10, тогда как пиксеты от 1 до 4 и от 5 до 8 окрашены в цвет многоугольника.

Точное определение тех пикселей, которые должны активироваться, требует некоторой осторожности. Рассмотрим простой прямоугольник, изображенный на рис. 2.35. Прямоугольник имеет

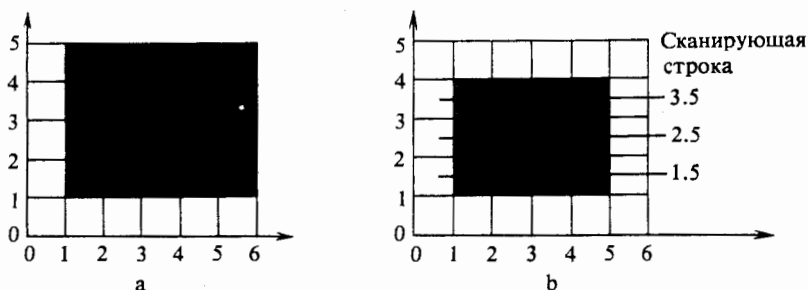


Рис. 2.35. Системы координат строк сканирования.

координаты (1,1), (5,1), (5,4), (1,4). Сканирующие строки с 1 по 4 имеют пересечения с ребрами многоугольника при $x = 1$ и 5. Вспомним, что пиксел адресуется координатами своего левого нижнего угла, значит, для каждой из этих сканирующих строк будут активированы пикселы с x -координатами 1, 2, 3, 4 и 5. На рис. 2.35, а показан результат. Заметим, что площадь, покрываемая активированными пикселями, равна 20, в то время как настоящая площадь прямоугольника равна 12.

Модификация системы координат сканирующей строки и теста активации устраняет эту проблему, как это показано на рис. 2.35, б. Считается, что сканирующие строки проходят через центр строк пикселов, т. е. через середину интервала, как это показано на рис. 2.35, б. Тест активации модифицируется следующим образом: проверяется, лежит ли внутри интервала центр пиксела, расположенного справа от пересечения. Однако пикселы все еще адресуются координатами левого нижнего угла. Как показано на рис. 2.35, б, результат данного метода корректен.

Горизонтальные ребра не могут пересекать сканирующую строку и, таким образом, игнорируются. Это совсем не означает, что их нет на рисунке. Эти ребра формируются верхней и нижней строками пикселов, как показано на рис. 2.35. Рис. 2.35 иллюстрирует корректность верхнего и нижнего ребер многоугольника, полученных в результате модификации системы координат сканирующих строк.

Дополнительная трудность возникает при пересечении сканирующей строки и многоугольника точно по вершине, как это показано на рис. 2.36. При использовании соглашения о середине интерва-

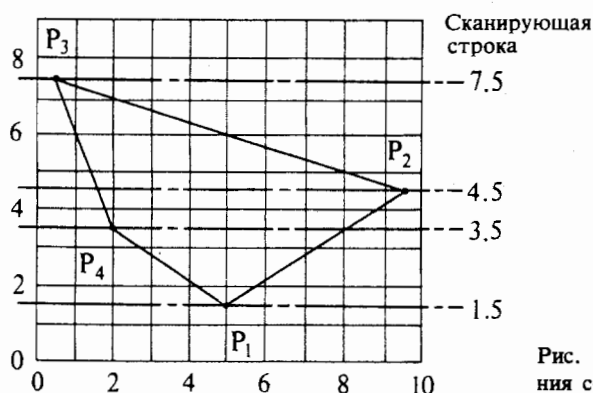


Рис. 2.36. Особенности пересечения со строками сканирования.

ла между сканирующими строками получаем, что строка $y = 3.5$ пересечет многоугольник в 2, 2 и 8, т. е. получится нечетное количество пересечений. Следовательно, разбиение пикселей на пары даст неверный результат, т. е. пиксели (0, 3), (1, 3) и от (3, 3) до (7, 3) будут фоновыми, а пиксели (2, 3), (8, 3), (9, 3) окрасятся в цвет многоугольника. Здесь возникает идея учитывать только одну точку пересечения с вершиной. Тогда для строки $y = 3.5$ получим правильный результат. Однако результат применения метода к строке $y = 1.5$, имеющей два пересечения в (5, 1), показывает, что метод неверен. Для этой строки именно разбиение на пары даст верный результат, т. е. окрашен будет только пиксел (5, 1). Если же учитывать в вершине только одно пересечение, то пиксели от (0, 1) до (4, 1) будут фоновыми, а пиксели от (5, 1) до (9, 1) будут окрашены в цвет многоугольника.

Правильный результат можно получить, учитывая точку пересечения в вершине два раза, если она является точкой локального минимума или максимума и учитывая ее один раз в противном случае. Определить локальный максимум или минимум многоугольника в рассматриваемой вершине можно с помощью проверки концевых точек двух ребер, соединенных в вершине. Если у обоих концов координаты y больше, чем у вершины, значит, вершина является точкой локального минимума. Если меньше, значит, вершина — точка локального максимума. Если одна больше, а другая меньше, следовательно, вершина не является ни точкой локального минимума, ни точкой локального максимума. На рис. 2.36 точка P_1 — локальный минимум, P_3 — локальный максимум, а P_2, P_4 — ни то и ни другое. Следовательно, в точках P_1 и P_3 учитываются два пересечения со сканирующими строками, а в P_2 и P_4 — одно.

2.18. ПРОСТОЙ АЛГОРИТМ С УПОРЯДОЧЕННЫМ СПИСКОМ РЕБЕР

Используя описанные выше методы, можно разработать эффективные алгоритмы растровой развертки сплошных областей, называемые алгоритмами с упорядоченным списком ребер. Они зависят от сортировки в порядке сканирования точек пересечений ребер многоугольника со сканирующими строками. Эффективность этих алгоритмов зависит от эффективности сортировки. Приведем очень простой алгоритм.

Простой алгоритм с упорядоченным списком ребер

Подготовить данные:

Определить для каждого ребра многоугольника точки пересечений со сканирующими строками, проведенными через середины интервалов, для чего можно использовать алгоритм Брезенхема или ЦДА. Горизонтальные ребра игнорируются. Занести каждое пересечение $(x, y + \frac{1}{2})$ в список.

Отсортировать список по строкам и по возрастанию x в строке; т. е. (x_1, y_1) предшествует (x_2, y_2) , если $y_1 > y_2$ или $y_1 = y_2$ и $x_1 \leq x_2$.

Преобразовать эти данные в растровую форму:

Выделить из отсортированного списка пары элементов (x_1, y_1) и (x_2, y_2) . Структура списка гарантирует, что $y = y_1 = y_2$ и $x_1 \leq x_2$. Активировать на сканирующей строке y пиксели для целых значений x , таких, что

$$x_1 \leq x + \frac{1}{2} \leq x_2.$$

Пример 2.8. Простой упорядоченный список ребер

Рассмотрим многоугольник, изображенный на рис. 2.34. Его вершины: $P_1(1,1)$, $P_2(8,1)$, $P_3(8,6)$, $P_4(5,3)$ и $P_5(1,7)$. Пересечения с серединами сканирующих строк следующие:

скан. строка 1.5: $(8, 1.5), (1, 1.5)$
скан. строка 2.5: $(8, 2.5), (1, 2.5)$
скан. строка 3.5: $(8, 3.5), (5.5, 3.5), (4.5, 3.5), (1, 3.5)$
скан. строка 4.5: $(8, 4.5), (6.5, 4.5), (3.5, 4.5), (1, 4.5)$
скан. строка 5.5: $(8, 5.5), (7.5, 5.5), (2.5, 5.5), (1, 5.5)$
скан. строка 6.5: $(1.5, 6.5), (1, 6.5)$
скан. строка 7.5: нет

Весь список сортируется в порядке сканирования сначала сверху вниз и затем — слева направо

$(1, 6.5), (1.5, 6.5), (1, 5.5), (2.5, 5.5), (7.5, 5.5), (8, 5.5), (1, 4.5), (3.5, 4.5),$
 $(6.5, 4.5), (8, 4.5), (1, 3.5), (4.5, 3.5), (5.5, 3.5), (8, 3.5), (1, 2.5), (8, 2.5),$
 $(1, 1.5), (8, 1.5)$

Выделяя из списка пары пересечений и применяя алгоритм, описанный выше, получим список пикселей, которые должны быть активированы:

$(1, 6)$
 $(1, 5), (2, 5), (7, 5)$
 $(1, 4), (2, 4), (3, 4), (6, 4), (7, 4)$
 $(1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3)$

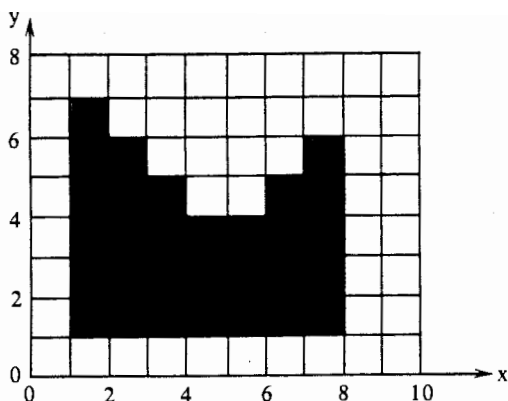


Рис. 2.37. Результат растровой развертки сплошной области, изображенной на рис. 2.34.

(1, 2), (2, 2), (3, 2) (4, 2), (5, 2), (6, 2), (7, 2) (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)

Результат представлен на рис. 2.37. Заметим, что оба вертикальных и нижнее ребра изображены верно.

2.19. БОЛЕЕ ЭФФЕКТИВНЫЕ АЛГОРИТМЫ С УПОРЯДОЧЕННЫМ СПИСКОМ РЕБЕР

В алгоритме, описанном в предыдущем разделе, генерируется большой список, который необходимо полностью отсортировать. Алгоритм можно улучшить, если повысить эффективность сортировки. Последнее достигается разделением сортировки по строкам в направлении y и сортировки в строке в направлении x с помощью групповой сортировки по y , описанной в разд. 2.8. В частности, алгоритм теперь можно представить в следующем виде:

Более эффективный алгоритм с упорядоченным списком ребер

Подготовить данные:

Определить для каждого ребра многоугольника точки пересечений со сканирующими строками, проведенными через середины интервалов, т. е. через $y + \frac{1}{2}$. Для этого можно использовать алгоритм Брезенхема или ЦДА. Горизонтальные ребра не учитывать. Поместить координату x точки пересечения в группу, соответствующую y .

Для каждой y -группы отсортировать список координат x точек пересечений в порядке возрастания; т. е. x_1 предшествует x_2 , если $x_1 \leq x_2$

Преобразовать эти данные в растровую форму:

Для каждой сканирующей строки выделить из списка коор-

динат x точек пересечений пары точек пересечений. Активировать на сканирующей строке y пиксели для целых значений x , таких, что $x_1 \leq x + \frac{1}{2} \leq x_2$.

В алгоритме сначала с помощью групповой сортировки по y происходит сортировка в порядке сканирования строк, а затем сортировка в строке. Таким образом, развертка начинается до завершения всего процесса сортировки. В таком алгоритме отчасти легче добавлять или удалять информацию из дисплейного списка. Необходимо только добавить или удалить информацию из соответствующих y -групп; следовательно, пересортировать придется только затронутые изменением строки. Ниже данный алгоритм иллюстрируется с помощью примера.

Пример 2.9. Более эффективный упорядоченный список ребер

Рассмотрим снова многоугольник на рис. 2.34, обсуждавшийся в примере 2.8. Сначала создаются y -группы для всех сканирующих строк, как это показано на рис. 2.38. Многоугольник обрабатывается против часовой стрелки, начиная с вершины P_1 , и получающиеся пересечения, несортированные по x , заносятся в соответствующие группы, как это показано на рис. 2.38,а. Пересечения были вычислены в соответствии с соглашением о середине интервала между сканирующими строками. В качестве иллюстрации на рис. 2.38,б показаны отсортированные пересечения. На практике можно использовать небольшой буфер сканирующей строки, содержащий отсортированные координаты x точек пересечения, как это показано на рис. 2.38,с. Такой буфер позволяет более эффективно добавлять или удалять пересечения в список, их просто добавлять к концу списка каждой y -группы, так как сортировка откладывается до момента помещения строки в буфер. Следовательно, не нужно держать полностью отсортированный список y -групп.

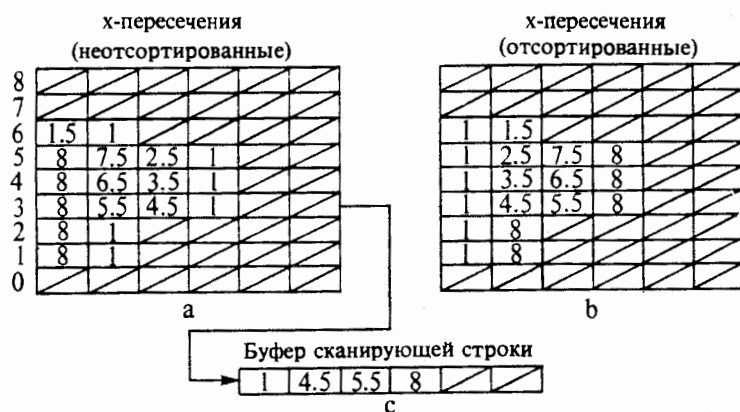


Рис. 2.38. y -группы сканирующих строк для многоугольника, изображенного на рис. 2.34.

В результате выделения пар пересечений из отсортированного по x списка и применения алгоритма для каждой сканирующей строки получим список активируемых пикселей. Результат показан на рис. 2.37 и совпадает с результатом примера 2.8.

Хотя в этой версии алгоритма задача сортировки упрощается, в ней либо ограничено число пересечений с данной сканирующей строкой, либо требуется резервирование большого количества памяти, значительная часть которой не будет использоваться. Этот недостаток можно преодолеть благодаря использованию связного списка, т. е. путем введения добавочной структуры данных. Предварительное вычисление пересечения каждой сканирующей строки с каждым ребром многоугольника требует больших вычислительных затрат и значительных объемов памяти. Введя список активных ребер (САР), как это предполагалось для растровой развертки в реальном времени (см. разд. 2.8), можно сократить потребность в памяти и вычислять пересечения со сканирующими строками в пошаговом режиме. Алгоритм, получившийся в результате всех улучшений, выглядит следующим образом:

Алгоритм с упорядоченным списком ребер, использующий список активных ребер

Подготовить данные:

Используя сканирующие строки, проведенные через середины отрезков, т. е. через $y + \frac{1}{2}$, определить для каждого ребра многоугольника наивысшую сканирующую строку, пересекаемую ребром.

Занести ребро многоугольника в y -группу, соответствующую этой сканирующей строке.

Сохранить в связном списке значения: начальное значение координат x точек пересечения, Δy — число сканирующих строк, пересекаемых ребром многоугольника, и Δx — шаг приращения по x при переходе от одной сканирующей строки к другой.

Преобразовать эти данные в растровую форму:

Для каждой сканирующей строки проверить соответствующую y -группу на наличие новых ребер. Новые ребра добавить в САР.

Отсортировать координаты x точек пересечения из САР в порядке возрастания; т. е. x_1 предшествует x_2 , если $x_1 \leq x_2$.

Выделить пары точек пересечений из отсортированного по x списка. Активировать на сканирующей строке y пиксели для целых значений x , таких, что $x_1 \leq x + \frac{1}{2} \leq x_2$. Для каждого ребра из CAP уменьшить Δy на 1. Если $\Delta y < 0$, то исключить данное ребро из CAP. Вычислить новое значение координат x точек пересечения $x_{\text{нов}} = x_{\text{стар}} + \Delta x$.

Перейти к следующей сканирующей строке.

В алгоритме предполагается, что все данные предварительно преобразованы в представление, принятое для многоугольников. Уиттед в [2-21] приводит более общий алгоритм, в котором данное ограничение устранено.

Пример 2.10. Упорядоченный список ребер вместе с CAP

Опять рассмотрим простой многоугольник на рис. 2.34. Проверка списка ребер многоугольника показывает, что сканирующая строка 5 — наивысшая для ребер P_2P_3 и P_3P_4 , а строка 6 — для ребер P_4P_5 и P_5P_1 . На рис. 2.39,а схематично показана структура связанного списка, содержащая данные для девяти y -групп, соответствующих девяти (с 0 по 8) сканирующим строкам рис. 2.34. Заметим, что большинство y -групп пусто. На рис. 2.39,б показан подход, применяемый на практике. Здесь список y -групп — это одномерный массив, по одному элементу на каждую сканирующую строку. В этом элементе содержится только указатель на массив данных, используемый для связанного списка. Массив показан на том же рисунке.

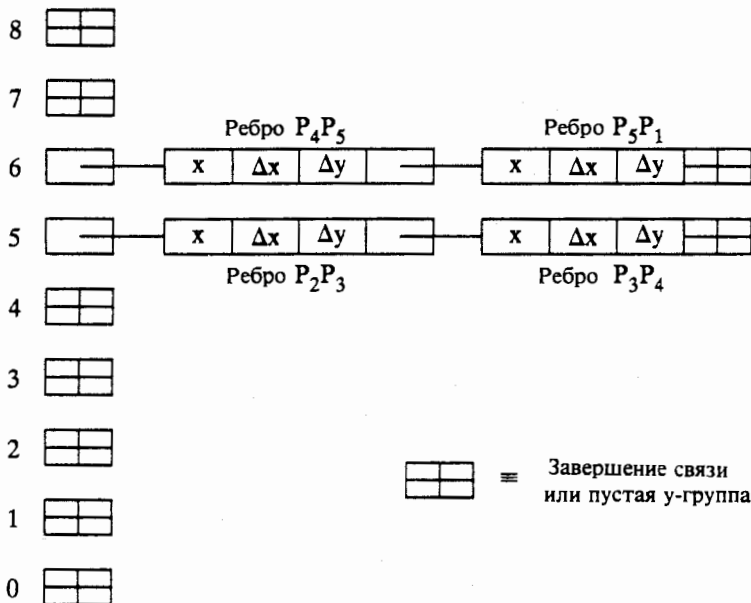
Связный список реализован как массив размерности $n \times 4$. Для каждого индекса массива n четыре элемента содержат: значение координаты x точки пересечения ребра многоугольника с наивысшей строкой, пересекаемой этим ребром; шаг приращения по x при переходе от одной сканирующей строки к другой; число сканирующих строк, пересекаемых ребром многоугольника; указатель на расположение данных для следующего ребра, начинающегося на этой же сканирующей строке. Это продемонстрировано на рис. 2.39,б. Заметим, что y -группа для строки 5 содержит указатель 1, соответствующий первому адресу в связанном списке данных. Первые три колонки данных в списке содержат информацию о ребре P_2P_3 . Число в четвертой колонке — это указатель на следующий элемент данных.

Список активных ребер реализован в виде стекового массива размерности $n \times 3$. Содержимое CAP для всех девяти сканирующих строк показано на рис. 2.39,с. Сканирующие строки (y -группы) последовательно проверяются сверху вниз, начиная со строки 8. Так как y -группы для строк 8 и 7 пусты, то CAP также пуст. Строка 6 добавляет в CAP два элемента и строка 5 — еще два. На строке 2 значение Δy для ребер P_3P_4 и P_4P_5 становится меньше нуля. Следовательно, эти ребра удаляются из CAP. Аналогичным образом ребра P_2P_3 и P_5P_1 удаляются на строке 0. Наконец, заметим, что для строки 0 y -группа пуста, список активных ребер пуст и больше y -групп нет. Значит, работа алгоритма завершена.

Для каждой сканирующей строки выделяются координаты x точек пересечения активных ребер для этой строки, сортируются по возрастанию x и записываются в интервальный буфер, реализованный в виде массива $1 \times n$. Из этого буфера пересечения выделяются в пары, и затем, используя описанный алгоритм, определяется список активных пикселей. Объединенный для всех сканирующих строк список активных пикселей совпадает со списком из предыдущего примера. Результат снова представлен на рис. 2.37.

у-группа
сканирующей
строки

Связный список данных для ребер многоугольника



а

у-группа
сканирующей
строки

Указатель

Адрес
списка

Связный список

0	
1	
2	
3	
4	
5	1
6	3
7	
8	

	х	Δх	Δу	Указатель
1	1.5	1	3	2
2	1	0	5	
3	8	0	4	4
4	7.5	-1	2	

б

Рис. 2.39. Схема связанного списка для многоугольника, изображенного на рис. 2.34.

Сканирующая строка ' Список активных ребер
 x Δx Δy

Отсортированные по x пересечения

Список пикселей

8

7

6

1.5	1	3
1	0	5

→

1	1.5
---	-----

→

(1, 6)

5

2.5	1	2
1	0	4
8	0	4
7.5	-1	2

→

1	2.5	7.5	8
---	-----	-----	---

→

(1, 5), (2, 5), (7, 5)

4

3.5	1	1
1	0	3
8	0	3
6.5	-1	1

→

1	3.5	6.5	8
---	-----	-----	---

→

(1, 4), (2, 4), (3, 4),
(6, 4), (7, 4)

3

4.5	1	0
1	0	2
8	0	2
5.5	-1	0

→

1	4.5	5.5	8
---	-----	-----	---

→

(1, 3), (2, 3), (3, 3), (4, 3),
(5, 3), (6, 3), (7, 3)

2

1	0	1
8	0	1

→

1	8
---	---

→

(1, 2), (2, 2), (3, 2), (4, 2),
(5, 2), (6, 2), (7, 2)

1

1	0	0
8	0	0

→

1	8
---	---

→

(1, 1), (2, 1), (3, 1), (4, 1),
(5, 1), (6, 1), (7, 1)

0

Рис. 2.39. Продолжение.

