4 Удаление невидимых граней

4.1 Алгоритм, использующий z-буфер

Это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Работает этот алгоритм в пространстве изображения.

Главное преимущество алгоритма - его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

Основной недостаток алгоритма - большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона координат z значений, то можно использовать z-буфер с фиксированной точностью. Информацию о глубине нужно обрабатывать с большей точностью, чем координатную информацию на плоскости (x, y); обычно бывает достаточно 20 бит. Буфер кадра размером 512 * 512 * 24 бит в комбинации с z-буфером размером 512 * 512 * 20 бит требует почти 1.5 мегабайт памяти. Однако снижение цен на память делает экономически оправданным создание специализированных запоминающих устройств для z-буфера и связанной ним аппаратуры.

Альтернативой созданию специальной памяти для z-буфера является использование для этой цели оперативной памяти. Уменьшение требуемой памяти достигается разбиением пространства изображения на 4, 16 или больше квадратов или полос. В предельном варианте можно использовать z-буфер размером в одну строку развертки. Для последнего случая имеется интересный *алгоритм построчного сканирования* . Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование z-буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены. Однако сортировка на плоскости, позволяющая не обрабатывать все многоугольники в каждом из квадратов или полос, может значительно сократить этот рост.

Другой недостаток алгоритма z-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания. Поскольку алгоритм заносит пикселы в буфер кадра в произвольном порядке, то нелегко получить информацию, необходимую для методов устранения лестничного эффекта, основывающихся на предварительной фильтрации. При реализации эффектов прозрачности и просвечивания пикселы могут заноситься в буфер кадра в некорректном порядке, что ведет к локальным ошибкам.

Хотя реализация методов устранения лестничного эффекта, основывающихся на префильтрации, в принципе возможна, практически это сделать трудно. Однако относительно легко реализуются методы постфильтрации (усреднение подпикселов). Напомним, что в методах устранения лестничного эффекта, основывающихся на постфильтрации, сцена вычисляется в таком пространстве изображения, разрешающая способность которого выше, чем разрешающая способность экрана.

Поэтому возможны два подхода к устранению летничного эффекта на основе постфильтрации. В первом используется буфер кадра, заданный в пространстве изображения, разрешение которого выше, чем у экрана, и z-буфер, разрешение которого

совпадает с разрешением экрана. Глубина изображения вычисляется только в центре той группы подпикселов, которая усредняется. Если для имитации расстояния от наблюдателя используется масштабирование интенсивности, то этот метод может оказаться неадекватным.

Во втором методе оба буфера, заданные в пространстве изображения, имеют повышенную разрешающую способность. При визуализации изображения как пикселная информация, так и глубина усредняются. В этом методе требуются очень большие объемы памяти. Например, изображение размером 512 * 512 * 24 бита, использующее z-буфер размером 20 бит на пиксел, разрешение которого повышено в 2 раза по осям x и y и на котором устранена ступенчатость методом равномерного усреднения, требует почти 6 мегабайт памяти.

Более формальное описание алгоритма z-буфера таково:

заполнить буфер кадра фоновым значением интенсивности или цвета;
заполнить z-буфер минимальным значением z;
преобразовать каждый многоугольник в растровую форму в произвольном порядке;
для каждого Пиксел(x, y) в многоугольнике вычислить его глубину z(x, y);
сравнить глубину z(x, y) со значением Zбуфер(x, y), хранящимся в z-буфере в этой же позиции;
если z(x, y) > Zбуфер(x, y), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить Zбуфер(x, y) на z(x, y);
в противном случае никаких действий не производить.

В качестве предварительного шага там, где это целесообразно, применяется удаление нелицевых граней.

Если известно уравнение плоскости, несущей каждый многоугольник, то вычисление глубины каждого пиксела на сканирующей строке можно проделать пошаговым способом. Напомним, что уравнение плоскости имеет вид:
$ax + by + cz + d = 0$
$z = -(ax + by + d)/c <> 0.$

Для сканирующей строки y = const. Поэтому глубина пиксела на этой строке, у которого $x_1 = x + Dx$, равна
$z_1 - z = -(ax_1 + d)/c + (ax + d)/c = a(x - x_1)/c$ или
$z_1 = z - (a/c)Dx.$
Но Dx = 1, поэтому $z_1 = z - (a/c)$.

Дальнейшей иллюстрацией алгоритма послужит следующий пример.