2.20. АЛГОРИТМ ЗАПОЛНЕНИЯ ПО РЕБРАМ

Алгоритм с упорядоченным списком ребер очень мощный. Каждый пиксел изображения активируется только один раз, следовательно, минимизированы операции ввода/вывода. До вывода вычисляются концевые точки каждой группы или интервалы активных пикселов. Это позволяет использовать алгоритмы закраски вдоль этих участков для получения полностью раскрашенных изображений. Так как этот алгоритм не зависит от деталей ввода/вывода, то можно его сделать не зависящим от устройства. Главный недостаток алгоритма состоит в больших накладных расходах, связанных с поддержкой и сортировкой различных списков. В другом методе растровой развертки сплошных областей большинство из этих списков устранено. Этот метод называется алгоритмом заполнения по ребрам [2-22].

Описываемый ниже алгоритм очень прост.

Алгоритм заполнения по ребрам

Для каждой сканирующей строки, пересекающей ребро многоугольника в (x_1, y_1) , дополнить все пикселы, у которых центры лежат справа от (x_1, y_1) , т. е. для (x, y_1) , $x + \frac{1}{2} > x_1$.

Вычисление пересечений сканирующих строк с ребрами производится в соответствии с соглашением о середине интервала между сканирующими строками. К каждому ребру алгоритм применяется индивидуально, причем порядок обработки ребер многоугольника не важен. На рис. 2.40 продемонстрированы различные стадии растровой развертки сплошной области тестового многоугольника с рис. 2.34. Заметим, что в этом случае результат не совпадает с результатом для упорядоченного списка ребер. В частности, в данном алгоритме не активируются пикселы (5, 3), (6, 4), (7, 5); т. е. ребро P_3P_4 обработано иначе. Отличие возникает для тех пикселов, которые разделены строго пополам: половина находится внутри, а половина — вне многоугольника. В алгоритме с упорядоченным списком ребер эти пикселы всегда активируются, а в данном алгоритме они активируются только в том случае, если внутренняя часть многоугольника лежит слева от центра пиксела.

Наиболее удобно использовать описываемый алгоритм вместе с буфером кадра, что позволяет обрабатывать ребра многоугольника в совершенно произвольном порядке. При обработке каждого реб-

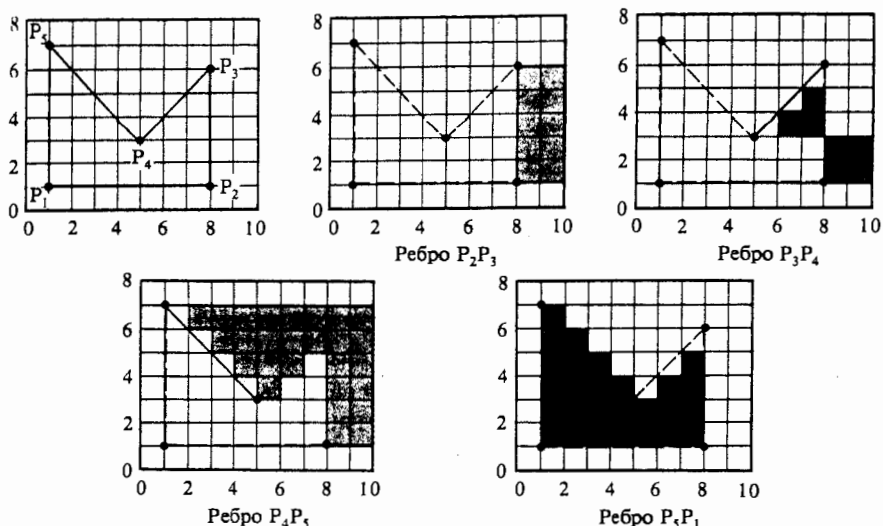


Рис. 2.40. Алгоритм заполнения по ребрам.

ра обрабатываются пиксели в буфере кадра, соответствующие пересечению ребра со сканирующей строкой. После завершения обработки всех ребер буфер кадра выводится в порядке сканирования на дисплей. На рис. 2.40 проиллюстрированы главные недостатки алгоритма — для сложного изображения каждый пиксел может обрабатываться много раз. Следовательно, эффективность алгоритма ограничена скоростью ввода/вывода.

Число обрабатываемых в данном алгоритме пикселей можно сократить, если ввести так называемую перегородку [2-23]. Основная идея алгоритма заполнения с перегородкой проиллюстрирована на рис. 2.41 для тестового многоугольника с рис. 2.34. Алгоритм можно описать следующим образом:

Алгоритм заполнения с перегородкой

Для каждой сканирующей строки, пересекающей ребро многоугольника:

Если пересечение находится слева от перегородки, то дополнить все пиксели, центры которых лежат справа от пересечения сканирующей строки с ребром и слева от перегородки.

Если пересечение находится справа от перегородки, то дополнить все пиксели, центры которых расположены слева

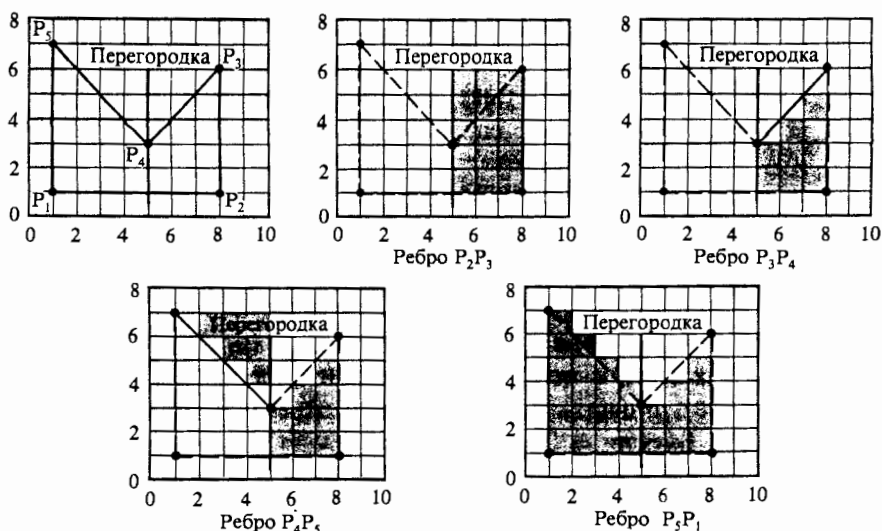


Рис. 2.41. Алгоритм заполнения с перегородкой.

или на пересечении сканирующей строки с ребром и справа от перегородки.

Используется соглашение о середине интервала между сканируемыми строками. Обычно перегородка проходит через одну из вершин многоугольника, и снова удобнее всего применять данный алгоритм с буфером кадра. Недостаток как алгоритма заполнения по ребрам, так и алгоритма заполнения с перегородкой заключается в неоднократном активировании части пикселей. Преодолеть его позволяет модификация алгоритма, известная как алгоритм со списком ребер и флагом [2-24]. Применение всех этих трех алгоритмов не ограничивается только простыми многоугольниками.

2.21. АЛГОРИТМ СО СПИСКОМ РЕБЕР И ФЛАГОМ

Алгоритм, использующий список ребер и флаг [2-24], является двухшаговым. Первый шаг состоит в обрисовке контура, в результате чего на каждой сканирующей строке образуются пары ограничивающих пикселей. Второй шаг состоит в заполнении пикселей, расположенных между ограничивающими. Более точно алгоритм можно сформулировать в следующем виде:

Алгоритм со списком ребер и флагом

Обрисовка контура:

Используя соглашение о середине интервала между сканирующими строками для каждого ребра, пересекающего сканируемую строку, отметить самый левый пиксел, центр которого лежит справа от пересечения; т. е.
 $x + 1/2 > x_{\text{пересечения}}$

Заполнение:

Для каждой сканирующей строки, пересекающей многоугольник

Внутри = FALSE

for $x = 0$ (левая граница) **to** $x = x_{\text{max}}$ (правая граница)

if пиксел в точке x имеет граничное значение

then инвертировать значение переменной **Внутри**

if **Внутри = TRUE then**

присвоить пикселу в x значение цвета многоугольника

else

присвоить пикселу в x значение цвета фона

end if

next x

Пример 2.11. Алгоритм, использующий список ребер и флаг

Рассмотрим применение данного алгоритма к тестовому многоугольнику на рис. 2.34. Сначала обрисовывается контур, результат этого шага показан на рис. 2.42, а. Активированы пикселы в точках (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (3, 5), (4, 4), (5, 3), (6, 3), (7, 4), (8, 4), (8, 3), (8, 2), (8, 1).

Затем многоугольник заполняется. Для иллюстрации этого процесса выделена и показана строка 3 на рис. 2.42, б. Для обрисовки контура на этой строке активированы пикселы при $x = 1, 5, 6$ и 8. Применение алгоритма заполнения дает следующие

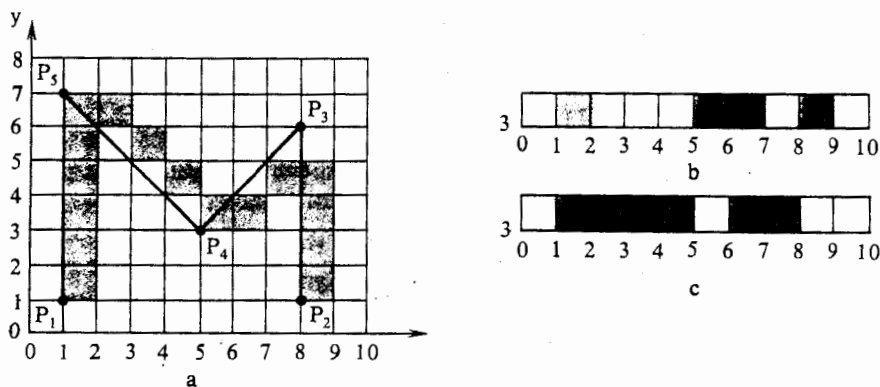


Рис. 2.42. Алгоритм заполнения по ребрам и флагу.

результаты:

Сначала

Внутри = FALSE

Для $x = 0$

Пиксел — не граничный и Внутри = FALSE, следовательно, предпринимать никаких действий не надо.

Для $x = 1$

Пиксел — граничный, поэтому переменная Внутри инвертируется и получает значение TRUE. Переменная Внутри = TRUE, поэтому пикселу присваивается цвет или интенсивность многоугольника.

Для $x = 2, 3, 4$

Пиксел — не граничный и Внутри = TRUE, поэтому пикселу присваивается цвет многоугольника.

Для $x = 5$

Пиксел — граничный, поэтому переменная Внутри инвертируется. Внутри = FALSE, поэтому пикселу присваивается фоновый цвет.

Для $x = 6$

Пиксел — граничный, поэтому переменная Внутри инвертируется и получает значение TRUE. Переменная Внутри = TRUE, поэтому пикселу присваивается цвет многоугольника.

Для $x = 7$

Пиксел — не граничный и Внутри = TRUE, поэтому пикселу присваивается цвет многоугольника.

Для $x = 8$

Пиксел — граничный, поэтому переменная Внутри инвертируется и получает значение FALSE. Переменная Внутри = FALSE, поэтому пикселу присваивается цвет фона.

Результат представлен на рис. 2.42,с, а конечный результат для всего многоугольника совпадает с результатом работы алгоритма со списком ребер, показанным на рис. 2.40.

В данном алгоритме каждый пиксел обрабатывается только один раз, так что затраты на ввод/вывод значительно меньше, чем в алгоритме со списком ребер или алгоритме с перегородкой. Ни один из этих алгоритмов, если они работают с буфером кадра, не требует построения, поддержки и сортировки каких-либо списков. При программной реализации алгоритм с упорядоченным списком ребер и алгоритм со списком ребер и флагом работают примерно с одинаковой скоростью [2-21]. Однако алгоритм со списком ребер и флагом годится для аппаратной или микропрограммной реализации, в результате чего он выполняется на один-два порядка быстрее, чем алгоритм с упорядоченным списком ребер [2-24]. Для простых изображений даже возможна анимация в реальном времени.

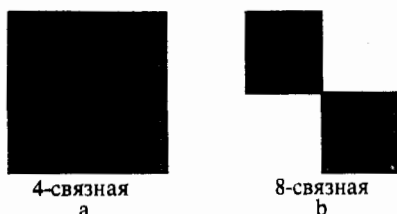


Рис. 2.45. 4- и 8-связные внутренне-определенные области.

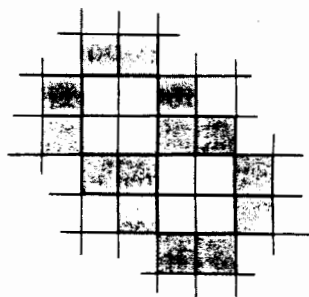


Рис. 2.46. 4- и 8-связные гранично-определенные области.

ях. Алгоритм заполнения 8-связной области заполнит и 4-связную область, однако обратное неверно. На рис. 2.45 показаны простые примеры 4- и 8-связных внутренне-определенных областей. Хотя каждая из подобластей 8-связной области на рис. 2.45, б является 4-связной, для перехода из одной подобласти в другую требуется 8-связный алгоритм. Однако в ситуации, когда надо заполнить разными цветами две отдельные 4-связные подобласти, использование 8-связного алгоритма вызовет неправильное заполнение обеих областей одним и тем же цветом.

На рис. 2.46 показана 8-связная область с рис. 2.45, переопределенная в виде гранично-определенной области. На рис. 2.46 иллюстрируется тот факт, что для 8-связной области, у которой две подобласти соприкасаются углами, граница 4-связна, а граница 4-связной области 8-связна. Далее речь в основном пойдет об алгоритмах для 4-связных областей, однако их можно легко переделать для 8-связных областей, если заполнение проводить не в 4, а в 8 направлениях.

2.23. ПРОСТОЙ АЛГОРИТМ ЗАПОЛНЕНИЯ С ЗАТРАВКОЙ

Используя стек, можно разработать простой алгоритм заполнения гранично-определенной области. Стек — это просто массив или другая структура данных, в которую можно последовательно помещать значения и из которой их можно последовательно извлекать. Когда новые значения добавляются или помещаются в стек, все остальные значения опускаются вниз на один уровень. Когда значения удаляются или извлекаются из стека, остальные значения всплывают или поднимаются вверх на один уровень. Такой стек называется стеком прямого действия или стеком с дисциплиной обслуживания «первым пришел, последним обслужен» (FILO). Про-

стой алгоритм заполнения с затравкой можно представить в следующем виде:

Простой алгоритм заполнения с затравкой и стеком

Поместить затравочный пиксел в стек

Пока стек не пуст

Извлечь пиксел из стека

Присвоить пикселу требуемое значение

Для каждого из соседних к текущему 4-связных пикселов проверить: является ли он граничным пикселом или не присвоено ли уже пикселу требуемое значение. Пропустить пиксел в любом из этих двух случаев. В противном случае поместить пиксел в стек.

Алгоритм можно модифицировать для 8-связных областей, если просматривать 8-связные пикселы, а не только 4-связные. Приведем более формальное изложение алгоритма, в котором предполагается существование затравочного пиксела и гранично-определенной области

Простой алгоритм заполнения

Затравка(x, y) *выдает затравочный пиксел*

Push — процедура, которая помещает пиксел в стек

Pop — процедура, которая извлекает пиксел из стека

Пиксел(x, y) = Затравка(x, y)

инициализируем стек

Push Пиксел(x, y)

while (стек не пуст)

извлекаем пиксел из стека

Pop Пиксел(x, y)

if Пиксел(x, y) < > Нов_значение **then**

Пиксел(x, y) = Нов_значение

end if

проверим, надо ли помещать соседние пикселы в стек

if (Пиксел($x + 1, y$) < > Нов_значение **and**

Пиксел($x + 1, y$) < > Гран_значение) **then**

Push Пиксел ($x + 1, y$)

if (Пиксел($x, y + 1$) < > Нов_значение **and**

Пиксел($x, y + 1$) < > Гран_значение) **then**

Push Пиксел ($x, y + 1$)

if (Пиксел($x - 1, y$) < > Нов_значение **and**

Пиксел($x - 1, y$) < > Гран_значение) **then**


```

Push Пиксел (x - 1, y)
If (Пиксел(x, y - 1) < > Нов_значение and
   Пиксел(x, y - 1) < > Гран_значение) then
  Push Пиксел (x, y - 1)
end If
end while

```

В алгоритме проверяются и помещаются в стек 4-связные пиксели, начиная с правого от текущего пикселя. Направление обхода пикселей — против часовой стрелки.

Пример 2.12. Простой алгоритм заполнения с затравкой

В качестве примера применения алгоритма рассмотрим гранично-определенную полигональную область, заданную вершинами (1, 0), (7, 0), (8, 1), (8, 4), (6, 6), (1, 6), (0, 5) и (0, 1). Область изображена на рис. 2.47. Затравочный пиксел — (4, 3). Область заполняется пиксел за пикселем в порядке, указанном на рис. 2.47 линией со стрелками. Числа, изображенные на пикселе, обозначают занимаемую при работе алгоритма позицию пикселя в стеке. Для некоторых пикселей таких чисел несколько. Это означает, что пиксел помещали в стек более одного раза. Когда алгоритм доходит до пикселя (5, 5), стек насчитывает 25 уровней глубины и содержит пиксели (7, 4), (7, 3), (7, 2), (7, 1), (6, 2), (6, 3), (5, 5), (6, 4), (5, 5), (4, 4), (3, 3), (3, 4), (3, 5), (2, 4), (2, 3), (2, 2), (2, 2), (3, 2), (5, 1), (3, 2), (5, 2), (3, 3), (4, 4), (5, 3).

Так как все пиксели, окружающие (5, 5), содержат либо граничные, либо новые значения цвета, то ни один из них в стек не помещается. Следовательно, из стека извлекается пиксел (7, 4) и алгоритм продолжает заполнять колонку (7, 4), (7, 3), (7, 2), (7, 1). При достижении пикселя (7, 1) проверка снова показывает, что окружающие пиксели либо уже заполнены, либо являются граничными пикселями. Так как в этот момент многоугольник полностью заполнен, то извлечение пикселей из стека до полного его опустошения не вызывает появления дополнительных пикселей, которые следует заполнить. Когда стек становится пустым, алгоритм завершает работу.

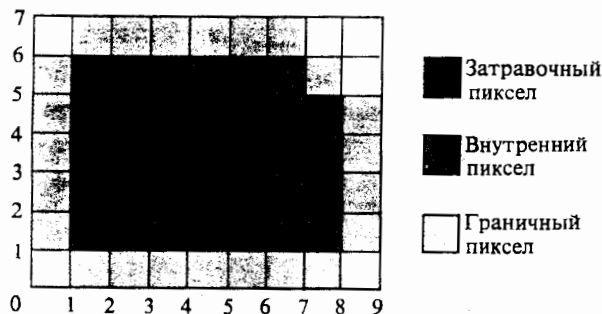


Рис. 2.47. Затравочное заполнение с помощью простого стекового алгоритма.

Многоугольник из примера 2.12 является односвязной областью, но алгоритм будет правильно заполнять и области, содержащие дыры. Это показано в следующем примере.

Пример 2.13. Алгоритм заполнения с затравкой для многоугольника с дыркой

В качестве примера применения алгоритма рассмотрим гранично-определенную область, содержащую дыру. Она изображена на рис. 2.48. Как и в предыдущем примере, вершины многоугольника заданы пикселями (1, 0), (7, 0), (8, 1), (8, 4), (6, 6), (1, 6), (0, 5) и (0, 1). Внутренняя дыра определяется пикселями (3, 2), (5, 2), (5, 3), (3, 3). Затравочный пиксел — (4, 4). Из-за дыры многоугольник заполняется не в том порядке, как в примере 2.12. Новый порядок заполнения указан на рис. 2.48 линией со стрелками. Как и в предыдущем примере, числа в квадратике пиксела показывают позицию в стеке, занимаемую пикселом. Когда обработка доходит до пиксела (3, 1), все окружающие его 4-связные пиксели либо уже заполнены, либо являются граничными. Поэтому ни один из пикселей не помещается в стек. Глубина стека в этот момент равна 15. В стеке находятся пиксели (7, 1), (7, 2), (7, 3), (6, 5), (7, 4), (6, 5), (3, 1), (1, 2), (1, 3), (1, 4), (2, 5), (3, 5), (4, 5), (5, 4).

После удаления из стека пиксела (7, 1) заполняется колонка (7, 1), (7, 2), (7, 3), (7, 4), при этом ни один новый пиксел в стек не добавляется. Для пиксела (7, 4) снова все 4-связные окружающие пиксели либо уже заполнены, либо являются граничными. Обращаясь к стеку, алгоритм извлекает пиксел (6, 5), его заполнение завершает заполнение всего многоугольника. Дальнейшая обработка происходит без какого-либо заполнения, и когда стек становится пустым, алгоритм завершает работу.

2.24. ПОСТРОЧНЫЙ АЛГОРИТМ ЗАПОЛНЕНИЯ С ЗАТРАВКОЙ

Как видно из обоих примеров, стек может стать довольно большим. Еще один недостаток предыдущего алгоритма — стек зачастую содержит дублирующую или ненужную информацию. В построочном алгоритме заполнения с затравкой размер стека минимизируется за счет хранения только одного затравочного пиксела для

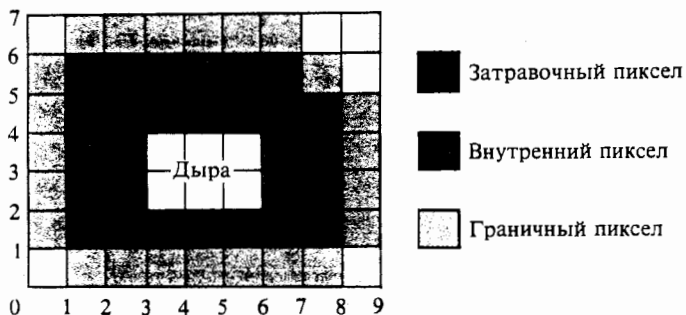


Рис. 2.48. Затравочное заполнение области с отверстием с помощью простого стекового алгоритма.

любого непрерывного интервала на сканирующей строке [2-25]. Непрерывный интервал — это группа примыкающих друг к другу пикселей (ограниченная уже заполненными или граничными пикселями). Мы для разработки алгоритма используем эвристический подход, однако также возможен и теоретический подход, основанный на теории графов [2-26].

Данный алгоритм применим к гранично-определенным областям. Гранично-определенная 4-связная область может быть как выпуклой, так и не выпуклой, а также может содержать дыры. В области, внешней и примыкающей к нашей гранично-определенной области, не должно быть пикселей с цветом, которым область или многоугольник заполняется. Схематично работу алгоритма можно разбить на четыре этапа.

Построчный алгоритм заполнения с затравкой

Затравочный пиксел на интервале извлекается из стека, содержащего затравочные пиксели.

Интервал с затравочным пикселем заполняется влево и вправо от затравки вдоль сканирующей строки до тех пор, пока не будет найдена граница.

В переменных $X_{лев}$ и $X_{прав}$ запоминаются крайний левый и крайний правый пиксели интервала.

В диапазоне $X_{лев} \leq x \leq X_{прав}$ проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется, есть ли на них еще не заполненные пиксели. Если такие пиксели есть (т. е. не все пиксели граничные, или уже заполненные), то в указанном диапазоне крайний правый пиксел в каждом интервале отмечается как затравочный и помещается в стек.

При инициализации алгоритма в стек помещается единственный затравочный пиксел, работа завершается при опустошении стека. Как показано на рис. 2.49 и в примере ниже, алгоритм справляется с дырами и зубцами на границе. Ниже приводится более подробное описание алгоритма на псевдокоде.

Построчный алгоритм заполнения с затравкой

Затравка (x, y) выдает затравочный пиксел

Pop — процедура, которая извлекает пиксел из стека

Отсечение

Отсечение, т. е. процесс выделения некоторой части базы данных, играет важную роль в задачах машинной графики. В гл. 2 было показано, что отсечение используется и для устранения ступенчатости, помимо своего более привычного применения для отбора той информации, которая необходима для визуализации конкретной сцены или вида, как части более обширной обстановки. В следующих главах будет показано, что отсечение применяется в алгоритмах удаления невидимых линий и поверхностей, при построении теней, а также при формировании фактуры. Алгоритмы и понятия, рассматриваемые здесь, можно применить для создания более совершенных алгоритмов, которые отсекают многогранники другими многогранниками, хотя эта задача и выходит за рамки данной книги. Такие алгоритмы можно использовать для реализации булевых операций, которые нужны в простых системах геометрического моделирования, например при вычислении пересечений и объединений простых тел, ограниченных плоскостями или поверхностями второго порядка. Получающиеся при этом приближенные решения годятся во многих приложениях.

Алгоритмы отсечения бывают дву- или трехмерными и применяются как к регулярным¹⁾, так и к нерегулярным областям и объемам. Эти алгоритмы можно реализовать аппаратно или программно. Алгоритмы отсечения, реализованные программно, зачастую

¹⁾ Имеются в виду канонические (стандартные) области и объемы. В частности, к ним относятся прямоугольники и параллелепипеды со сторонами, параллельными осям координат. В литературе такие объекты называют также изотетичными.—
Прим. ред.

оказываются недостаточно быстродействующими для приложений, ориентированных на процессы, протекающие в реальном времени. Поэтому как трех-, так и двумерные алгоритмы отсечения реализуются аппаратными или микропрограммными средствами. В подобных реализациях обычно ограничиваются дву- или трехмерными отсекателями типовых форм. Однако с появлением сверхбольших интегральных схем (СБИС) открываются возможности для более общих реализаций, позволяющих работать в реальном времени [3-1] как с регулярными, так и с нерегулярными областями и телами.

3.1. ДВУМЕРНОЕ ОТСЕЧЕНИЕ

На рис. 3.1 показана плоская сцена и отсекающее окно регулярной формы. Окно задается левым (Л), правым (П), верхним (В) и нижним (Н) двумерными ребрами. Регулярным отсекающим окном является прямоугольник, стороны которого параллельны осям координат объектного пространства или осям координат экрана. Целью алгоритма отсечения является определение тех точек, отрезков или их частей, которые лежат внутри отсекающего окна. Эти точки, отрезки или их части остаются для визуализации. А все остальное отбрасывается.

Поскольку в обычных сценах или картинках необходимо отсекалть большое число отрезков или точек, то эффективность алгоритмов отсечения представляет особый интерес. Во многих случаях подавляющее большинство точек или отрезков лежит целиком внутри или вне отсекающего окна. Поэтому важно уметь быстро отбирать отрезки, подобные ab , или точки, подобные p , и отбрасывать отрезки, подобные ij , или точки, подобные q на рис. 3.1.

Точки, лежащие внутри отсекающего окна, удовлетворяют условию: $x_{\text{Л}} \leq x \leq x_{\text{П}}$ и $y_{\text{Н}} \leq y \leq y_{\text{В}}$. Знак равенства здесь показывает,

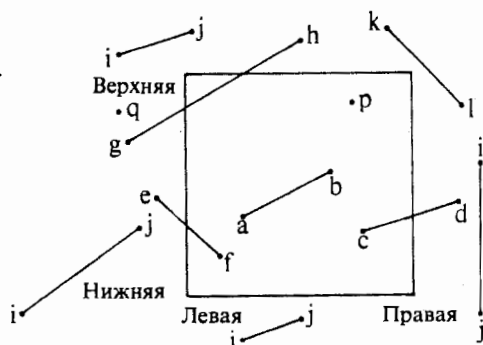


Рис. 3.1. Двумерное отсекающее окно.

что точки, лежащие на границе окна, считаются находящимися внутри него.

Отрезок лежит внутри окна и, следовательно, является видимым, если обе его концевые точки лежат внутри окна, например отрезок ab на рис. 3.1. Однако если оба конца отрезка лежат вне окна, то этот отрезок не обязательно лежит целиком вне окна, например отрезок gh на рис. 3.1. Если же оба конца отрезка лежат справа, слева, выше или ниже окна, то этот отрезок целиком лежит вне окна, а значит, невидим. Проверка последнего условия устранил все отрезки, помеченные ij на рис. 3.1. Но она не устранил ни отрезка gh , который видим частично, ни отрезка kl , который целиком невидим.

Пусть a и b — концы отрезка, тогда можно предложить алгоритм, определяющий все полностью видимые и большинство невидимых отрезков:

простой алгоритм определения видимости

a, b — концевые точки отрезка в координатном пространстве (x, y)

для каждого отрезка проверить полную видимость отрезка если одна из координат какого-нибудь конца отрезка находится вне окна, то отрезок не является полностью видимым

if $x_a < x_L$ or $x_a > x_P$ then 1

if $x_b < x_L$ or $x_b > x_P$ then 1

if $y_a < y_H$ or $y_a > y_B$ then 1

if $y_b < y_H$ or $y_b > y_B$ then 1

отрезок полностью видимый

визуализировать отрезок

go to 3

проверить полную невидимость отрезка

если оба конца отрезка лежат слева, справа, сверху или снизу от окна, то факт невидимости отрезка тривиален

1 if $x_a < x_L$ and $x_b < x_L$ then 2

if $x_a > x_P$ and $x_b > x_P$ then 2

if $y_a > y_B$ and $y_b > y_B$ then 2

if $y_a < y_H$ and $y_b < y_H$ then 2

отрезок частично видим или пересекает продолжение диагонали, оставаясь невидимым

определить пересечения отрезка с окном

- 2 отрезок невидим
- 3 переход к следующему отрезку

Здесь через x_L , x_P , y_B , y_H обозначены координаты x и y левого, правого, верхнего и нижнего краев окна соответственно. Порядок проведения сравнений при определении видимости или невидимости несуществен. Для некоторых отрезков может понадобиться проведение всех четырех сравнений, прежде чем определится их полная видимость или невидимость. Для других отрезков может потребоваться только одно сравнение. Несущественно также, что выявляется раньше — полностью видимые или полностью невидимые отрезки. Однако, поскольку определение пересечения отрезка с окном требует большого объема вычислений, его следует проводить в последнюю очередь.

Приведенные выше тесты полной видимости или невидимости отрезков можно формализовать, используя метод Д. Козна и А. Сазерленда. В этом методе для определения той из девяти областей, которой принадлежит конец ребра, вводится четырехразрядный (битовый) код. Коды этих областей показаны на рис. 3.2. Крайний правый бит кода считается первым. В соответствующий бит заносится 1 при выполнении следующих условий:

- для первого бита — если точка левее окна
- для второго бита — если точка правее окна
- для третьего бита — если точка ниже окна
- для четвертого бита — если точка выше окна

В противном случае в бит заносится нуль. Отсюда, очевидно, следует, что если коды обоих концов ребра равны нулю, то обе эти точки лежат внутри окна, и отрезок видимый. Коды концевых то-

	1001	1000	1010
В		Окно	
	0001	0000	0010
Н			
	0101	0100	0110
	Л	П	

Рис. 3.2. Коды областей, которым принадлежат концевые точки.

чек можно использовать также и для тривиального отбрасывания полностью невидимых отрезков. Рассмотрим таблицу истинности, эквивалентную логическому оператору «и»:

Истина и Ложь	→	Ложь	Ложь = 0	1 и 0 → 0
Ложь и Истина	→	Ложь		0 и 1 → 0
Ложь и Ложь	→	Ложь	Истина = 1	0 и 0 → 0
Истина и Истина	→	Истина		1 и 1 → 1

Если побитовое логическое произведение кодов концевых точек отрезка *не равно нулю*, то отрезок полностью невидим и его можно отбросить тривиально. Несколько примеров, приведенных в табл. 3.1, помогут прояснить высказанные утверждения. Из табл. 3.1 видно, что если результат логического умножения не равен нулю, то фактически отрезок будет целиком невидим. Однако если логическое произведение равно нулю, то отрезок может оказаться целиком или частично видимым или даже целиком невидимым. Поэтому для определения полной видимости необходимо проверять значения кодов обоих концов отрезка по отдельности.

Проверку значений кодов концов отрезка можно легко реализовать, если воспользоваться подпрограммами, оперирующими с битами. Однако в алгоритмах, которые рассматриваются ниже, такие подпрограммы не используются.

Если прежде всего найдены целиком видимые и тривиально невидимые отрезки, то подпрограмме, вычисляющей пересечение от-

Таблица 3.1. Коды концов отрезков

Отрезок (рис. 3.1)	Коды концов (рис. 3.2)	Результаты логического умножения	Примечания
<i>ab</i>	0000 0000	0000	Целиком видим
<i>ij</i>	0010 0110	0010	Целиком невидим
<i>ij</i>	1001 1000	1000	—"
<i>ij</i>	0101 0001	0001	—"
<i>ij</i>	0100 0100	0100	—"
<i>cd</i>	0000 0010	0000	Частично видим
<i>ef</i>	0001 0000	0000	—"
<i>gh</i>	0001 1000	0000	—"
<i>kl</i>	1000 0010	0000	Целиком невидим

резков, передаются только отрезки, которые, возможно, частично видимы, т. е. те, для которых результат логического умножения кодов их концевых точек равен нулю. Конечно же, эта подпрограмма должна правильно определять переданные ей такие целиком невидимые отрезки.

Пересечение двух отрезков можно искать как параметрическим, так и непараметрическим способами. Очевидно, что уравнение бесконечной прямой, проходящей через точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$, имеет вид $y = m(x - x_1) + y_1$ или $y = m(x - x_2) + y_2$, где $m = (y_2 - y_1)/(x_2 - x_1)$ — это наклон данной прямой. Точки пересечения этой прямой со сторонами окна имеют следующие координаты:

$$\text{с левой: } x_{\text{Л}}, y = m(x_{\text{Л}} - x_1) + y_1 \quad m \neq \infty$$

$$\text{с правой: } x_{\text{П}}, y = m(x_{\text{П}} - x_1) + y_1 \quad m \neq \infty$$

$$\text{с верхней: } y_{\text{В}}, x = x_1 + (1/m)(y_{\text{В}} - y_1) \quad m \neq 0$$

$$\text{с нижней: } y_{\text{Н}}, x = x_1 + (1/m)(y_{\text{Н}} - y_1) \quad m \neq 0$$

В примере 3.1 показано, как этот простой метод позволяет отбросить некорректные пересечения путем простого сравнения координат точек пересечения с координатами сторон окна.

Пример 3.1. Простое двумерное отсечение

Рассмотрим отсекающее окно и отрезки, изображенные на рис. 3.3.

Наклон отрезка от $P_1(-3/2, 1/6)$ до $P_2(1/2, 3/2)$ равен $m = (y_2 - y_1)/(x_2 - x_1) = (3/2 - 1/6) / [1/2 - (-3/2)] = 2/3$. Его пересечения со сторонами окна таковы:

$$\text{с левой: } x = -1 \quad y = (2/3)[-1 - (-3/2)] + 1/6 = 1/2$$

$$\text{с правой: } x = 1 \quad y = (2/3)[1 - (-3/2)] + 1/6 = 11/6$$

(последнее число больше, чем $y_{\text{В}}$, и поэтому отвергается)

$$\text{с верхней: } y = 1 \quad x = -3/2 + (3/2)[1 - 1/6] = -1/4$$

$$\text{с нижней: } y = -1 \quad x = -3/2 + (3/2)[-1 - (1/6)] = -13/4$$

(последнее число меньше, чем $x_{\text{П}}$, и поэтому отвергается)

Аналогично, отрезок от $P_3(-3/2, -1)$ до $P_4(3/2, 2)$ имеет

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{2 - (-1)}{3/2 - (-3/2)} = 1$$

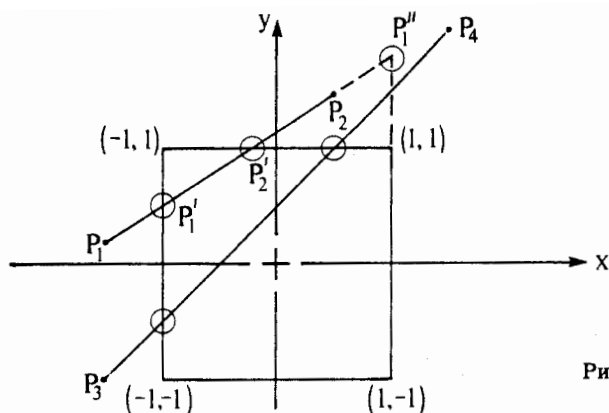


Рис. 3.3. Двумерное параметрическое отсечение

и пересечения:

$$\text{с левой: } x = -1 \quad y = (1)[-1 - (-3/2)] + (-1) = -1/2$$

$$\text{с правой: } x = 1 \quad y = (1)[1 - (-3/2)] + (-1) = 3/2$$

(последнее число больше, чем y_B , и поэтому отвергается)

$$\text{с верхней: } y = 1 \quad x = -3/2 + (1)[1 - (-1)] = 1/2$$

$$\text{с нижней: } y = -1 \quad x = -3/2 + (1)[-1 - (-1)] = -3/2$$

(последнее число больше, чем x_D , и поэтому отвергается).

Чтобы разработать схему эффективного алгоритма отсечения, необходимо сначала рассмотреть несколько частных случаев. Напомним, что, как уже указывалось, если наклон бесконечен, то отрезок параллелен левой и правой сторонам окна и надо искать его пересечения только с верхней и нижней сторонами. Аналогично, если наклон равен нулю, то отрезок параллелен верхней и нижней сторонам окна, а искать его пересечения надо только с левой и правой сторонами. Наконец, если код одного из концов отрезка равен нулю, то этот конец лежит внутри окна, и поэтому отрезок может пересечь только одну сторону окна. На рис. 3.4 приводится блок-схема алгоритма, использующего приведенные выше соображения. Ниже следует запись этого алгоритма на псевдокоде.

Простой алгоритм двумерного отсечения

P_1, P_2 — концевые точки отрезка

P'_1, P'_2 — концевые точки видимой части отрезка

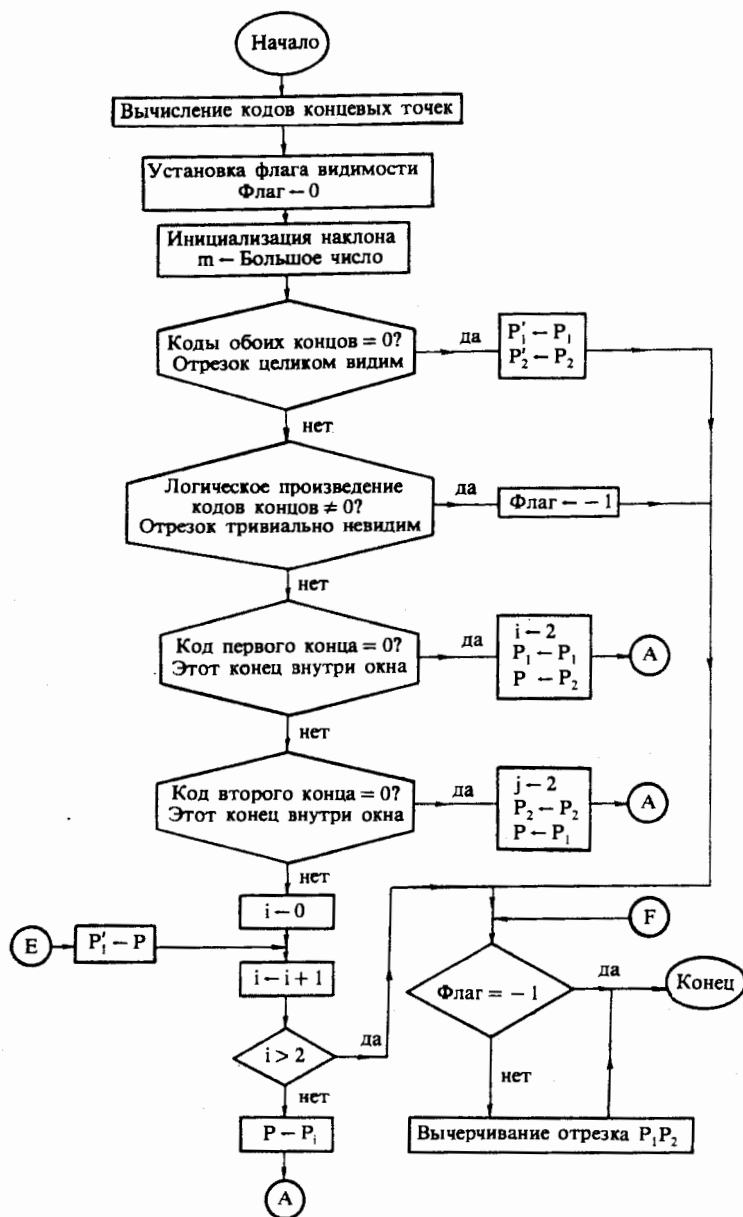


Рис. 3.4. Блок-схема алгоритма простого двумерного отсечения.