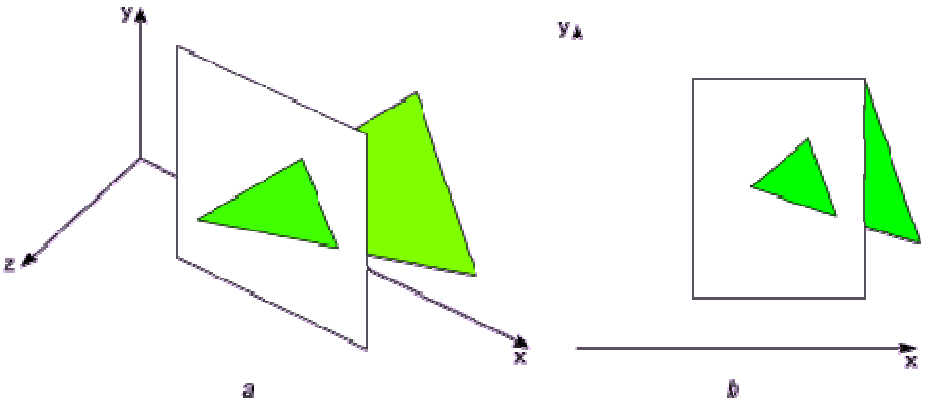

рис. 21.1

Пример. Алгоритм, использующий z-буфер. Рассмотрим многоугольник, координаты угловых точек которого равны P_1(10, 5, 10), P_2(10, 25, 10), P_3(25, 25, 10), P_4(25, 5, 10) и треугольник с вершинами P_5(15, 15, 15), P_6(25, 25, 5), P_7(30, 10, 5). Треугольник протыкает прямоугольник, как показано на рис. 21.1. Эти многоугольники нужно изобразить на экране с разрешением 32 * 32, используя простой буфер кадра с двумя битовыми плоскостями. В этом буфере фон обозначен через 0, прямоугольник - через 1, а треугольник - через 2. Под z-буфер отводится 4 битовых плоскости размером 32 * 32 бит. Таким образом, содержимое z-буфера окажется в диапазоне от 0 до 16. Точка наблюдения расположена в бесконечности на положительной полуоси z, как показано на рис. 21.1b.

Вначале и в буфере кадра, и в z-буфере содержатся нули. После растровой развертки прямоугольника содержимое буфера кадра будет иметь следующий вид:
Напомним, что в левом нижнем углу находится пиксел (0, 0).
Используя метод Ньюэла, получаем уравнение плоскости треугольника 3x+y+4z-120 = 0.
Значит, глубина треугольника в любой его точке задается уравнением z = -(3x + y - 120)/4.
Для последовательных пикселов, лежащих на сканирующей строке z_1 = z - 3/4.
Вычисление пересечений сторон треугольника со сканирующими строками развертки с учетом соглашения о половине интервала между соседними сканирующими строками дает следующие пары координат (25.2, 24.5), (25.5, 23.5), (25.8, 22.5), (26.2, 21.5), (26.5, 20.5), (26.8, 19.5), (27.2, 18.5), (27,5, 17.5), (27.8, 16.5), (28.2, 15.5), (28.5, 14.5), (28.8, 13.5), (29.2, 12.5), (29.5, 11.5), (29.8, 10.5) для строк от 24 до 10. Напомним, что активируется тот пиксел, у которого центр лежит внутри или на границе треугольника, то есть при x_1 <= x <= x_2. Преобразование в растровую форму и сравнение глубины каждого пиксела со значением z-буфера дает новое состояние буфера кадра:
Для примера рассмотрим пиксел (20, 15). Оценка z в центре этого пиксела дает z = -[(3), (20.5) + 15.5 - 120]/4 = 43/4 = 10.75.
Сравнивая его со значением z-буфера в точке (20, 15) после обработки прямоугольника, видим, что треугольник здесь расположен перед прямоугольником. Поэтому значение буфера кадра в точке (20, 15) заменяется на 2. Поскольку в нашем примере z-буфер состоит лишь из 4 битовых плоскостей, он может содержать числа только в диапазоне от 0 до 15. Поэтому значение z округляется до ближайшего целого числа. В результате в ячейку (20, 15) z-буфера заносится число 11.
Линия пересечения треугольника с прямоугольником получается при подстановке z = 10 в уравнение плоскости, несущей треугольник. Результат таков:
3x + y - 80 = 0.
Пересечения этой прямой с ребрами треугольника происходят в точках (20, 20) и (22.5, 12.5). Линия пересечения, на которой треугольник становится видимым, хорошо отражена в буфере кадра.
Алгоритм, использующий z-буфер, можно также применить для построения сечений поверхностей. Изменится только оператор сравнения:
z(x, y) > Zбуфер(x, y) **and** z(x, y) <= Zсечения
где Zсечения - глубина искомого сечения. Эффект заключается в том, что остаются только такие элементы поверхности, которые лежат на самом сечении или позади него.