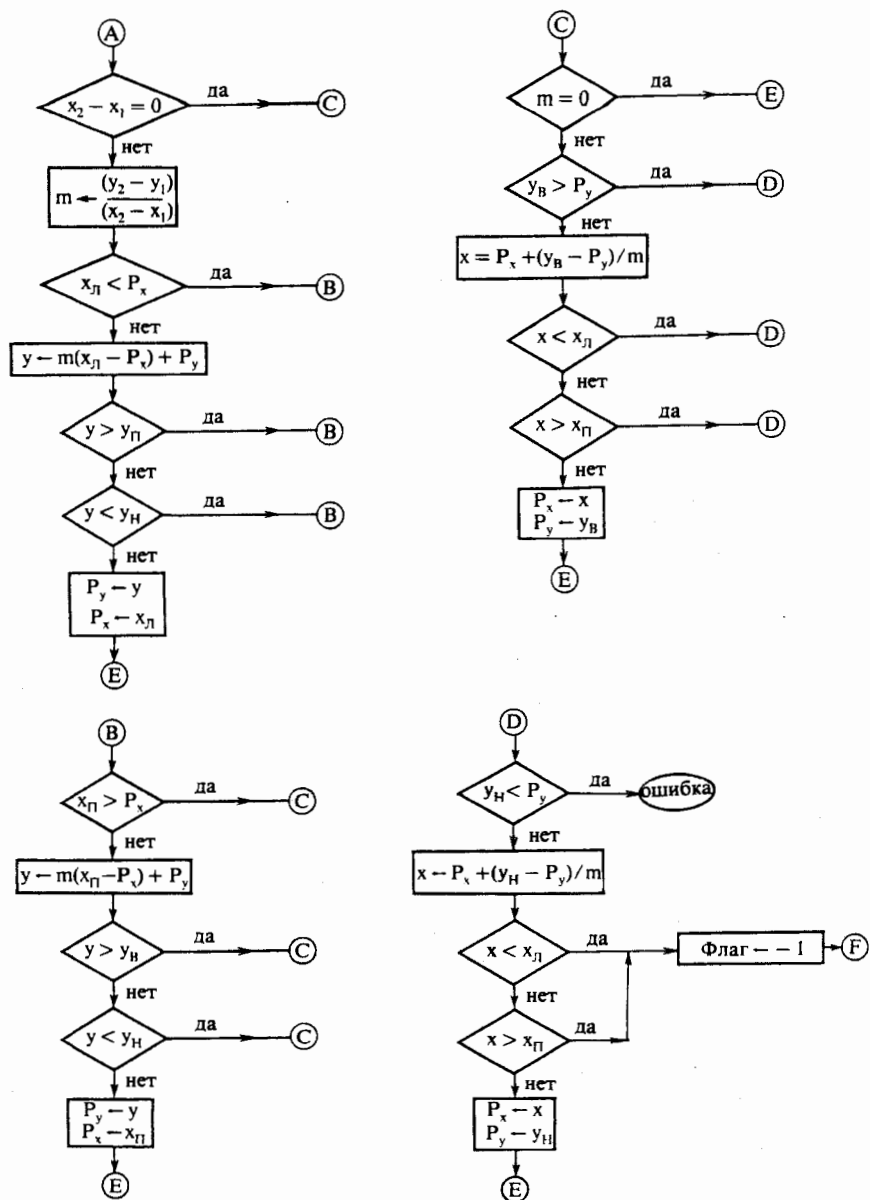


Рис. 3.4. Продолжение.

x_L, x_P, y_B, y_H — координаты левой, правой, верхней и нижней сторон окна

Флаг — признак видимости, равный: 0, видимость; -1, невидимость

вычисление кодов концевых точек

занесение этих кодов в два массива T1код и T2код, размерностью 1×4 каждый

для первого конца: P_1

if $x_1 < x_L$ then T1код(4) = 1 else T1код(4) = 0

if $x_1 > x_P$ then T1код(3) = 1 else T1код(3) = 0

if $y_1 < y_H$ then T1код(2) = 1 else T1код(2) = 0

if $y_1 > y_B$ then T1код(1) = 1 else T1код(1) = 0

для второго конца: P_2

if $x_2 < x_L$ then T2код(4) = 1 else T2код(4) = 0

if $x_2 > x_P$ then T2код(3) = 1 else T2код(3) = 0

if $y_2 < y_H$ then T2код(2) = 1 else T2код(2) = 0

if $y_2 > y_B$ then T2код(1) = 1 else T2код(1) = 0

инициализация признака видимости и видимых концевых точек

инициализация m очень большим числом, имитирующим бесконечный наклон

Флаг = 0

$P'_1 = P_1$

$P'_2 = P_2$

m = Большое число

проверка полной видимости отрезка

Сумма1 = 0

Сумма2 = 0

for i = 1 to 4

Сумма1 = Сумма1 + T1код(i)

Сумма2 = Сумма2 + T2код(i)

next i

if Сумма1 = 0 and Сумма2 = 0 then 7

отрезок не является полностью видимым

проверка случая тривиальной невидимости

вычисление логического произведения (Произвед) кодов концевых точек отрезка

Произвед = 0

for i = 1 to 4

Произвед = Произвед + Целая часть ((T1код(i) + T2код(i))/2)

if Произвед < > 0 then

Флаг = -1

```

        go to 7
    end if
next i
отрезок может быть частично видимым
проверка попадания первой точки внутрь окна
if Сумма1 = 0 then
    Номер = 1
     $P'_1 = P_1$ 
     $P = P_2$ 
    go to 2
end if
проверка попадания второй точки внутрь окна
if Сумма2 = 0 then
    Номер = 2
     $P'_1 = P_2$ 
     $P = P_1$ 
    go to 2
end if
внутри окна нет концов отрезка
инициализация номера конца отрезка
Номер = 0
1 if Номер < > 0 then  $P'_{\text{Номер}} = P$ 
    Номер = Номер + 1
    if Номер > 2 then 7
     $P = P_{\text{Номер}}$ 
    проверка пересечения с левым краем
    проверка вертикальности отрезка
2 if  $(x_2 - x_1) = 0$  then 4
     $m = (y_2 - y_1) / (x_2 - x_1)$ 
    if  $x_{\text{Л}} < P_x$  then 3
     $y = m * (x_{\text{Л}} - P_x) + P_y$ 
    if  $y > y_{\text{В}}$  then 3
    if  $y < y_{\text{Н}}$  then 3
    обнаружено корректное пересечение
     $P_y = y$ 
     $P_x = x_{\text{Л}}$ 
    go to 1
    проверка пересечения с правым краем
3 if  $x_{\text{П}} > P_x$  then 4
     $y = m * (x_{\text{П}} - P_x) + P_y$ 
    if  $y > y_{\text{В}}$  then 4

```

```

if  $y < y_H$  then 4
    обнаружено корректное пересечение
     $P_y = y$ 
     $P_x = x_{\Pi}$ 
    go to 1
    проверка пересечения с верхним краем
    проверка горизонтальности отрезка
4 if  $m = 0$  then 1
    if  $y_B > P_y$  then 5
     $x = (1/m) * (y_B - P_y) + P_x$ 
    if  $x < x_{\Pi}$  then 5
    if  $x > x_{\Pi}$  then 5
    обнаружено корректное пересечение
     $P_x = x$ 
     $P_y = y_B$ 
    go to 1
    проверка пересечения с нижним краем
5 if  $y_H < P_y$  then ошибка
     $x = (1/m) * (y_H - P_y) + P_x$ 
    if  $x < x_{\Pi}$  then 6
    if  $x > x_{\Pi}$  then 6
    обнаружено корректное пересечение
     $P_x = x$ 
     $P_y = y_H$ 
    go to 1
    отрезок невидим
6 Флаг = -1
    завершение работы и вызов процедуры черчения
7 if Флаг = -1 then 8
    Draw  $P_1'P_2'$ 
    перейти к обработке следующего отрезка
8 finish

```

3.2. АЛГОРИТМ ОТСЕЧЕНИЯ САЗЕРЛЕНДА — КОЭНА, ОСНОВАННЫЙ НА РАЗБИЕНИИ ОТРЕЗКА

Алгоритм, описанный в предыдущем разделе, аналогичен тому, который предложили Коэн и Сазерленд. В предыдущем алгоритме отрезок отсекался поочередно каждой из сторон окна, а для полученных точек пересечения проверялась их принадлежность внутренней области окна, т. е. корректность пересечения. Эта процедура

применялась сначала к отрезку P_1P_2 и получался отрезок P'_1P_2 , а затем к отрезку P'_1P_2 и получался результирующий отрезок $P'_1P'_2$.

В алгоритме Сазерленда — Козна отрезок тоже разбивается сторонами окна. Отличие состоит в том, что здесь не производится проверки попадания точки пересечения внутрь окна, вместо этого каждая из пары получающихся частей отрезка сохраняется или отбрасывается в результате анализа кодов ее концевых точек. Рассмотрение отрезка P_1P_2 на рис. 3.3 сразу же демонстрирует трудность реализации этой простой идеи. Если P_1P_2 разбивается левой стороной окна, то получаются два новых отрезка, $P_1P'_1$ и P'_1P_2 . Коды концевых точек каждого из этих отрезков таковы, что оба они могут быть частично видимы. Следовательно, ни один из них нельзя отвергнуть как невидимый или оставить как видимый. Ключом к алгоритму Сазерленда — Козна является информация о том, что одна из концевых точек отрезка лежит вне окна. Поэтому тот отрезок, который заключен между этой точкой и точкой пересечения, можно отвергнуть как невидимый. Фактически это означает замену исходной концевой точки на точку пересечения. В содержательных понятиях алгоритм Сазерленда — Козна формулируется следующим образом:

Для каждой стороны окна выполнить:

Для каждого отрезка P_1P_2 определить, не является ли он полностью видимым или может быть тривиально отвергнут как невидимый.

Если P_1 вне окна, то продолжить выполнение, иначе поменять P_1 и P_2 местами.

Заменить P_1 на точку пересечения P_1P_2 со стороной окна.

Этот алгоритм иллюстрирует следующий пример.

Пример 3.2. Алгоритм отсечения Сазерленда — Козна

Рассмотрим вновь отсечение отрезка P_1P_2 окном, показанным на рис. 3.3. Коды концевых точек $P_1(-\frac{1}{2}, \frac{1}{4})$ и $P_2(\frac{1}{2}, \frac{1}{2})$ равны (0001) и (1000) соответственно. Этот отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок пересекает левую сторону окна. P_1 — вне окна.

Пересечение с левой стороной ($x = -1$) окна происходит в точке $P'_1(-1, \frac{1}{2})$. Замена P_1 на P'_1 даст новый отрезок от $P'_1(-1, \frac{1}{2})$ до $P_2(\frac{1}{2}, \frac{1}{2})$.

Коды концевых точек P_1 и P_2 теперь стали (0000) и (1000) соответственно. Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок не пересекается с правой стороной окна. Перейти к нижней стороне.

Коды концевых точек P_1 и P_2 остаются по-прежнему равными (0000) и (1000) соответственно. Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок не пересекается с нижней стороной окна. Перейти к верхней стороне.

Коды концевых точек P_1 и P_2 остаются равными (0000) и (1000) соответственно. Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок пересекается с верхней стороной окна. P_1 — не снаружи окна. Поменяв P_1 и P_2 местами, мы получим новый отрезок от $P_1(1/2, 1/2)$ до $P_2(-1, 1/2)$.

Точка пересечения отрезка с верхней стороной окна ($y = 1$) равна $P'_1(-1/4, 1)$. Заменив P_1 на P'_1 , получаем новый отрезок от $P_1(-1/4, 1)$ до $P_2(-1, 1/2)$.

Теперь коды концевых точек P_1 и P_2 равны (0000) и (0000) соответственно. Отрезок полностью видим.

Процедура завершена.

Начертить отрезок.

Запись этого алгоритма на псевдокоде приводится ниже. Поскольку в алгоритме неоднократно используются одни и те же операции, то введены подпрограммы.

Алгоритм двумерного отсечения Сазерленда — Козна

Окно — массив 1×4 , содержащий координаты (x_L , x_P , y_N , y_B) сторон окна

P_1 , P_2 — концевые точки отрезка с координатами (P_1x , P_1y) и (P_2x , P_2y) соответственно

T1код, T2код — массивы 1×4 , содержащие коды точек P_1 и P_2

Флаг — индикатор координатной ориентации отрезка, равный:
-1, при вертикальности, 0, при горизонтальности

инициализация Флаг

Флаг = 1

проверка вертикальности и горизонтальности отрезка

if $P_2x - P_1x = 0$ then

 Флаг = -1

else

 вычисление наклона

 Наклон = $(P_2y - P_1y) / (P_2x - P_1x)$

 if Наклон = 0 then Флаг = 0

end if

для каждой стороны окна

for $i = 1$ **to** 4

call Козн (P_1 , P_2 , Окно; Видимость)

if Видимость = да **then** 2

if Видимость = нет **then** 3

проверка пересечения отрезка и стороны окна

if T1код($5 - i$) = T2код($5 - i$) **then** 1

проверка нахождения P_1 вне окна; если P_1 внутри окна, то поменять P_1 и P_2 местами

if T1код($5 - i$) = 0 **then**

$P_{a6} = P_i$

$P_1 = P_2$

$P_2 = P_{a6}$

end if

поиск пересечений отрезка со сторонами окна

выбор соответствующей подпрограммы вычисления пересечения

контроль вертикальности отрезка

if Флаг < > -1 и $i \leq 2$ **then**

$P_{1y} = \text{Наклон} * (\text{Окно}_i - P_{1x}) + P_{1y}$

$P_{1x} = \text{Окно}_i$

else

if Флаг < > 0 **then**

if Флаг < > -1 **then**

$P_{1x} = (1/\text{Наклон}) * (\text{Окно}_i - P_{1y}) + P_{1x}$

end if

$P_{1y} = \text{Окно}_i$

end if

end if

1 **next** i

начертить видимый отрезок

2 Draw P_1P_2

3 **finish**

подпрограмма определения видимости отрезка

subroutine Козн (P_1 , P_2 , Окно; Видимость)

P_1 , P_2 — концевые точки отрезка с координатами (P_{1x} , P_{1y}) и (P_{2x} , P_{2y}) соответственно.

Окно — массив 1×4 , содержащий координаты ($x_{\text{Л}}$, $x_{\text{П}}$, $y_{\text{Н}}$, $y_{\text{В}}$) сторон окна

Видимость — признак видимости отрезка равный: «нет»,

*«частично», «да», если отрезок соответственно:
полностью невидим, видим частично или полно-
стью видим*

вычисление кодов концевых точек отрезка

call Конец (P_1 , Окно; Т1код, Сумма1)

call Конец (P_2 , Окно; Т2код, Сумма2)

предположим, что отрезок частично видим

Видимость = частично

проверка полной видимости отрезка

if Сумма 1 = 0 и Сумма 2 = 0 **then**

Видимость = да

else

проверка тривиальной невидимости отрезка

call Логическое (Т1код, Т2код, Произвед)

if Произвед < > 0 **then** Видимость = нет

end if

отрезок может оказаться частично видимым

return

подпрограмма вычисления кодов концевой точки отрезка

subroutine Конец (Р, Окно; Ткод, Сумма)

P_x, P_y — координаты точки Р

Окно — массив 1×4 , содержащий координаты (x_L, x_P, y_H, y_B)
сторон окна

Ткод — массив 1×4 , содержащий коды концевой точки

Сумма — сумма всех элементов массива Ткод

вычисление кодов концевой точки

if $P_x < x_L$ **then** Ткод(4) = 1 **else** Ткод(4) = 0

if $P_x > x_P$ **then** Ткод(3) = 1 **else** Ткод(3) = 0

if $P_y < y_H$ **then** Ткод(2) = 1 **else** Ткод(2) = 0

if $P_y > y_B$ **then** Ткод(1) = 1 **else** Ткод(1) = 0

вычисление суммы

Сумма = 0

for i = 1 **to** 4

Сумма = Сумма + Ткод(i)

next i

return

подпрограмма вычисления логического произведения

subroutine Логическое (Т1код, Т2код; Произвед)

T1код, T2код — массивы 1×4 , содержащие коды первой и второй концевых точек соответственно

Произвед — сумма побитовых логических произведений

Произвед = 0

for i = 1 to 4

 Произвед = Произвед + Целая часть $((T1код(i) + T2код(i))/2)$

next i

return

3.3. АЛГОРИТМ РАЗБИЕНИЯ СРЕДНЕЙ ТОЧКОЙ

В алгоритме из предыдущего раздела требовалось вычислить пересечение отрезка со стороной окна. Можно избежать непосредственного вычисления, если реализовать двоичный поиск такого пересечения путем деления отрезка его средней точкой. Алгоритм, основанный на этой идее и являющийся частным случаем алгоритма Сазерленда — Козна, был предложен Спруллом и Сазерлендом [3-2] для аппаратной реализации. Программная реализация этого алгоритма медленнее, чем реализация предыдущего алгоритма. Аппаратная же реализация быстрее и эффективнее, поскольку можно использовать параллельную архитектуру, и, кроме того, аппаратные сложения и деления на 2 очень быстры. Деление на 2 аппаратно эквивалентно сдвигу каждого бита вправо. Например, четырехбитовый код десятичного числа 6 равен 0110. Побитовый сдвиг вправо на одну позицию дает 0011, что является кодом десятичного числа $3 = 6/2$.

В алгоритме используются коды концевых точек отрезка и проверки, выявляющие полную видимость отрезков, например отрезка

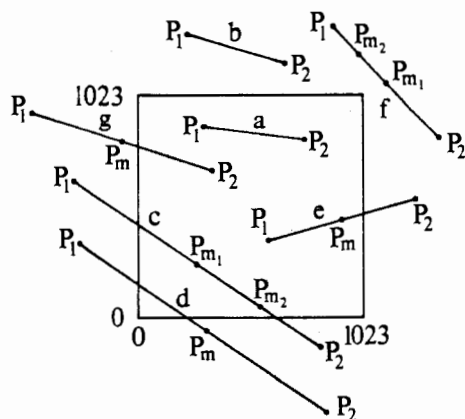


Рис. 3.5. Разбиение средней точкой.

a на рис. 3.5, и тривиальную невидимость отрезков, например b на рис. 3.5. Те отрезки, которые только с помощью таких простых проверок нельзя отнести к одной из двух категорий, например отрезки от c до g на рис. 3.5, разбиваются на две равные части. Затем те же проверки применяются к каждой из половин до тех пор, пока не будет обнаружено пересечение со стороной окна или длина разделяемого отрезка не станет пренебрежимо малой, т. е. пока он не выродится в точку. Эта идея проиллюстрирована на примере отрезка f с рис. 3.5. После вырождения определяется видимость полученной точки. В результате процесс обнаружения пересечения сводится к двоичному поиску. Максимальное число разбиений пропорционально точности задания координат концевых точек отрезка.

Для иллюстрации изложенного метода рассмотрим отрезки c и f с рис. 3.5. Хотя отрезок f невидим, он пересекает прямую, несущую диагональ окна, и не может быть тривиально отвергнут. Разбиение его средней точкой P_{m_1} позволяет тривиально отвергнуть половину $P_{m_1} P_2$. Однако половина $P_{m_1} P_1$ тоже пересекает диагональ окна, и ее нельзя отвергнуть тривиально. Далее проводится разбиение точкой P_{m_2} , которое позволяет отвергнуть невидимый отрезок $P_{m_2} P_1$. Разбиение оставшегося куска $P_{m_1} P_{m_2}$ продолжается до тех пор, пока не будет найдено пересечение этого отрезка с прямой, несущей правую сторону окна, с наперед заданной точностью. Затем исследуется обнаруженная точка и она оказывается невидимой. Следовательно, и весь отрезок невидим.

Если судить только по кодам концевых точек, то отрезок на рис. 3.5 также не является ни полностью видимым, ни тривиально невидимым. Разбиение его средней точкой P_{m_1} приводит к одинаковым результатам для обеих половин. Отложив отрезок $P_{m_1} P_{m_2}$ на потом, разобьем отрезок $P_{m_1} P_2$ точкой P_{m_2} . Теперь отрезок $P_{m_1} P_{m_2}$ полностью видим, а отрезок $P_{m_2} P_2$ видим частично. Отрезок $P_{m_1} P_{m_2}$ можно было бы начертить. Однако это привело бы к тому, что видимая часть отрезка изображалась бы неэффективно, как серия коротких кусков. Поэтому точка P_{m_2} запоминается как текущая видимая точка, которая наиболее удалена от P_1 . А разбиение отрезка $P_{m_2} P_2$ продолжается. Каждый раз, когда обнаруживается видимая средняя точка, она объявляется текущей наиболее удаленной от P_1 , до тех пор пока не будет обнаружено пересечение с нижней стороной окна с заранее заданной точностью. Это пересечение и будет объявлено самой удаленной от P_1 видимой точкой. Затем точно так же обрабатывается отрезок $P_{m_1} P_1$. Для отрезка c на рис. 3.5 наиболее удаленной от P_2 видимой точкой будет точка

его пересечения с левой стороной окна. Теперь можно начертить видимую часть отрезка $P_1 P_2$, заключенную между двумя найденными пересечениями.

Для отрезков, подобных c и d на рис. 3.5, в алгоритме разбиения средней точкой реализуется два двоичных поиска двух таких видимых точек, которые наиболее удалены от концов отрезка. Такими точками являются точки пересечения отрезка со сторонами окна. Каждое разбиение средней точкой дает грубую оценку искомым пересечениям. Для отрезков, у которых виден один из концов, подобных e и g , один из двух поисков становится ненужным. При программной реализации алгоритма указанные процедуры поиска проводятся последовательно. А при аппаратной реализации они проводятся параллельно. Данный алгоритм можно формально разбить на три этапа [3-3].

Для каждой концевой точки отрезка:

Если концевая точка видима, то она будет наиболее удаленной видимой точкой. Процесс завершен. Иначе — продолжить.

Если отрезок тривиально характеризуется как невидимый, то выходная информация не формируется. Процесс завершен. Иначе — продолжить.

Грубо оценить наиболее удаленную видимую точку путем деления отрезка $P_1 P_2$ средней точкой P_m . Применить вышеизложенные тесты к двум кускам $P_1 P_m$ и $P_m P_2$. Если $P_m P_2$ тривиально отвергается как невидимый, то средняя точка дает верхнюю оценку для наиболее удаленной видимой точки. Продолжить процедуру с отрезком $P_1 P_m$. Иначе — средняя точка дает оценку снизу для наиболее удаленной видимой точки. Продолжить процедуру с куском $P_2 P_m$. Если отрезок становится настолько мал, что его средняя точка совпадает с его концами с машинной или наперед заданной точностью, то надо оценить ее видимость и закончить процесс.

Конкретный пример лучше проиллюстрирует этот алгоритм.

Пример 3.3. Разбиение отрезка

Рассмотрим окно, показанное на рис. 3.5 и имеющее заданные координаты левой, правой, нижней и верхней сторон: 0, 1023, 0, 1023 соответственно. Заданные координаты концов отрезка c равны: $P_1(-307, 631)$ и $P_2(322, 631)$. Их двоичные представления равны: для P_1 — (0001), а для P_2 — (0100). Оба конца отрезка c находятся вне окна, отрезок не является полностью видимым. Логическое прохождение координатных тестов

чек равно (0000). Значит, отрезок нельзя тривиально отвергнуть как невидимый. Займемся поиском пересечений.

Координаты средней точки с учетом округления равны:

$$x_m = \frac{x_2 + x_1}{2} = \frac{820 - 307}{2} = 256.5 = 256$$

$$y_m = \frac{y_2 + y_1}{2} = \frac{-136 + 631}{2} = 247.5 = 247$$

Код средней точки равен (0000). Оба отрезка $P_1 P_m$ и $P_2 P_m$ не являются ни полностью видимыми, ни тривиально невидимыми. Отложим на время отрезок $P_2 P_m$ и займемся отрезком $P_1 P_m$. Процесс разбиения отрезков показан в табл. 3.2.

Таблица 3.2

P_1	P_2	P_m	Примечания
-307, 631	820, -136	256, 247	Запомнить $P_m P_2$, Продолжить с $P_1 P_m$
-307, 631	256, 247	-26, 439	Продолжить с $P_m P_2$
-26, 439	256, 247	115, 343	Продолжить с $P_1 P_m$
-26, 439	115, 343	44, 391	Продолжить с $P_1 P_m$
-26, 439	44, 391	9, 415	Продолжить с $P_1 P_m$
-26, 439	9, 415	-9, 427	Продолжить с $P_m P_2$
-9, 427	9, 415	0, 421	Найдено пересечение
256, 247	820, -136	538, 55	Восстановить $P_m P_2$, Продолжить с $P_m P_2$
538, 55	820, -136	679, -41	Продолжить с $P_1 P_m$
538, 55	679, -41	608, 7	Продолжить с $P_m P_2$
608, 7	679, -41	643, -17	Продолжить с $P_1 P_m$
608, 7	643, -17	625, -5	Продолжить с $P_1 P_m$
608, 7	625, -5	616, 1	Продолжить с $P_m P_2$
616, 1	625, -5	620, -2	Продолжить с $P_1 P_m$
616, 1	620, -2	618, -1	Продолжить с $P_1 P_m$
616, 1	618, -1	617, 0	Найдено пересечение

Использование точного уравнения отрезка $P_1 P_2$ дает пересечения в точках (0,422) и (620,0). Отличие этих значений от значений из табл. 3.2 объясняется погрешностью целочисленных округлений.

Блок-схема изложенного алгоритма дана на рис. 3.6. Запись его на псевдокоде приведена ниже.

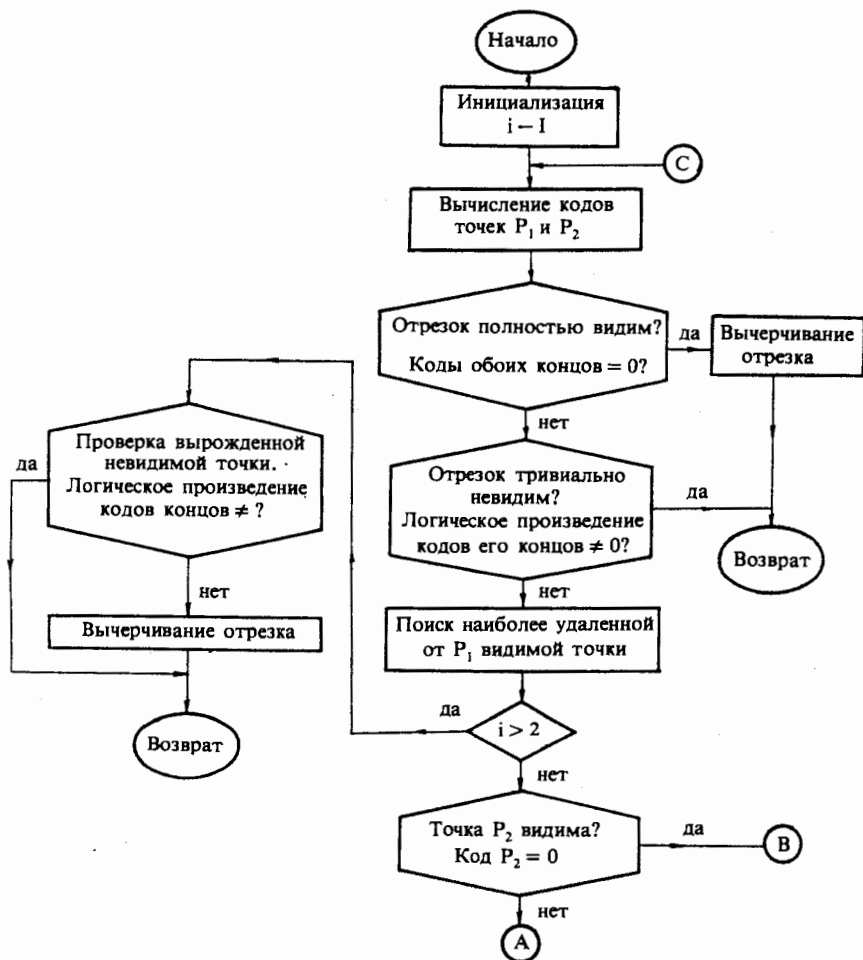


Рис. 3.6. Блок-схема алгоритма разбиения средней точкой.

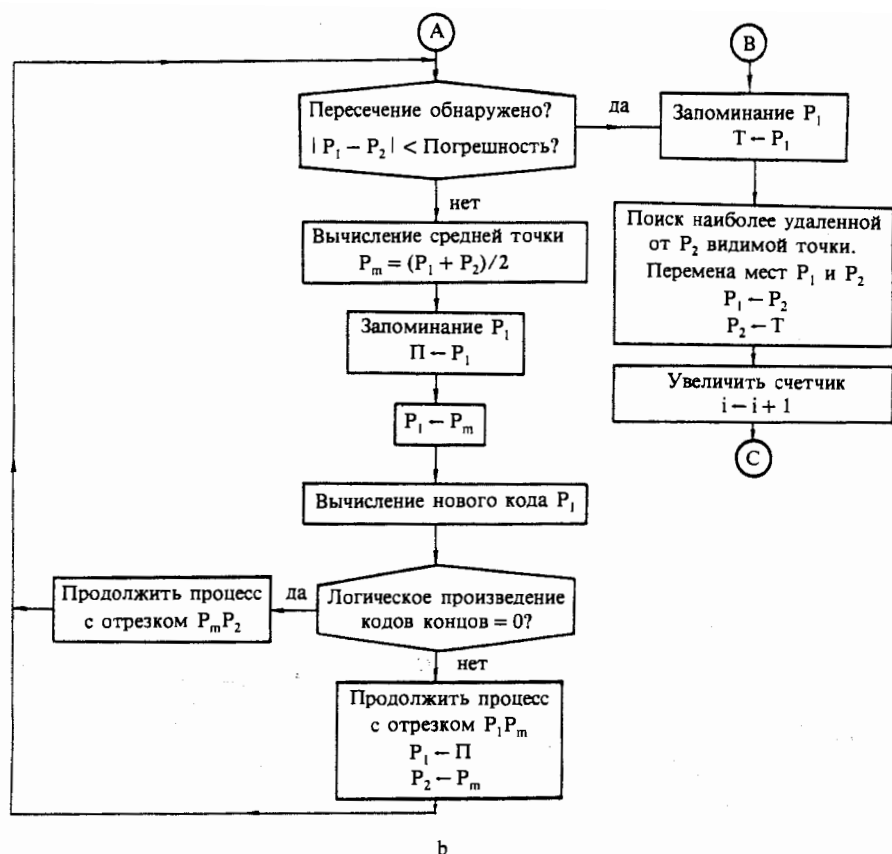


Рис. 3.6. Продолжение.

Двумерный алгоритм отсекаания методом разбиения отрезка средней точкой

Окно — массив 1×4 , в котором содержатся координаты ($x_{Л}$, $x_{П}$, $y_{Н}$, $y_{В}$) левой, правой, нижней и верхней сторон прямоугольного отсекающего окна

P_1 , P_2 — концевые точки отрезка

инициализация i

$i = 1$

вычисление кодов концевых точек

занесение кодов каждого конца в массивы 1×4 , именуемые T1код и T2код

первая концевая точка: P_1

- 1
 - call** Конец(P_1 , Окно; T1код, Сумма1)
 - вторая концевая точка: P_2*
 - call** Конец(P_2 , Окно; T2код, Сумма2)
 - проверка полной видимости отрезка*
 - if** Сумма1 = 0 и Сумма2 = 0 **then** 5
 - отрезок не является полностью видимым*
 - проверка тривиальной невидимости отрезка*
 - call** Логическое(T1код, T2код; Произвед)
 - if** Произвед < > 0 **then** 6
 - отрезок может оказаться частично видимым*
 - поиск наиболее удаленной от P_1 видимой точки отрезка*
 - запоминание исходной точки P_1*
 - $P_{аб} = P_1$
 - проверка окончания процесса решения*
 - if** $i > 2$ **then** 4
 - P_2 — это наиболее удаленная от P_1 видимая точка отрезка?
 - if** Сумма2 = 0 **then** 3
 - пересечение уже найдено?*
- 2
 - if** $|P_1 - P_2| < \text{Погрешность}$ **then** 3
 - вычисление средней точки*
 - $P_m = (P_1 + P_2)/2$
 - запоминание текущей точки P_1*
 - Память = P_1
 - замена P_1 на P_m*
 - $P_1 = P_m$
 - вычисление нового кода точки P_1*
 - call** Конец (P_1 , Окно; T1код, Сумма1)
 - проверка тривиальной невидимости отрезка $P_m P_2$*
 - call** Логическое(T1код, T2код; Произвед)
 - if** Произвед = 0 **then** 2
 - $P_m P_2$ невидим, продолжить процедуру с $P_1 P_m$
 - $P_1 = \text{Память}$
 - $P_2 = P_m$
 - go to** 2
 - обнаружена наиболее удаленная от P_1 видимая точка отрезка*
 - поиск наиболее удаленной от P_2 видимой точки отрезка*
 - перемена мест P_1 и P_2*
- 3
 - $P_1 = P_2$
 - $P_2 = P_{аб}$

увеличить счетчик

$i = i + 1$

начать процесс с «новым» укороченным отрезком

go to 1

теперь обнаружены оба пересечения

проверка вырожденной невидимой точки

4 **call** Логическое(Т1код, Т2код; Произвед)

If Произвед < > 0 **then** 6

5 Начертить отрезок

6 **finish**

подпрограмма вычисления кодов концевой точки отрезка

subroutine Конец(Р, Окно; Ткод, Сумма)

P_x, P_y — координаты x и y точки Р

Окно — массив 1×4 , содержащий координаты (x_L, x_P, y_H, y_B) левой, правой, нижней и верхней сторон прямоугольного отсекающего окна

Ткод — массив 1×4 , содержащий коды концевой точки

Сумма — поэлементная сумма Ткод

определение кодов концевой точки

If $P_x < x_L$ **then** Ткод(4) = 1 **else** Ткод(4) = 0

If $P_x > x_P$ **then** Ткод(3) = 1 **else** Ткод(3) = 0

If $P_y < y_H$ **then** Ткод(2) = 1 **else** Ткод(2) = 0

If $P_y > y_B$ **then** Ткод(1) = 1 **else** Ткод(1) = 0

вычисление суммы кодов

Сумма = 0

for $i = 1$ **to** 4

Сумма = Сумма + Ткод(i)

next i

return

подпрограмма вычисления логического произведения

subroutine Логическое(Т1код, Т2код; Произвед)

Т1код, Т2код — массивы 1×4 , содержащие коды концевых точек

Произвед — сумма битов в логическом произведении кодов концов

Произвед = 0

for $i = 1$ **to** 4

Произвед = Произвед + Целая часть((Т1код(i) +
+ Т2код(i))/2)

next i

return

В ранее описанном простом алгоритме отсечения коды конечных точек отрезка и их логическое произведение определялись прямо в теле алгоритма. В данном же алгоритме для этого используются подпрограммы, поскольку коды концов и их логическое произведение вычисляются неоднократно.

3.4. ОБОБЩЕНИЕ: ОТСЕЧЕНИЕ ДВУМЕРНОГО ОТРЕЗКА ВЫПУКЛЫМ ОКНОМ

В описанных выше алгоритмах предполагалось, что отсекающее окно является координатно ориентированным прямоугольником. Однако во многих случаях окно не таково. Например, предположим, что прямоугольное окно повернуто относительно системы координат так, как показано на рис. 3.7. В этом случае неприменим ни один из ранее описанных алгоритмов. Кирус и Бек предложили [3-4] алгоритм отсечения окном произвольной выпуклой формы.

Прежде чем подробно описывать алгоритм Кируса — Бека (это делается в следующем разделе), рассмотрим отсечение параметрически заданного отрезка прямоугольным окном. Параметрическое уравнение отрезка от P_1 до P_2 имеет вид

$$P(t) = P_1 + (P_2 - P_1)t \quad 0 \leq t \leq 1 \quad (3.1)$$

где t — параметр. Если ограничиться значениями $0 \leq t \leq 1$, то получается именно отрезок, а не бесконечная прямая. Параметрическое описание отрезка не зависит от выбора системы координат. Это свойство делает параметрическое представление особенно полезным для определения пересечения отрезка со стороной произ-

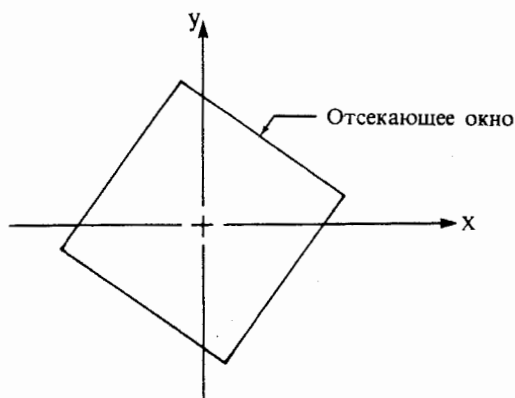


Рис. 3.7. Окно, повернутое относительно осей.

вольного выпуклого многоугольника. Проиллюстрируем этот метод сначала на примере регулярного прямоугольного окна.

В двумерной декартовой системе координат уравнение (3.1) сводится к паре одномерных параметрических уравнений вида:

$$x(t) = x_1 + (x_2 - x_1)t \quad 0 \leq t \leq 1 \quad (3.2a)$$

$$y(t) = y_1 + (y_2 - y_1)t \quad 0 \leq t \leq 1 \quad (3.2b)$$

В случае прямоугольного отсекающего окна одна из координат пересечения отрезка с каждой стороной известна. Нужно вычислить только вторую координату. Из уравнения (3.1) получаем: $t = (P(t) - P_1) / (P_2 - P_1)$. А из (3.2) следует, что значения t , соответствующие пересечениям со сторонами окна, равны:

$$\text{для левой стороны: } t = \frac{x_{\text{л}} - x_1}{x_2 - x_1} \quad 0 \leq t \leq 1$$

$$\text{для правой стороны: } t = \frac{x_{\text{п}} - x_1}{x_2 - x_1} \quad 0 \leq t \leq 1$$

$$\text{для нижней стороны: } t = \frac{y_{\text{н}} - y_1}{y_2 - y_1} \quad 0 \leq t \leq 1$$

$$\text{для верхней стороны: } t = \frac{y_{\text{в}} - y_1}{y_2 - y_1} \quad 0 \leq t \leq 1$$

Здесь через $x_{\text{л}}$, $x_{\text{п}}$, $y_{\text{н}}$, $y_{\text{в}}$ обозначены координаты левой, правой, нижней и верхней сторон окна соответственно. Если решения этих уравнений дают значения t за пределами интервала $0 \leq t \leq 1$, то такие решения отвергаются, поскольку они соответствуют точкам, лежащим вне исходного отрезка.

Пример 3.4. Простой частично видимый отрезок

Рассмотрим частично видимый отрезок от $P_1(-3/2, -3/4)$ до $P_2(3/2, 1/2)$, отсекаемый окном с координатами сторон $x_{\text{л}}$, $x_{\text{п}}$, $y_{\text{н}}$, $y_{\text{в}}$, равными $(-1, 1, -1, 1)$, как показано на рис. 3.8. Имеем значения t для точек пересечения отрезка

$$\text{с левой стороной: } t = \frac{x_{\text{л}} - x_1}{x_2 - x_1} = \frac{-1 - (-3/2)}{3/2 - (-3/2)} = \frac{1/2}{3} = \frac{1}{6}$$

$$\text{с правой стороной: } t = \frac{x_{\text{п}} - x_1}{x_2 - x_1} = \frac{1 - (-3/2)}{3/2 - (-3/2)} = \frac{5/2}{3} = \frac{5}{6}$$

$$\text{с нижней стороной: } t = \frac{y_{\text{н}} - y_1}{y_2 - y_1} = \frac{-1 - (-3/4)}{1/2 - (-3/4)} = \frac{-1/4}{5/4} = \frac{-1}{5}$$

(это число меньше нуля и поэтому отвергается)
с верхней стороной:

$$t = \frac{y_B - y_1}{y_2 - y_1} = \frac{1 - (-3/4)}{1/2 - (-3/4)} = \frac{7/4}{5/4} = \frac{7}{5}$$

(это число больше единицы и также отвергается).

Видимый участок отрезка заключен в интервале $1/6 \leq t \leq 5/6$.

Значения x и y координат точек пересечения получаются из параметрических уравнений. В частности, при $t = 1/6$ из уравнения (3.2) имеем:

$$x(1/6) = -3/2 + [3/2 - (-3/2)](1/6) = -1$$

что, разумеется, априори известно, поскольку $x = -1$ — это абсцисса левой стороны окна. Для координаты y имеем:

$$y(1/6) = -3/4 + [1/2 - (-3/4)](1/6) = -13/24$$

Аналогично при $t = 5/6$ имеем:

$$\begin{aligned} [x(5/6) \ y(5/6)] &= [-3/2 \ -3/4] + [3/2 - (-3/2) \ 1/2 - (-3/4)] (5/6) \\ &= [1 \ 7/24] \end{aligned}$$

Здесь вычисления значений x и y объединены. Опять, поскольку отрезок пересекает правую сторону окна, то координата x априори известна.

После изложенного примера может показаться, что предлагаемый метод прост и естествен. Однако существует ряд трудностей, которые лучше продемонстрировать на следующих примерах.

Пример 3.5. Частично видимый отрезок

Рассмотрим отрезок от $P_3(-5/2, -1)$ до $P_4(3/2, 2)$, который тоже показан на рис. 3.8, а отсекающее окно опять имеет координаты сторон $(-1, 1, -1, 1)$. Точки пересечения задаются следующими значениями параметра:

$$t_L = 3/8, \ t_P = 7/8, \ t_H = 0, \ t_B = 2/3,$$

и все эти четыре значения принадлежат интервалу $0 \leq t \leq 1$.

Хорошо известно, что если отрезок прямой пересекает выпуклый многоугольник, то возникает не более двух точек пересечения. Следовательно, нужны только два из четырех значений параметра, вычисленных в примере 3.5. Упорядочение этих четырех значений по возрастанию t дает последовательность t_H, t_L, t_B, t_P . На рис. 3.8 показано, что искомыми являются значения $t_L = 3/8$ и $t_B = 2/3$, соответствующие точкам пересечения $(-1, 1/8)$ и $(1/6, 1)$. Эти значения совпадают с $t_{\max\min}$ и $t_{\min\max}$. Формальное вычисление этих значений сводится к простой классической задаче линейного

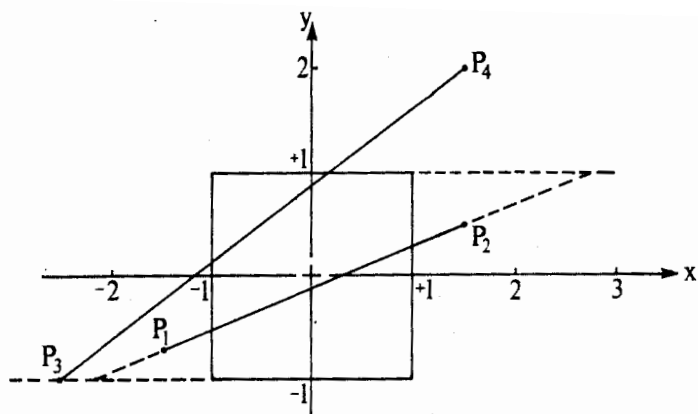


Рис. 3.8. Параметрическое отсечение частично видимых отрезков.