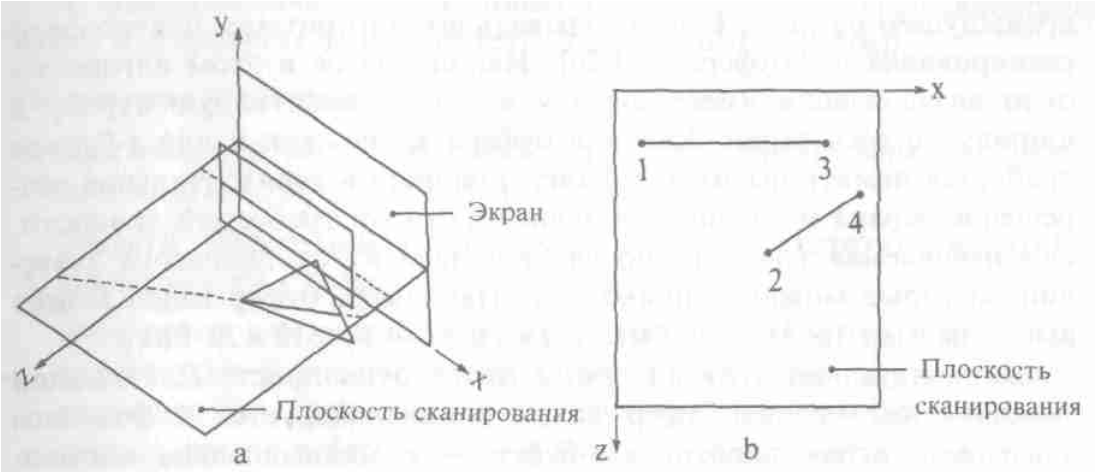

Рис. 4.57. Сканирующая плоскость.

Одним из простейших алгоритмов **построчного сканирования**, который решает задачу удаления невидимых поверхностей, является специальный **случай алгоритма z-буфера**, который обсуждался в предыдущем разделе. Будем называть его алгоритмом построчного сканирования с z-буфером [4-26]. Используемое в этом алгоритме окно визуализации имеет высоту в одну сканирующую строку и ширину во весь экран. Как для буфера кадра, так и для z-буфера требуется память высотой в 1 бит, шириной в горизонтальное разрешение экрана и глубиной в зависимости от требуемой точности. Обеспечиваемая точность по глубине зависит от диапазона значений, которые может принимать z. Например, буфер кадра может иметь размер 1×512×24 бит, а z-буфер - 1×512×20 бит.

Концептуально этот алгоритм достаточно прост. Для каждой сканирующей строки буфер кадра инициализируется с фоновым значением интенсивности, а z-буфер - с минимальным значением z. Затем определяется пересечение сканирующей строки с двумерной проекцией каждого многоугольника сцены, если они существуют. Эти пересечения образуют пары, как указывалось в разд. 2.19. При рассмотрении каждого пиксела на сканирующей строке в интервале между концами пар его глубина сравнивается с глубиной, содержащейся в соответствующем элементе z-буфера. Если глубина этого пиксела больше глубины из z-буфера., то рассматриваемый отрезок будет текущим видимым отрезком. И следовательно, атрибуты многоугольника, соответствующего данному , отрезку, заносятся в буфер кадра в позиции данного пиксела; соответственно корректируется и z-буфер в этой позиции. После обработки всех многоугольников сцены буфер кадра размером в одну сканирующую строку содержит решение задачи удаления невидимых поверхностей для данной сканирующей строки. Он выводится на экран дисплея в порядке, определяемом растровым сканированием, т. е. слева направо. В этом алгоритме можно использовать методы устранения ступенчатости, основывающиеся как на пре-, так и на постфильтрации.