программирования. Один алгоритм решения этой задачи дан в следующем разделе.

Как и в любом алгоритме отсечения, здесь важно уметь быстро выявлять и отделять полностью видимые и полностью невидимые отрезки. Следующие два примера проиллюстрируют ряд новых трудностей.

Пример 3.6. Полностью видимый отрезок

Рассмотрим полностью видимый отрезок от $P_1(-1/2, 1/2)$ до $P_2(1/2, -1/2)$, опять отсекаемый окном со сторонами $(-1, 1, -1, 1)$, показанным на рис. 3.9. Значения параметров, соответствующие пересечениям отрезка со сторонами окна, равны:

$$t_D = -1/2, \quad t_P = 3/2, \quad t_H = 3/2, \quad t_B = -1/2.$$

Все эти значения находятся вне интервала $0 \leq t \leq 1$.

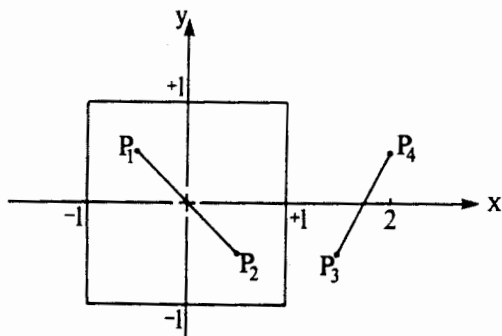


Рис. 3.9. Отсечение полностью видимых и невидимых отрезков.

Из анализа примера 3.6 может показаться, что найден метод определения полностью видимых отрезков. Однако следующий пример показывает, что это неверно.

Пример 3.7. Полностью невидимый отрезок

Рассмотрим полностью невидимый отрезок от $P_3(3/2, -1/2)$ до $P_4(2, 1/2)$, который тоже показан на рис. 3.9. Отсекающее окно опять задано сторонами $(-1, 1, -1, 1)$. Здесь значения параметра, соответствующие пересечениям отрезка со сторонами окна, равны:

$$t_L = -5, \quad t_P = -1, \quad t_H = -1/2, \quad t_B = 3/2.$$

И опять все эти значения находятся вне интервала 0.

Из примера 3.7 следует, что значения параметра удовлетворяют ранее определенным условиям полной видимости отрезка. Однако в отличие от отрезка $P_1 P_2$ из примера 3.6 отрезок $P_3 P_4$ невидим. Из последующих двух примеров следует, что для параметрически заданных отрезков нет простого и единственного метода, различающего полностью видимые и полностью невидимые отрезки. Кроме того, ясно, что эта задача требует более формального подхода.

3.5. АЛГОРИТМ КИРУСА — БЕКА

Для создания надежного алгоритма отсекающего окна нужно иметь хороший метод определения местоположения относительно окна (внутри, на границе или вне его) точки, принадлежащей отрезку. Для этой цели в алгоритме Кируса — Бека [3-4] используется вектор нормали.

Возьмем выпуклую отсекающую область R . Хотя R не обязана быть двумерной, в примерах, приведенных в данном разделе, предполагается, что она двумерна. Итак, R может быть любым плоским выпуклым многоугольником. Она *не должна быть* вогнутым многоугольником. Внутренняя нормаль \mathbf{n} в произвольной точке \mathbf{a} , лежащей на границе R , удовлетворяет условию: $\mathbf{n} \cdot (\mathbf{b} - \mathbf{a}) \geq 0$, где \mathbf{b} — любая другая точка на границе R . Чтобы убедиться в этом, вспомним, что скалярное произведение двух векторов \mathbf{V}_1 и \mathbf{V}_2 равно: $\mathbf{V}_1 \mathbf{V}_2 \cdot |\mathbf{V}_1| |\mathbf{V}_2| \cos \theta$, где θ — это меньший из двух углов, образованных \mathbf{V}_1 и \mathbf{V}_2 . Заметим, что если $\theta = \pi/2$, то $\cos \theta = 0$ и $\mathbf{V}_1 \cdot \mathbf{V}_2 = 0$, т. е. когда скалярное произведение пары векторов равно нулю, то они перпендикулярны. На рис. 3.10 показана выпуклая область R т. е. отсекающее окно. Там же показаны внешняя (наружная) \mathbf{n}_n и внутренняя $\mathbf{n}_в$ нормали к границе окна, исходящие из

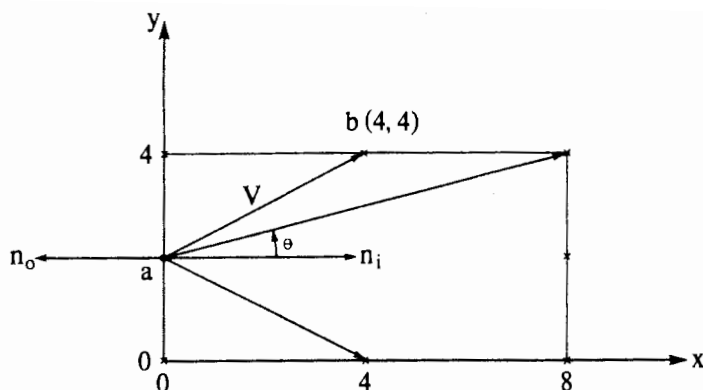


Рис. 3.10. Внутренняя и внешняя (наружная) нормали.

точки a , лежащей на этой границе. Кроме того, на рис. 3.10 показаны еще несколько векторов, проведенных из точки a в другие точки на границе окна. Угол между n_b и любым из таких векторов всегда принадлежит интервалу $-\pi/2 \leq \theta \leq \pi/2$. При таких значениях угла косинус его всегда положителен. Следовательно, положительно и скалярное произведение этих векторов, как было установлено выше. А вот угол между внешней нормалью и любым из подобных векторов равен $\pi - \theta$, а $\cos(\pi - \theta) = -\cos \theta$ при этом отрицателен. Для лучшего понимания этого факта рассмотрим следующий пример.

Пример 3.8. Внешняя и внутренняя нормали

Возьмем прямоугольную область, приведенную на рис. 3.10. Здесь внутренняя и внешняя нормали в точке a равны $n_b = I$ и $n_n = -I$ соответственно, где I — единичный вектор вдоль оси x . В табл. 3.3 показаны значения скалярных произведений

Таблица 3.3.

a	b	$n_b \cdot (b - a)$	$n_n \cdot (b - a)$
(0, 2)	(0, 4)	0	0
	(4, 4)	4	-4
	(8, 4)	8	-8
	(8, 2)	8	-8
	(8, 0)	8	-8
	(4, 0)	4	-4
	(0, 0)	0	0

внешней и внутренней нормалей на векторы, проведенные из точки a в разные точки b , лежащие на границе области. Особо выделим случай точки $b(4,4)$, тогда $b - a = 4i + 2j$, а $n_b \cdot (b - a) = 1 \cdot (4i + 2j) = 4$. Нули в последнем столбце табл. 3.3 означают, что соответствующие векторы перпендикулярны нормальям.

Возвращаясь к определению пересечения отрезка со стороной окна, вновь возьмем параметрическое представление отрезка $P_1 P_2$:

$$P(t) = P_1 + (P_2 - P_1)t, \quad 0 \leq t \leq 1.$$

Если f — граничная точка выпуклой области R , а n — внутренняя нормаль к одной из ограничивающих эту область плоскостей, то для любой конкретной величины t , т. е. для любой точки отрезка $P_1 P_2$, из условия $n \cdot [P(t) - f] < 0$ следует, что вектор $P(t) - f$ направлен вовне области R . Из условия $n \cdot [P(t) - f] = 0$ следует, что $P(t) - f$ принадлежит плоскости, которая проходит через f и перпендикулярна нормали. Из условия $n \cdot [P(t) - f] > 0$ следует, что вектор $P(t) - f$ направлен внутрь R , как показано на рис. 3.11. Из всех этих условий взятых вместе следует, что бесконечная прямая пересекает замкнутую выпуклую область, которая в двумерном случае сводится к замкнутому выпуклому многоугольнику ровно в двух точках. Далее, пусть эти две точки не принадлежат одной граничной плоскости или ребру. Тогда уравнение $n \cdot [P(t) - f] = 0$ имеет только одно решение. Если точка f лежит на той граничной плоскости или на том ребре, для которых n является внутренней нормалью, то точка на отрезке $P(t)$, которая удовлетворяет последнему уравнению, будет точкой пересечения этого отрезка с указанной граничной плоскостью.

Пример 3.9. Алгоритм Кируса — Бека. Частично видимый отрезок. Рассмотрим отрезок от $P_1(-1, 1)$ до $P_2(9, 3)$, отсекаемый прямоугольным окном, показанным на рис. 3.12. Уравнение прямой, несущей $P_1 P_2$, имеет вид

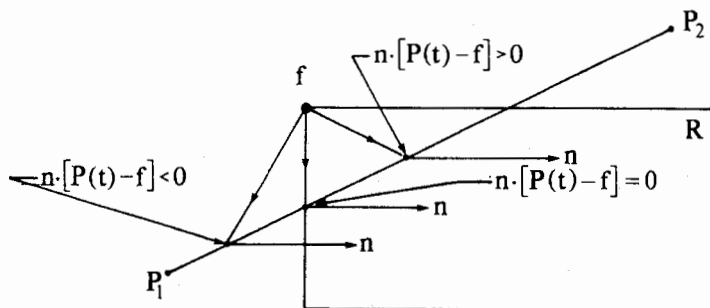


Рис. 3.11. Направления векторов.



Рис. 3.12. Отсечение Кируса — Бека: частично видимый отрезок.

$y = 0.2(x - 6)$, и она пересекает окно в точках $(0, 1.2)$ и $(8, 2.8)$. Параметрическое описание отрезка P_1P_2 таково:

$$\begin{aligned} P(t) &= P_1 + (P_2 - P_1)t = [-1 \ 1] + [10 \ 2]t \\ &= (10t - 1)i + (2t + 1)j \quad 0 \leq t \leq 1 \end{aligned}$$

здесь i и j — единичные векторы, ориентированные вдоль осей x и y соответственно. Четыре внутренние нормали к сторонам окна равны: левая $n_L = i$, правая $n_P = -i$, нижняя $n_H = j$, верхняя $n_B = -j$.

Выбрав точку $f(0,0)$ на левой стороне окна, имеем $P(t) - f = (10t - 1)i + (2t + 1)j$, а уравнение $n_L \cdot [P(t) - f] = 10t - 1 = 0$ дает значение $t = 1/10$ для пересечения отрезка с левой стороной окна. Подстановка $P(1/10) = [-1 \ 1] + [10 \ 2](1/10) = [0 \ 1.2]$ дает то же самое значение, которое было вычислено выше другим методом.

Выбрав точку $f(8, 4)$ на правой стороне окна, имеем $P(t) - f = (10t - 9)i + (2t - 3)j$, а уравнение $n_P \cdot [P(t) - f] = -(10t - 9) = 0$ дает значение $t = 9/10$ для пересечения отрезка с выбранной стороной. Подстановка дает $P(9/10) = [-1 \ 1] + [10 \ 2](9/10) = [8 \ 2.8]$, что также совпадает с результатом вычисления другим методом.

Использование точки $f(0, 0)$ на нижней стороне окна приводит к уравнению $n_H \cdot [P(t) - f] = 2t + 1 = 0$, решением которого является значение $t = -1/2$. Оно лежит вне интервала $0 \leq t \leq 1$ и поэтому отвергается.

Использование точки $f(8, 4)$ на верхней стороне окна дает уравнение $n_B \cdot [P(t) - f] = -(2t - 3) = 0$ с решением $t = 3/2$. Последнее значение t тоже лежит вне интервала $0 \leq t \leq 1$ и также отвергается. Видимый участок отрезка P_1P_2 , отсекаемый прямоугольной областью, показанной на рис. 3.12, лежит в интервале $1/10 \leq t \leq 9/10$ или от точки $[0 \ 1.2]$ до точки $[8 \ 2.8]$.

Из последнего примера следует, что точки пересечения можно отыскивать без большого труда. Способы идентификации полностью видимого и полностью невидимого отрезков продемонстрированы в следующих трех примерах.

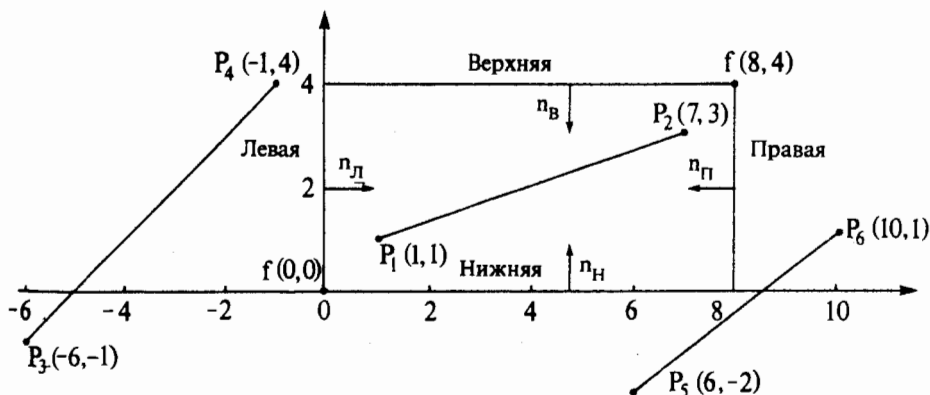


Рис. 3.13. Отсечение Кируса — Бека: полностью видимый и невидимый отрезки.

Пример 3.10. Алгоритм Кируса — Бека. Полностью видимый отрезок

Рассмотрим отрезок от $P_1(1, 1)$ до $P_2(7, 3)$, отсекаемый прямоугольным окном, показанным на рис. 3.13. Его параметрическое описание имеет вид $P(t) = [1 \ 1] + [6 \ 2] t$.

Результаты вычислений, использующих те же внутренние нормали и граничные точки, что и в примере 3.9, приведены в табл. 3.4. Все значения t , которые соответствуют точкам пересечения лежат вне интервала $0 \leq t \leq 1$. Значит, отрезок полностью видимый.

Таблица 3.4.

Ребро	n	f	$P(t) - f$	$n \cdot [P(t) - f]$	t
Левая	i	$(0, 0)$	$(1 + 6t)i + (1 + 2t)j$	$1 + 6t$	$-1/6$
Правая	$-i$	$(8, 4)$	$(-7 + 6t)i + (-3 + 2t)j$	$7 - 6t$	$7/6$
Нижняя	j	$(0, 0)$	$(1 + 6t)i + (1 + 2t)j$	$1 + 2t$	$-1/2$
Верхняя	$-j$	$(8, 4)$	$(-7 + 6t)i + (-3 + 2t)j$	$3 - 2t$	$3/2$

В следующих двух примерах рассматриваются два типа невидимых отрезков. Один из этих отрезков целиком расположен слева от окна, и его невидимость может быть установлена анализом кодов концевых точек, который был описан ранее. Второй отрезок пересекает прямую, несущую диагональ окна за его пределами. Его нельзя объявить невидимым, используя только коды концевых точек.

Пример 3.11. Алгоритм Кируса — Бека. Тривиально невидимый отрезок

Рассмотрим отрезок от $P_3(-6, -1)$ до $P_4(-1, 4)$, отсекаемый прямоугольным окном, показанным на рис. 3.13. Этот отрезок невидим. Его параметрическое представление имеет вид: $P(t) = [-6 - 1] + [5 \ 5] t$.

Результаты вычислений, использующих те же внутренние нормали и граничные точки, которые использовались в предыдущем примере, приведены в табл. 3.5.

Анализ результатов из табл. 3.5 показывает, что оба значения параметров для точек пересечения с левой и правой сторонами лежат вне интервала $0 \leq t \leq 1$, а оба значения для пересечения с нижней и верхней сторонами лежат внутри интервала $0 \leq t \leq 1$. Поэтому сначала можно было бы предположить, что отрезок видим в интервале $1/5 \leq t \leq 1$. Однако дальнейший анализ пересечений с левой и правой сторонами показывает, что оба значения параметра больше единицы. Отсюда следует, что окно полностью расположено справа от отрезка. Значит, отрезок невидим.

Таблица 3.5.

Ребро	n	f	$P(t)-f$	$n \cdot [P(t)-f]$	t
Левая	i	(0, 0)	$(-6 + 5t)i + (-1 + 5t)j$	$-6 + 5t$	$t = 6/5$
Правая	-i	(8, 4)	$(-14 + 5t)i + (-5 + 5t)j$	$-(-14 + 5t)$	$t = 14/5$
Нижняя	j	(0, 0)	$(-6 + 5t)i + (-1 + 5t)j$	$-1 + 5t$	$t = 1/5$
Верхняя	-j	(8, 4)	$(-14 + 5t)i + (-5 + 5t)j$	$-(-5 + 5t)$	$t \neq 1$

Если в последнем примере поменять P_3 и P_4 местами, то окно будет полностью расположено слева от отрезка. Ориентация отрезка играет важную роль при принятии решения о его невидимости. В следующем примере этот вопрос исследуется полнее.

Пример 3.12. Алгоритм Кируса — Бека. Нетривиально невидимый отрезок.

Здесь рассматривается отрезок от $P_5(6, -2)$ до $P_6(10, 1)$, который снова отсекается прямоугольным окном, показанным на рис. 3.13. Параметрическое описание отрезка имеет вид $P(t) = [6 - 2] + [4 \ 3] t$.

Использование внутренних нормалей и граничных точек, взятых из предыдущих примеров, приводит к результатам, приведенным в табл. 3.6.

Таблица 3.6.

Ребро	n	f	$P(t)-f$	$n \cdot [P(t)-f]$	t
Левая	i	(0, 0)	$(6 + 4t)i + (-2 + 3t)j$	$6 + 4t$	$t = -3/2$
Правая	-i	(8, 4)	$(-2 + 4t)i + (-6 + 3t)j$	$-(-2 + 4t)$	$t = 1/2$
Нижняя	j	(0, 0)	$(6 + 4t)i + (-2 + 3t)j$	$-2 + 3t$	$t = 2/3$
Верхняя	-j	(8, 4)	$(-2 + 4t)i + (-6 + 3t)j$	$-(-6 + 3t)$	$t = 2$

Эти результаты свидетельствуют о том, что значения параметра для пересечения с левой и верхней сторонами окна выходят за допустимые пределы. Однако значения параметра для пересечений с правой и нижней сторонами окна допустимы. Но учет ориентации отрезка от P_5 к P_6 показывает, что такой отрезок не может пересечь правую сторону окна при $t = 1/2$ прежде, чем пересечет его нижнюю сторону при $t = 2/3$, и при этом иметь с окном общие точки. Поэтому отрезок невидим.

Из приведенных примеров следует, что учет ориентации помогает корректно идентифицировать полностью видимые отрезки. Это соображение используется при формальной записи алгоритма Кируса — Бека, которая приводится ниже.

Для формализации этого алгоритма напомним, что параметрическое описание отрезка имеет вид:

$$P(t) = P_1 + (P_2 - P_1)t \quad 0 \leq t \leq 1 \quad (3.3)$$

а скалярное произведение внутренней нормали на вектор, начинающийся в произвольной точке отрезка и заканчивающийся в другой точке, лежащей на той же границе окна, т. е.

$$n_i \cdot [P(t) - f_i] \quad i = 1, 2, 3, \dots \quad (3.4)$$

будет положительно, равно нулю или отрицательно — в зависимости от того, будет ли избранная точка отрезка лежать с внутренней стороны границы, на самой этой границе или с ее внешней стороны. Последнее соотношение применимо к любой плоскости или ребру номер i , ограничивающим область. Подстановка (3.3) в (3.4) дает условие пересечения отрезка с границей области:

$$n_i \cdot [P_1 + (P_2 - P_1)t - f_i] = 0 \quad (3.5)$$

В другой форме уравнение (3.5) имеет вид:

$$n_i \cdot [P_1 - f_i] + n_i \cdot [P_2 - P_1]t = 0 \quad (3.6)$$

Заметим, что вектор $P_2 - P_1$ определяет ориентацию отрезка, а вектор $P_1 - f_i$ пропорционален расстоянию от первого конца отрезка до избранной граничной точки. Обозначим через $D = P_2 - P_1$ директрису или ориентацию отрезка, а через $w_i = P_1 - f_i$ — некий весовой множитель. Тогда уравнение (3.6) можно записать так:

$$t(n_i \cdot D) + w_i \cdot n_i = 0 \quad (3.7)$$

Решая последнее уравнение относительно t , имеем:

$$t = -\frac{w_i \cdot n_i}{D \cdot n_i} \quad D \neq 0 \quad i = 1, 2, 3, \dots \quad (3.8)$$

Здесь $\mathbf{D} \cdot \mathbf{n}_i$ может быть равным нулю только при $\mathbf{D} = 0^{1)}$, а это означает, что $P_2 = P_1$, т. е. вырождение отрезка в точку. Если

$$\mathbf{w}_i \cdot \mathbf{n}_i \begin{cases} < 0, \text{ то эта точка вне окна} \\ = 0, \text{ то она на границе окна} \\ > 0, \text{ то она внутри окна} \end{cases}$$

Уравнение (3.8) используется для получения значений t , соответствующих пересечениям отрезка с каждой стороной окна. Если значение t лежит за пределами интервала $0 \leq t \leq 1$, то можно его проигнорировать. Хотя известно, что отрезок может пересечь выпуклое окно не более чем в двух точках, т. е. при двух значениях t , уравнения (3.8) могут дать большее число решений в интервале $0 \leq t \leq 1$. Эти решения следует разбить на две группы: нижнюю и верхнюю, в зависимости от того, к началу или к концу отрезка будет ближе соответствующая точка. Нужно найти наибольшую из нижних и наименьшую из верхних точек. Если $\mathbf{D} \cdot \mathbf{n}_i > 0$, то найденное значение t рассматривается в качестве возможного нижнего предела. Если $\mathbf{D} \cdot \mathbf{n}_i < 0$, то значение t рассматривается в качестве возможного верхнего предела. На рис. 3.14 дана блок-схема алгоритма, в котором данные условия используются для решения получающейся задачи линейного программирования. Ниже приводится запись этого алгоритма на псевдокоде.