Однако практически сравнение каждого многоугольника с каждой сканирующей строкой оказывается неэффективным. Поэтому используется некоторая разновидность списка упорядоченных ребер, которая обсуждалась в разд. 2.19. В частности, для повышения эффективности этого алгоритма применяются групповая сортировка (является одной из разновидностей распределяющей сортировки) по оси y, список активных многоугольников и список активных ребер.

С использованием этих методов алгоритм построчного сканирования с z-буфером формулируется следующим образом:

Подготовка информации:

Для каждого многоугольника определить самую верхнюю сканирующую строку, которую он пересекает.

Занести многоугольник в группу y, соответствующую этой сканирующей строке.

Запомнить для каждого многоугольника, например в связном списке, как минимум следующую информацию: Δy - число сканирующих строк, которые пересекаются этим многоугольником; список ребер многоугольника; коэффициенты (*a,b,c,d*) уравнения плоскости многоугольника; визуальные атрибуты многоугольника.

Решение задачи удаления невидимых поверхностей: Инициализировать буфер кадра дисплея. Для каждой сканирующей строки:

Инициализировать буфер кадра размером с одну сканирующую строку, заполнив его фоновым значением.

Инициализировать z-буфер размером с одну сканирующую строку значением z_{min}.

Проверить появление в группе y сканирующей строки новых многоугольников. Добавить все новые многоугольники к списку активных многоугольников.

Проверить появление новых многоугольников в списке активных многоугольников. Добавить все пары ребер новых многоугольников к списку активных ребер.

Если какой-нибудь элемент из пары ребер многоугольника удаляется из списка активных ребер, то надо определить, сохранился ли соответствующий многоугольник в списке активных многоугольников. Если сохранился, то укомплектовать пару активных ребер этого многоугольника в списке активных ребер. В противном случае удалить и второй элемент пары ребер из списка активных ребер.

В списке активных ребер содержится следующая информация для каждой пары ребер многоугольника, которые пересекаются сканирующей строкой:

$x_Л$ — пересечение левого ребра из пары с текущей сканирующей строкой.
$\Delta x_Л$ — приращение $x_Л$ в интервале между соседними сканирующими строками.
$\Delta y_Л$ — число сканирующих строк, пересекаемых левой стороной.
$x_П$ — пересечение правого ребра из пары с текущей сканирующей строкой.
$\Delta x_П$ — приращение $x_П$ в интервале между соседними сканирующими строками.
$\Delta y_П$ — число сканирующих строк, пересекаемых правой стороной.
$z_Л$ — глубина многоугольника в центре пиксела, соответствующего левому ребру.
Δz_x — приращение по z вдоль сканирующей строки. Оно равно a/c при $c \neq 0$.
Δz_y — приращение по z в интервале между соседними сканирующими строками. Оно равно b/c при $c \neq 0$.

Пары активных ребер многоугольников заносятся в соответствующий список в произвольном порядке. В пределах одной пары пересечения упорядочены слева направо. Для одного многоугольника может оказаться более одной пары активных ребер (В силу его невыпуклости).

Для каждой пары ребер многоугольника из списка активных ребер:

Извлечь эту пару ребер из списка активных ребер.

Инициализировать z со значением $z_Л$.

Для каждого пиксела, такого, что $x_Л <= x+1/2 <= x_П$, вычислить глубину z(x + 1/2, y + 1/2) в его центре, используя уравнение плоскости многоугольника. Для сканирующей строки это сведется к вычислению прира-щения: $z_{x+\Delta x} = z_x - \Delta z_x$.

Сравнить глубину z(x + 1/2, y + 1/2) с величиной Zбуфер(x), хранящейся в z-буфере для одной сканирующей строки. Если z(x + 1/2, y + 1/2) > Zбуфер(x), то занести атрибуты многоугольника в буфер кадра для одной сканирующей строки и заменить Zбуфер(x) на z(x + 1/2, y + 1/2). В противном случае не производить никаких действий.

Записать буфер кадра для сканирующей строки в буфер кадра дисплея.

Скорректировать список активных ребер:

Для каждой пары ребер многоугольника определить $\Delta y_Л$ и $\Delta y_П$. Если $\Delta y_Л$ или $\Delta y_П < 0$, то удалить соответствующее ребро из списка. Пометить положение обоих ребер в списке и породивший их многоугольник.