Алгоритм двумерного отсекаания Кируса — Бека

P_1, P_2 — концевые точки отрезка
 k — число сторон отсекающего окна
 \mathbf{n}_i — вектор нормали к i -й стороне окна
 \mathbf{f}_i — точка, лежащая на i -й стороне окна
 \mathbf{D} — директриса отрезка, равная $P_2 - P_1$
 \mathbf{w}_i — весовая функция, равная $P_1 - \mathbf{f}_i$
 t_H, t_B — нижний и верхний пределы значений параметра t
 инициализировать пределы значений параметра, предполагая что отрезок полностью видимый
 $t_H = 0$
 $t_B = 1$
 вычислить директрису \mathbf{D}
 $\mathbf{D} = P_2 - P_1$
 начать главный цикл
for $i = 1$ **to** k

¹⁾ А также при параллельности \mathbf{D} избранной границе. — Прим. перев.

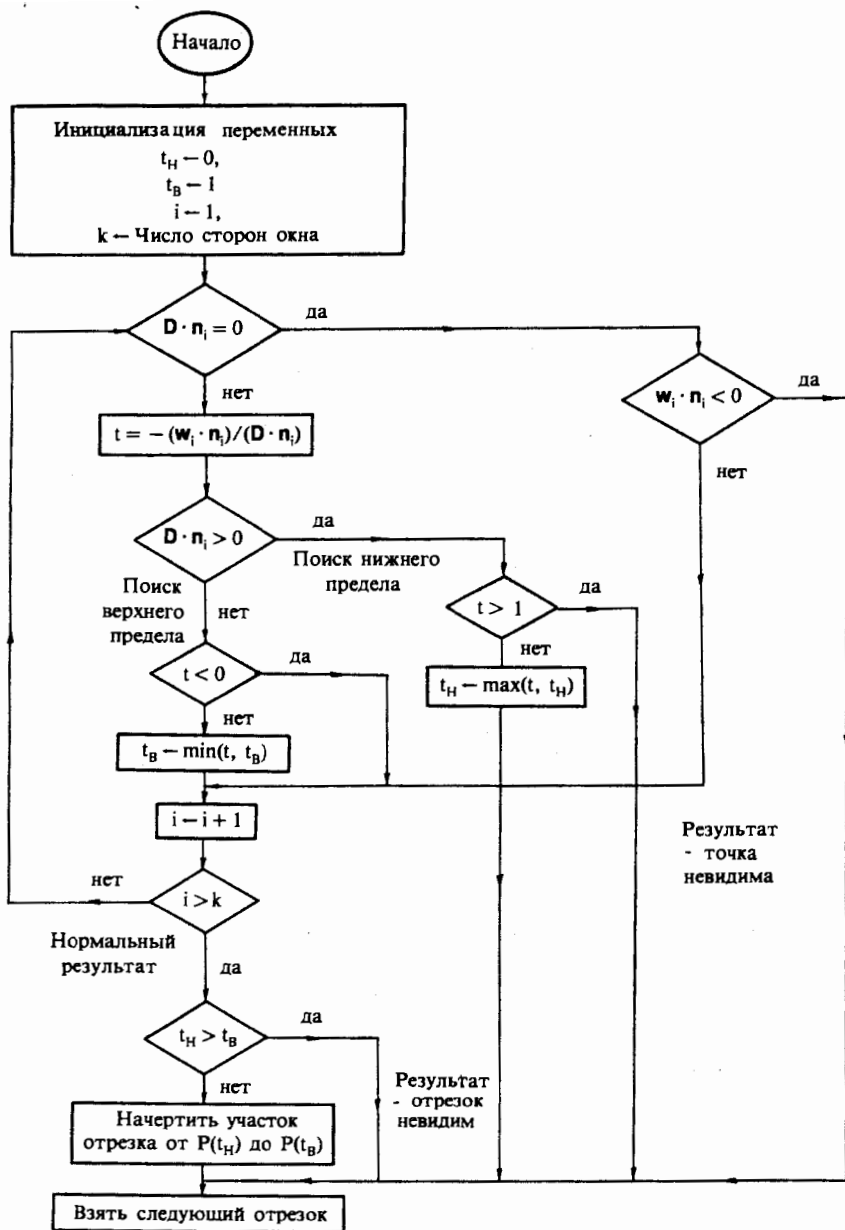


Рис. 3.14. Блок-схема алгоритма отсечения Кируса — Бека.

вычислить w_i , $D \cdot n_i$ и $w_i \cdot n$ для данного i
 $w_i = P_i - f_i$
call Скал-произвед($D, n_i; D_{Ck}$)
call Скал-произвед($w_i, n_i; W_{Ck}$)
 отрезок вырождается в точку?
if $D_{Ck} = 0$ **then** 2
 отрезок невырожден, вычислить t
 $t = -W_{Ck}/D_{Ck}$
 поиск верхнего и нижнего пределов t
if $D_{Ck} > 0$ **then** 1
 поиск верхнего предела
 верно ли, что $0 \leq t \leq 1$?
if $t < 0$ **then** 4
 $t_B = \min(t, t_B)$
go to 3
 поиск нижнего предела
 верно ли, что $t \leq 1$?
 1 **if** $t > 1$ **then** 4
 $t_L = \max(t, t_L)$
go to 3
 отрезок выродился в точку
 2 **if** $W_{Ck} < 0$ **then** 4
 точка видима относительно текущей границы
 3 **next** i
 произошел нормальный выход из цикла
 проверка фактической видимости отрезка
if $t_H > t_B$ **then** 4
 Начертить отрезок от $P(t_H)$ до $P(t_B)$
 4 Обработать следующий отрезок
 подпрограмма вычисления скалярного произведения
subroutine Скал-произвед (Вектор1, Вектор2; Скал)
 Вектор1 — первый вектор, заданный компонентами x и y
 Вектор2 — второй вектор, заданный компонентами x и y
 Скал — скалярное произведение векторов
 $\text{Скал} = \text{Вектор1}_x * \text{Вектор2}_x + \text{Вектор1}_y * \text{Вектор2}_y$
return

Для иллюстрации того, что данный алгоритм не ограничивается окнами прямоугольной формы, рассмотрим следующий пример.

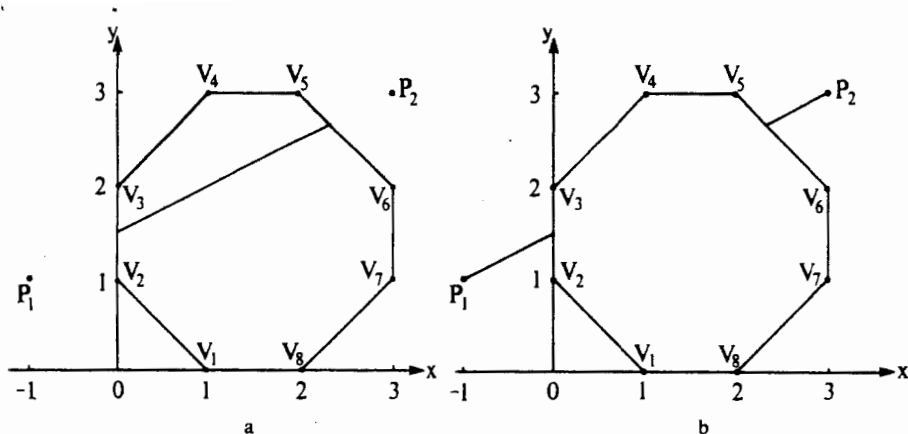


Рис. 3.15. Внутреннее (а) и внешнее (б) отсечение Кируса — Бека многоугольником.

Пример 3.13. Алгоритм Кируса — Бека. Нестандартное окно

На рис. 3.15, а показан восьмиугольник, служащий отсекающим окном. Отрезок от $P_1(-1, 1)$ до $P_2(3, 3)$ должен быть отсечен данным окном. В табл. 3.7 приведены результаты работы алгоритма Кируса — Бека. В качестве частного примера рассмотрим ребро от V_5 до V_6 . Алгоритм дает $D = P_2 - P_1 = [3 \ 3] - [-1 \ 1] = [4 \ 2]$. Для граничной точки $f(2, 3)$ имеем $w = P_1 - f = [-1 \ 1] - [2 \ 3] = [-3 \ -2]$. Внутренняя нормаль к ребру V_5V_6 равна $n = [-1 \ -1]$. Значит, $D \cdot n = -6 < 0$, и найден верхний предел. Поскольку $w \cdot n = 5$, то $t_B = -5/(-6) = 5/6$.

Анализ табл. 3.7 показывает, что максимальное среди значений t_H равно $1/4$, а минимальное среди t_B равно $5/6$. Как показано на рис. 3.15, а, отрезок видим в интервале $1/4 \leq t \leq 5/6$, или от точки $(0, 3/2)$ до точки $(7/3, 8/3)$.

Таблица 3.7.

Ребро	n	f	w	$w \cdot n$	$D \cdot n^{1)}$	t_H	t_B
V_1V_2	$[1 \ 1]$	$(1, 0)$	$[2 \ 1]$	-1	6	$1/5$	
V_2V_3	$[1 \ 0]$	$(0, 2)$	$[-1 \ -1]$	-1	4	$1/4$	
V_3V_4	$[1 \ -1]$	$(0, 2)$	$[-1 \ -1]$	0	2	0	
V_4V_5	$[0 \ -1]$	$(2, 3)$	$[-3 \ -2]$	2	-2		1
V_5V_6	$[-1 \ -1]$	$(2, 3)$	$[-3 \ -2]$	5	-6		$5/6$
V_6V_7	$[-1 \ 0]$	$(3, 1)$	$[-4 \ 0]$	4	-4		1
V_7V_8	$[-1 \ 1]$	$(3, 1)$	$[-4 \ 0]$	4	-2		1
V_8V_1	$[0 \ 1]$	$(1, 0)$	$[-2 \ 1]$	1	2	$-1/2$	

¹⁾ При $D \cdot n < 0$, верхний предел (t_B); при $D \cdot n > 0$, нижний (t_H).

3.6. ВНУТРЕННЕЕ И ВНЕШНЕЕ ОТСЕЧЕНИЕ

В предыдущих разделах упор был сделан на отсечение отрезка внутренней областью окна. Однако существует возможность отсечения отрезка и внешней его областью. Для этого надо определить часть или части отрезка, лежащие вне окна, и начертить их. Например, видимые части отрезка P_1P_2 , показанного на рис. 3.15, b, расположены в интервалах $0 \leq t < 1/6$ и $5/6 < t \leq 1$, или от точки $(-1, 1)$ до точки $(0, 3/2)$, а также от точки $(7/3, 8/3)$ до точки $(3, 3)$. Результаты как внутреннего, так и внешнего отсечений этого отрезка показаны на рис. 3.15.

Внешнее отсечение играет важную роль в дисплеях, допускающих работу с несколькими окнами, как это показано на рис. 3.16. На этом рисунке приоритет окон 1, 2, 3 выше приоритета окна всего экрана, а приоритет окон 1 и 3 выше приоритета окна 2. Поэтому изображение в окне экрана отсекается его собственной внутренней областью и внешними областями окон 1, 2, 3. Изображение в окне 2 отсекается его собственной областью и внешними областями окон 1 и 3. Изображения в окнах 1 и 3 нужно отсечь только соответствующими внутренними областями.

Внешним отсечением можно воспользоваться и при работе с вогнутым полигональным окном. На рис. 3.17 показан вогнутый многоугольник, заданный вершинами $V_1V_2V_3V_4V_5V_6V_1$. Этот вогнутый многоугольник можно преобразовать в выпуклый путем соединения вершин V_3 и V_5 , что и показано на рис. 3.17 штриховым отрезком. Отрезок P_1P_2 внутренне отсекается полученным много-

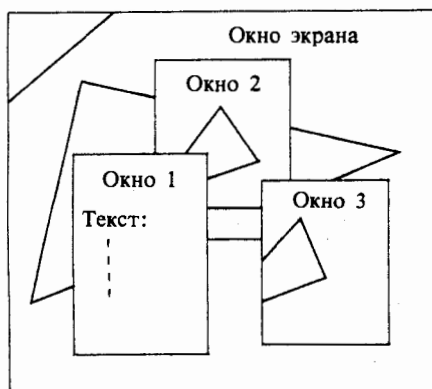


Рис. 3.16. Отсечение несколькими окнами.

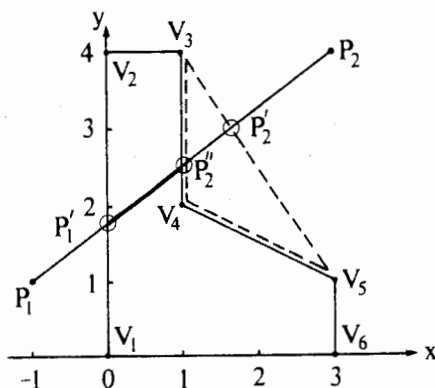


Рис. 3.17. Отсечение невыпуклым окном.

угольником с помощью алгоритма Кируса — Бека. А затем полученный при этом отрезок $P'_1P'_2$ внешне отсекается многоугольником $V_3V_5V_4V_3$, что и дает искомый результат — $P'_1P'_2$.

3.7. ОПРЕДЕЛЕНИЕ ФАКТА ВЫПУКЛОСТИ МНОГОУГОЛЬНИКА И ВЫЧИСЛЕНИЕ ЕГО ВНУТРЕННИХ НОРМАЛЕЙ

Для работы с алгоритмом Кируса — Бека надо прежде всего убедиться, что окно является выпуклым, а затем вычислить внутренние нормали к каждой его стороне. Факт выпуклости или невыпуклости двумерного полигонального окна можно установить путем вычисления векторных произведений его смежных сторон. Выводы, которые можно сделать из анализа знаков этих произведений, таковы:

- | | |
|---|---|
| Все знаки равны нулю | — многоугольник вырождается в отрезок. |
| Есть как положительные, так и отрицательные знаки | — многоугольник невыпуклый. |
| Все знаки неотрицательные | — многоугольник выпуклый, а внутренние нормали ориентированы влево от его контура. |
| Все знаки неположительны | — многоугольник выпуклый, а внутренние нормали ориентированы вправо от его контура. |

Рис. 3.18 иллюстрирует эти правила.

Другой подход заключается в том, что одна из вершин многоугольника может быть выбрана базой, и могут вычисляться векторные произведения для пар векторов, начинающихся в этой базе и заканчивающихся в последовательных вершинах многоугольника. Результаты этого метода интерпретируются точно так же.

Векторные произведения будут перпендикулярны к плоскости многоугольника. Векторные произведения двух плоских векторов V_1 и V_2 равно $(V_{x_1} V_{y_2} - V_{y_1} V_{x_2})\mathbf{k}$, где \mathbf{k} — единичный вектор, перпендикулярный к плоскости, несущей векторы-сомножители.

Нормаль к стороне многоугольника можно вычислить, если вспомнить, что скалярное произведение пары перпендикулярных

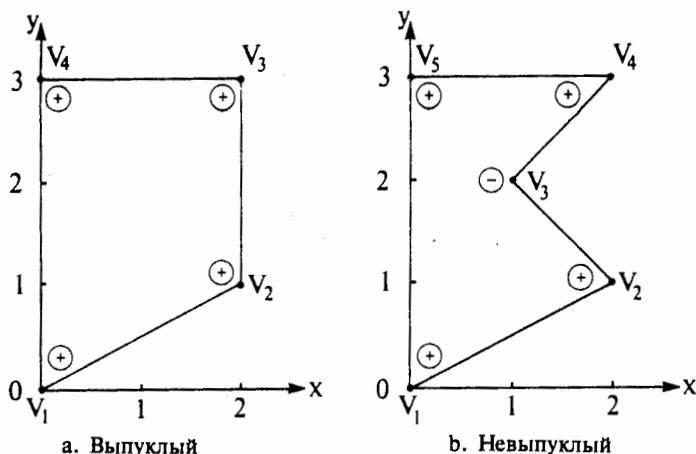


Рис. 3.18. Определение факта выпуклости многоугольника.

векторов равно нулю. Если n_x и n_y — неизвестные компоненты нормали к известному вектору (V_x, V_y) стороны многоугольника, то

$$\mathbf{n} \cdot \mathbf{V}_e = (n_x \mathbf{i} + n_y \mathbf{j}) \cdot (V_x \mathbf{i} + V_y \mathbf{j}) = n_x V_x + n_y V_y = 0$$

$$n_x V_x = -n_y V_y$$

Поскольку нас интересует только направление нормали, то положим, что $n_y = 1$ без потери общности. Следовательно, нормаль равна $\mathbf{n} = -V_y / V_x \mathbf{i} + \mathbf{j}$.

Если вектор стороны многоугольника образован как разность векторов пары смежных его вершин V_{i-1} и V_i и если скалярное произведение нормали и вектора от V_{i-1} до V_{i+1} положительно, то \mathbf{n} — внутренняя нормаль. В противном случае \mathbf{n} — внешняя нормаль. В последнем случае внутреннюю нормаль можно получить, умножив \mathbf{n} на -1 . Один простой пример проиллюстрирует этот метод.

Пример 3.14. Векторное произведение

На рис. 3.18, а показан простой выпуклый многоугольник, а на рис. 3.18, б — невыпуклый многоугольник. В табл. 3.8 и 3.9 приведены результаты всех вычислений. Рассмотрим, например, векторное произведение сторон, смежных вершине V_2 , и внутреннюю нормаль к стороне $V_1 V_2$ для многоугольника с рис. 3.18, а.

Стороны, смежные вершине V_2 , равны: $V_1 V_2 = 2\mathbf{i} + \mathbf{j}$, $V_2 V_3 = 2\mathbf{j}$. Векторное их произведение равно $V_1 V_2 \otimes V_2 V_3 = 4\mathbf{k}$, где \mathbf{k} — единичный вектор, перпендикулярный плоскости многоугольника. Это векторное произведение положительно. В табл. 3.8 показано, что векторные произведения положительны для всех вершин

Код этой точки равен (000000). Отрезок $P_c P_2$ — полностью видимый, отрезок $P_1 P_c$ — частично видимый. Алгоритм продолжает работу с отрезком $P_1 P_c$. Результаты процесса подразбиения даны в табл. 3.10. Фактические координаты точки пересечения равны $(-357.14, -357.14, 357.14)$. Отличие этих значений от приведенных в табл. 3.10 объясняется тем, что в алгоритме использовалась целочисленная арифметика.

3.11. ТРЕХМЕРНЫЙ АЛГОРИТМ КИРУСА — БЕКА

В двумерном варианте алгоритма Кируса — Бека [3-4] на форму отсекаателя не накладывалось никаких ограничений, за исключением выпуклости. Поэтому и в трехмерном варианте отсекаатель может быть произвольным выпуклым объемом. Можно непосредственно воспользоваться ранее разработанной двумерной версией. Теперь k обозначает не число сторон многоугольника, а число граней многогранника (см. рис. 3.14). Все векторы теперь имеют по три компоненты: x, y, z . Обобщение подпрограммы Скал — произвед на случай трехмерных векторов реализуется также непосредственно. Для более полной иллюстрации этого алгоритма рассмотрим следующие примеры. В первом примере отсекаателем служит прямоугольный параллелепипед, т. е. полый брусок.

Пример 3.17. Трехмерный алгоритм Кируса — Бека

На рис. 3.22 показан отрезок от $P_1(-2, -1, 1/2)$ до $P_2(3/2, 3/2, -1/2)$, который нужно отсечь по объему с координатами $(x_d, x_n, y_n, y_b, z_b, z_d) = (-1, 1, -1, 1, 1, -1)$. Шесть внутренних нормалей к граням отсекаателя, очевидно, равны:

верхняя : $n_b = -j = [0 \ -1 \ 0]$
 нижняя : $n_n = j = [0 \ 1 \ 0]$
 правая : $n_n = -i = [-1 \ 0 \ 0]$
 левая : $n_l = i = [1 \ 0 \ 0]$
 ближняя : $n_b = -k = [0 \ 0 \ -1]$
 дальняя : $n_d = k = [0 \ 0 \ 1]$

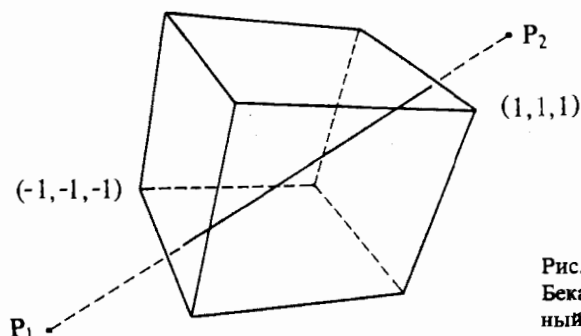


Рис. 3.22. Отсечение Кируса — Бека: трехмерный прямоугольный объем.

Таблица 3.11.

Грань	n	f	w	w · n	D · n ¹⁾	t _H	t _B
Верхняя	[0 -1 0]	(1, 1, 1)	[-3 -2 -1/2]	2	-5/2		4/5
Нижняя	[0 1 0]	(-1, -1, -1)	[-1 0 3/2]	0	5/2	0	
Правая	[-1 0 0]	(1, 1, 1)	[-3 -2 -1/2]	3	-7/2		6/7
Левая	[1 0 0]	(-1, -1, -1)	[-1 0 3/2]	-1	7/2	2/7	
Ближняя	[0 0 -1]	(1, 1, 1)	[-3 -2 -1/2]	1/2	1	-1/2	
Дальняя	[0 0 1]	(-1, -1, -1)	[-1 0 3/2]	3/2	-1		3/2

¹⁾ При $D \cdot n < 0$, верхний предел (t_B); при $D \cdot n > 0$, нижний предел (t_H).

Выбор точек, лежащих на каждой грани отсекаателя, также очевиден. Достаточно взять две точки, лежащие на концах одной из главных диагоналей параллелепипеда. Итак,

$$f_B = f_{\Pi} = f_B(1, 1, 1), \text{ а } f_H = f_{\Delta} = f_{\Delta}(-1, -1, -1)$$

Вместо этих точек можно взять центры или любые угловые точки на соответствующих гранях.

Директриса отрезка P_1P_2 равна:

$$D = P_2 - P_1 = [3/2 \ 3/2 \ -1/2] - [-2 \ -1 \ 1/2] = [7/2 \ 5/2 \ -1]$$

Для граничной точки $f_{\Delta}(-1, -1, -1)$:

$$w = P_1 - f = [-2 \ -1 \ 1/2] - [-1 \ -1 \ -1] = [-1 \ 0 \ 3/2]$$

а внутренняя нормаль к левой грани отсекаателя равна

$$n_{\Delta} = [1 \ 0 \ 0].$$

Следовательно,

$$D \cdot n_{\Delta} = [7/2 \ 5/2 \ -1] \cdot [1 \ 0 \ 0] = 7/2 > 0$$

что соответствует нижнему пределу, а

$$w \cdot n_{\Delta} = [-1 \ 0 \ 3/2] \cdot [1 \ 0 \ 0] = -1$$

и

$$t_{\Delta} = -(-1)/(7/2) = 2/7$$

Полностью результаты работы алгоритма собраны в табл. 3.11. Анализ таблицы 3.11 показывает, что максимальным из нижних значений будет $t_H = 2/7$, а минимальным из верхних значений будет $t_B = 4/5$. Параметрическое представление отрезка P_1P_2 имеет вид:

$$P(t) = [-2 \ -1 \ 1/2] + [7/2 \ 5/2 \ -1]t$$

Подстановка сюда t_H и t_B дает:

$$P(2/7) = [-2 \ -1 \ 1/2] + [7/2 \ 5/2 \ -1] \cdot (2/7) = [-1 \ -2/7 \ 3/4]$$

для точки пересечения отрезка с левой гранью отсекаателя и

$$P(4/5) = [-2 \ -1 \ 1/2] + [7/2 \ 5/2 \ -1] \cdot (4/5) = [4/5 \ 1 \ -3/10]$$

для точки его пересечения с верхней гранью отсекаателя.

Отсечение относительно стандартной пирамиды видимости будет ненамного сложнее. Здесь внутренние нормали к граням отсекаателя следует определить формально, так как их значения неочевидны.

Пример 3.18. Отсечение относительно пирамиды видимости

Возьмем тот же отрезок, что и в примере 3.17, от $P_1(-2, -1, 1/2)$ до $P_2(3/2, 3/2, -1/2)$, который отсекается пирамидой видимости с координатами $(x_L, x_{\Pi}, x_H, x_B, x_D) = (-1, 1, -1, 1, 1, -1)$, а центр проекции расположен на $z_{\text{ЦП}} = 5$. См. рис. 3.20, б.

Значения внутренних нормалей к ближней и дальней граням очевидны. Для остальных четырех граней их можно получить, вычислив векторные произведения, образованные парами векторов, которые начинаются в центре проекции, а заканчиваются в углах соответствующих граней при $z = 0$. Эти векторы равны:

$$\begin{aligned} V_1 &= [1 \ 1 \ -5] \\ V_2 &= [-1 \ 1 \ -5] \\ V_3 &= [-1 \ -1 \ -5] \\ V_4 &= [1 \ -1 \ -5] \end{aligned}$$

Внутренние нормали равны:

$$\begin{aligned} n_B &= V_1 \otimes V_2 = [0 \ -10 \ -2] \\ n_L &= V_2 \otimes V_3 = [10 \ 0 \ -2] \\ n_H &= V_3 \otimes V_4 = [0 \ 10 \ -2] \\ n_{\Pi} &= V_4 \otimes V_1 = [-10 \ 0 \ -2] \\ n_B &= [0 \ 0 \ -1] \\ n_D &= [0 \ 0 \ 1] \end{aligned}$$

Поскольку центр проекции принадлежит четырем из шести плоскостей, несущих грани отсекаателя, то удобно выбрать граничные точки так:

$$f_B = f_L = f_H = f_{\Pi} = (0, 0, 5)$$

а для ближней и дальней граней взять их центры

$$f_B = (0, 0, 1) \text{ и } f_D = (0, 0, -1)$$

Директриса $P_1 P_2$ равна

$$D = P_2 - P_1 = [7/2 \ 5/2 \ -1]$$

Для граничной точки на левой грани отсекаателя

$$w = P_1 - f_L = [-2 \ -1 \ 1/2] - [0 \ 0 \ 5] = [-2 \ -1 \ -9/2]$$

Заметим, что

$$D \cdot n_L = [7/2 \ 5/2 \ -1] \cdot [10 \ 0 \ -2] = 37 > 0$$

Таблица 3.12.

Грань	n	f	w	w · n	D · n ¹⁾	t _H	t _B
Верхняя	[0 -10 -2]	(0, 0, 5)	[-2 -1 -9/2]	19	-23		0.826
Нижняя	[0 10 -2]	(0, 0, 5)	[-2 -1 -9/2]	-1	27	0.037	
Правая	[-10 0 -2]	(0, 0, 5)	[-2 -1 -9/2]	29	-33		0.879
Левая	[10 0 -2]	(0, 0, 5)	[-2 -1 -9/2]	-11	37	0.297	
Ближняя	[0 0 -1]	(0, 0, 1)	[-2 -1 -1/2]	1/2	1	-0.5	
Дальняя	[0 0 1]	(0, 0, -1)	[-2 -1 3/2]	3/2	-1		1.5

¹⁾ При $D \cdot n < 0$, верхний предел (t_B); при $D \cdot n > 0$, нижний предел (t_H).

значит, ищется нижний предел. Далее,

$$w \cdot n_L = [-2 -1 -9/2] \cdot [10 0 -2] = -11$$

и

$$t_H = -(-11)/37 = 11/37 = 0.297$$

Полностью результаты работы алгоритма даны в табл. 3.12. Анализ этой таблицы показывает, что максимальным из нижних значений будет $t_H = 0.297$, а минимальным из верхних будет $t_B = 0.826$. Из параметрического описания отрезка следует, что

$$P(0.297) = [-0.961 -0.258 0.203]$$

и

$$P(0.826) = [0.891 1.065 -0.323]$$

это соответствует пересечениям с левой и верхней гранями отсекателя

В последнем примере отсекателем будет нестандартное тело с семью гранями.

Пример 3.19. Отсечение относительно произвольного объема

Отсекатель изображен на рис. 3.23. Это куб, один из углов которого срезан. Грани задаются списками их вершин:

правая: (1, -1, 1), (1, -1, -1), (1, 1, -1), (1, 1, 1)

левая: (-1, -1, 1), (-1, -1, -1), (-1, 1, -1), (-1, 1, 0), (-1, 0, 1)

нижняя: (1, -1, 1), (1, -1, -1), (-1, -1, -1), (1, -1, -1)

верхняя: (1, 1, 1), (1, 1, -1), (-1, 1, -1), (-1, 1, 0), (0, 1, 1)

ближняя: (1, -1, 1), (1, 1, 1), (0, 1, 1), (-1, 0, 1), (-1, -1, 1)

дальняя: (-1, -1, -1), (1, -1, -1), (1, 1, -1), (-1, 1, -1)

косоугольная: (-1, 0, 1), (0, 1, 1), (-1, 1, 0)

В табл. 3.13 собраны все результаты работы алгоритма с отрезком от $P_1(-2, 3/2, 1)$ до $P_2(3/2, -1, -1/2)$, который отсекается относительно описанного объема.

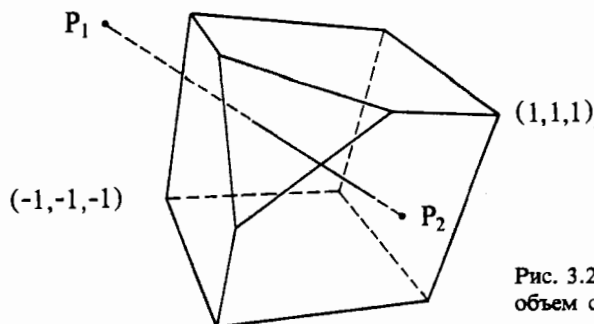


Рис. 3.23. Отсечение Кируса — Бека: объем с нечетным числом граней.

Из этой таблицы следует, что максимальным из нижних значений будет $t_H = 1/3$, а минимальным из верхних будет $t_B = 6/7$. Поэтому точками пересечения будут:

$$P(1/3) = [-5/6 \ 2/3 \ 1/2]$$

на косоугольной грани и

$$P(6/7) = [1 \ -9/14 \ -2/7]$$

на правой грани.

Таблица 3.13.

Грань	n	f	w	$w \cdot n$	$D \cdot n^{1)}$	t_H	t_B
Верхняя	[0 -1 0]	(1, 1, 1)	[-3 1/2 0]	-1/2	5/2	1/5	
Нижняя	[0 1 0]	(-1, -1, -1)	[-1 5/2 2]	5/2	-5/2		1
Правая	[-1 0 0]	(1, 1, 1)	[-3 1/2 0]	3	-7/2		6/7
Левая	[1 0 0]	(-1, -1, -1)	[-1 5/2 2]	-1	7/2	2/7	
Ближняя	[0 0 -1]	(1, 1, 1)	[-3 1/2 0]	0	3/2	0	
Дальняя	[0 0 1]	(-1, -1, -1)	[-1 5/2 2]	2	-3/2		4/3
Косо- угольная	[1 -1 -1]	(-1, 0, 1)	[-1 3/2 0]	-5/2	15/2	1/3	

¹⁾ При $D \cdot n < 0$, верхний предел (t_B); при $D \cdot n > 0$, нижний предел (t_H).

Заметим, что оценка числа операций в алгоритме Кируса — Бека растет линейно с ростом числа сторон или граней у отсекаателя.

3.12. ОТСЕЧЕНИЕ В ОДНОРОДНЫХ КООРДИНАТАХ

Когда отсечение необходимо провести в однородной системе координат [1-1], то нужно проявить осторожность, если при этом ис-

пользуется еще и преобразование проецирования. Главная причина возникающих затруднений объясняется тем, что отсекающая плоскость тут не обязательно разрезает отрезок на две части, одна из которых будет внутри, а другая — вне отсекателя. Отрезок может «заворачиваться» в бесконечности так, что внутри отсекателя будут лежать (и, следовательно, будут видимы) сразу две его части. Блинн [3-6] доказал, что отсечение всех отрезков *перед* завершением преобразования проецирования, осуществляемого путем деления всех координат на значение однородной координаты, удаляет отрезки, которые «возвращаются из бесконечности». Лианг и Барский [3-5] предложили алгоритм отсечения отрезков, работающий в однородных координатах. Они получили корректный результат путем искажения формы видимого объема, которым у них служила усеченная пирамида.

В алгоритме Кируса — Бека отрезок корректно отсекается относительно пирамиды видимости при условии, что он целиком расположен перед точкой наблюдения или, что то же самое, перед центром проекции (см. пример 3.18). Однако если отрезок заходит за центр проекции, то этот алгоритм отвергает его, даже если он частично видимый. На практике корректный результат получается, если отрезок сперва отсечь в обычной системе координат, а затем уже выполнить преобразование проецирования. Заметим, что и к отсекателю и к отрезку перед преобразованием проецирования можно применить любое аффинное преобразование (т. е. повороты, переносы и т. п.). Следующий пример проиллюстрирует вышеизложенные соображения.

Пример 3.20. Применение алгоритма Кируса — Бека к отрезку, заходящему за центр проекции

Возьмем отрезок от $P_1(0, 1, 6)$ до $P_2(0, -1, -6)$, отсекаемый пирамидой, заданной в обычной системе координат значениями $(x_d, x_p, y_n, y_b, z_b, z_d) = (-1, 1, -1, 1, 1, -1)$, причем центр проекции расположен в точке с $z = 5$. Отрезок P_1P_2 пересекает видимый объем, но начинается за центром проекции.

После выполнения преобразования проецирования (см. [1-1]) концы отрезка будут иметь следующие значения однородных координат:

$$P_1[0 \ 1 \ 6 \ -1/5] \text{ и } P_2[0 \ -1 \ -6 \ 11/5]$$

Переход к обычным координатам путем деления на однородную координату дает

$$P_1(0, -5, -30) \text{ и } P_2(0, -5/11, -30/11)$$

Заметим, что исходная точка P_1 лежала перед отсекателем, но дальше центра проекции, а теперь она, «завернувшись» в бесконечности, оказалась за отсекателем. По-

передняя: $P_{10}(0, 0, 2), P'_{10}(1, 0, 2), P_{13}(1, 3/2, 2), P_{12}(1, 2, 2), P_{11}(0, 2, 2)$
 задняя: $P'_5(1, 0, 0), P_2(0, 0, 0), P_3(0, 2, 0), P_4(1, 2, 0), P_5(1, 3/2, 0)$

V_2

левая: $P'_5(1, 0, 0), P'_{10}(1, 0, 2), P_{13}(1, 3/2, 2), P_5(1, 3/2, 0)$

правая: $P_1(3, 0, 0), P_8(3, 2, 0), P_{16}(3, 2, 2), P_9(3, 0, 2)$

правая у выемки: $P_6(3/2, 3/2, 0), P_7(3/2, 2, 0), P_{15}(3/2, 2, 2), P_{14}(3/2, 3/2, 2)$

нижняя у выемки: $P_{13}(1, 3/2, 2), P_{14}(3/2, 3/2, 2), P_6(3/2, 3/2, 0), P_5(1, 3/2, 0)$

верхняя справа: $P_{16}(3, 2, 2), P_8(3, 2, 0), P_7(3/2, 2, 0), P_{15}(3/2, 2, 2)$

нижняя: $P'_5(1, 0, 0), P_1(3, 0, 0), P_9(3, 0, 2), P'_{10}(1, 0, 2)$

После повторной обработки алгоритмом каждого из этих тел V_1 будет объявлен выпуклым, а V_2 будет разрезан на два новых тела, которые потом тоже окажутся выпуклыми. Окончательный результат в разрезанном виде показан на рис. 3.25, е.

3.15. ОТСЕЧЕНИЕ МНОГОУГОЛЬНИКОВ

Предыдущее обсуждение было связано с отсечением отрезков. Разумеется, многоугольник можно рассматривать как набор отрезков. В приложениях, связанных с вычерчиванием штриховых изображений, не слишком существенно, если многоугольник разбит на отрезки до его отсечения. Если замкнутый многоугольник отсекается, как набор отрезков, то исходная фигура может превратиться в один или более открытых многоугольников или просто стать совокупностью разрозненных отрезков, как показано на рис. 3.26. Однако если многоугольники рассматриваются как сплошные области, то необходимо, чтобы замкнутость сохранялась и у результата. Для примера на рис. 3.26 это означает, что отрезки bc , ef , fg и ha

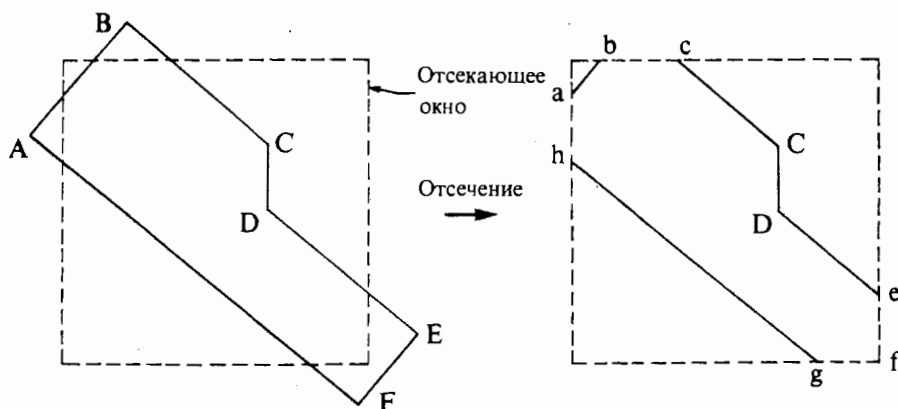


Рис. 3.26. Отсечение многоугольника: открытый многоугольник.

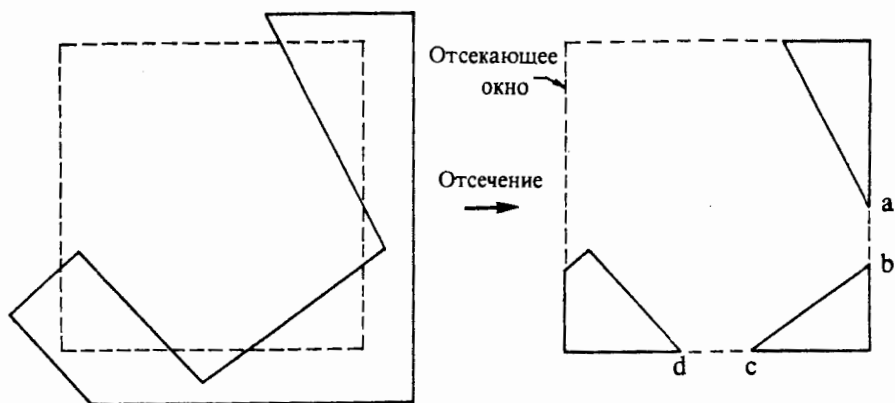


Рис. 3.27. Отсечение многоугольника: не связанные между собой многоугольники.

должны быть добавлены к описанию результирующего многоугольника. Добавление отрезков ef и fg представляет особые трудности. Трудные вопросы возникают и тогда, когда результат отсечения представляет собой несколько несвязанных между собой многоугольников меньших размеров, как это показано на рис. 3.27. Например, иногда отрезки ab и cd , показанные на рис. 3.27, включаются в описание результата. Если, например, исходный многоугольник объявлен красным на синем фоне, то отрезки ab и cd тоже будут выглядеть красными на синем фоне. Это противоречит ожидаемому результату.

3.16. ПОСЛЕДОВАТЕЛЬНОЕ ОТСЕЧЕНИЕ МНОГОУГОЛЬНИКА — АЛГОРИТМ САЗЕРЛЕНДА — ХОДЖМЕНА

Основная идея алгоритма Сазерленда — Ходжмена [3-7] состоит в том, что отсечь многоугольник относительно одной прямой или плоскости очень легко. В этом алгоритме исходный и каждый из промежуточных многоугольников отсекается последовательно относительно одной прямой. Работа алгоритма для прямоугольного окна показана на рис. 3.28. Исходный многоугольник задается списком вершин P_1, \dots, P_n , который порождает список его ребер $P_1P_2, P_2P_3, \dots, P_{n-1}P_n, P_nP_1$. На рис. 3.28 показано, что многоугольник сначала отсекается левой стороной окна, в результате чего получается промежуточная фигура. Затем алгоритм вновь отсекает эту фигуру верхней стороной окна. Получается вторая промежуточная фигура. Далее процесс отсечения продолжается с оставшимися сто-