Вычислить новые абсциссы пересечений:

$x_{лнов} = x_{лстар} + \Delta x_л$
$x_{пнов} = x_{пстар} + \Delta x_п$

Вычислить глубину многоугольника на левом ребре, используя уравнение плоскости этого многоугольника. Между сканирующими строками это сведется к вычислению приращения:

$z_{лнов} = z_{лстар} - \Delta z_л \Delta x - \Delta z_y$

Сократить список активных многоугольников. Если $\Delta y < 0$ для какого-нибудь многоугольника, то удалить его из списка.

Как и раньше используется предварительное удаление нелицевых плоскостей. Следующий пример послужит более полной иллюстрацией этого алгоритма.

4.2 Алгоритм построчного сканирования с z-буфером

Вновь возьмем прямоугольник и треугольник, ранее рассмотренные в примере 4.19. Напомним координаты углов прямоугольника: $P_1(10, 5, 10)$, $P_2(10, 25, 10)$, $P_3(25, 25, 10)$, $P_4(25, 5, 10)$ и вершин треугольника: $P_5(15, 15, 15)$, $P_6(25, 25, 5)$, $P_7(30, 10, 5)$, показанных на рис. 4.50. Разрешение экрана осталось равным 32×32×2 бита. Как и раньше, атрибут видимости фона будет равен 0, прямоугольника — 1, а треугольника — 2. Точка наблюдения находится в бесконечности на положительной полуоси z. Используя соглашение о половине интервала между сканирующими строками, видим, что самая верхняя сканирующая строка, пересекающая оба многоугольника, имеет y = 24. Поэтому только группа с y = 24 содержит какую-то информацию. Все остальные группы пусты.

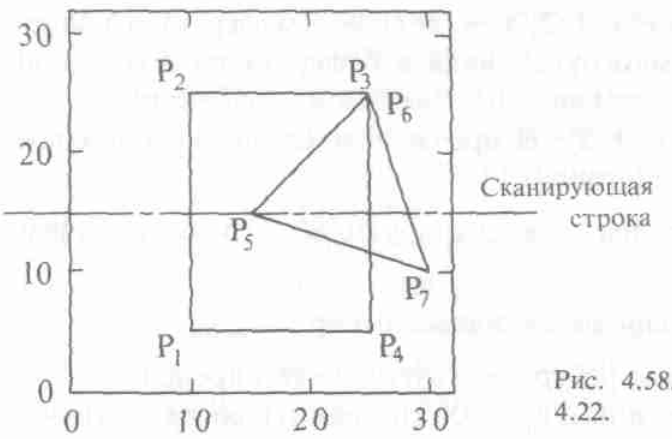

Рис. 4.58. Многоугольники из примера 4.22.

Список активных многоугольников при y = 24 содержит для прямоугольника (многоугольник 1) и треугольника (многоугольник 2) следующую информацию:

прямоугольник: 19, 2, P_1P_2, P_3P_4, 0, 0, 1, -10, 1;
треугольник: 14, 3, P_5P_6, P_6P_7, P_7P_5,3, 1, 4, -120, 2.

Элементы этого списка суть соответственно Δy, число ребер, список ребер, коэффициенты уравнений несущей плоскости (*a, b, c, d*) и номер многоугольника. Заметим, что для прямоугольника этот список содержит только два ребра. Горизонтальные ребра игнорируются.

Для сканирующей строки 15 (рис. 4.58) в списке активных многоугольников содержатся оба многоугольника. Для прямоугольника Δy = 11. Для треугольника Δy = 5. Вначале список активных ребер содержит две пары пересечений; первую -для прямоугольника и вторую - для треугольника:

прямоугольник: 10, 0, 19, 25, 0, 19, 10, 0, 0
треугольник: 24 1/2, -1, 9, 25 1/6, 1/3, 14, 5 1/2, 3/4, 1/4

Элементы этого списка суть соответственно $x_Л$, $\Delta x_Л$, $\Delta y_Л$, $x_П$, $\Delta x_П$, $\Delta y_П$, $z_Л$, Δz_X, Δz_Y. Непосредственно перед обработкой сканирующей строки 15 список активных ребер содержит:

прямоугольник: 10, 0, 10, 25, 0, 10, 10, 0, 0
треугольник : 15 1/2, -1,0, 28 1/6, 1/3, 5, 14 1/2, 3/4, 1/4

После первого обнуления буфера кадра и z-буфера для одной строки и последующей растровой развертки прямоугольника эти буферы будут содержать:

<table>
<tr><td colspan="33">Буфер кадра для строки</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="33">z-буфер для строки</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>10</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Рассмотрим теперь треугольник. На левом ребре z = 14.5, что больше значения Zбуфер(15) = 10. Поэтому атрибуты треугольника заносятся в буфер кадра, корректируется и z-буфер для сканирующей строки. Эти результаты после растровой

развертки треугольника таковы:

Буфер кадра для строки																															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	2	2	2	2	2	2	2	1	1	1	1	2	2	0	0	0	0	0
z-буфер для строки																															
0	0	0	0	0	0	0	0	0	0	10	10	10	10	15	14	13	12	12	11	10	10	10	10	10	6	6	0	0	0	0	0

Здесь значения в z-буфере округлены до ближайших целых для экономии памяти. Тот же результат получился для соответствующей сканирующей строки и в примере 4.19. Для визуализации буфер кадра копируется слева направо.

Теперь корректируется список активных ребер. В результате его сокращения $\Delta y_Л = -1 < 0$. Поэтому ребро P_6P_5 удаляется из списка активных ребер, а треугольник помечается. Корректировка правого ребра треугольника дает:

$x_{пнов} = x_{пстар} + \Delta x_п = 28\ 1/6 + 1/3 = 28\ 1/2$
$\Delta y_{пнов} = \Delta y_{пстар} - 1 = 5 - 1 = 4$

После корректировки списка активных ребер сокращается список активных многоугольников. Поскольку прямоугольник остается в этом списке, то следующий цикл алгоритма вставит ребро P_5P_7 в список активных ребер в помеченной позиции. Пересечение сканирующей строки 14 ($y = 14.5$) с ребром P_5P_7 дает новое значение $x_Л = 16\ 1/2$. Глубина треугольника равна:

$$z_Л = -x[ax + by + d]/c = -[(3)(16.5) + (1)(14.5) - 120]/4 = 14$$

Окончательный список активных ребер для сканирующей строки 14 таков:

прямоугольник: 10, 0, 9, 25, 0, 9, 10, 0, 0
треугольник: 16 1/2, 3, 4, 25 1/2, 4, 14, 3/4, 1/4

Полные результаты приведены в примере 4.19.

4.3 Интервальный алгоритм построчного сканирования

В алгоритме построчного сканирования с использованием z-буфера глубина многоугольника вычисляется для каждого пиксела на сканирующей строке. Количество вычислений глубины можно уменьшить, если использовать понятие интервалов, впервые введенных в алгоритме Ваткинса [4-25]. На рис. 4.59, а показано пересечение двух многоугольников со сканирующей плоскостью. Решение задачи удаления невидимых поверхностей сводится к выбору видимых отрезков в каждом из интервалов, полученных путем деления сканирующей строки проекциями точек пересечения ребер (рис. 4.59, а). Из этого рисунка видно, что возможны только три варианта:

Рис. 4.59. Интервалы для построчного сканирования.

Интервал пуст, как, например, интервал 1 на рис. 4.59, a. В этом случае изображается фон.

Интервал содержит лишь один отрезок, как, например, интервалы 2 и 4 на рис. 4.59, a. В этом случае изображаются атрибуты многоугольника, соответствующего отрезку.

Интервал содержит несколько отрезков, как, например, интервал 3 на рис. 4.59, a. В этом случае вычисляется глубина каждого отрезка в интервале. Видимым будет отрезок с максимальным значением z. В этом интервале будут изображаться атрибуты видимого отрезка.

Если многоугольники не могут протыкать друг друга, то достаточно вычислять глубину каждого отрезка в интервале на одном из его концов. Если два отрезка касаются, но не проникают в концы интервала, то вычисление глубины производится в середине интервала, как показано на рис. 4.59, b. Для интервала 3 вычисление глубины, проведенное на его левом конце, не позволяет принять определенное решение. Реализация вычисления глубины в центре интервала дает уже корректный результат, как показано на рис. 4.59, b.

Если многоугольники могут протыкать друг друга, то сканирующая строка делится не только проекциями точек пересечения ребер со сканирующей плоскостью, но и проекциями точек их попарного пересечения, как показано на рис. 4.59, c. Вычисление глубины в концевых точках таких интервалов будет давать неопределенные результаты. Поэтому здесь достаточно проводить вычисление глубины в середине каждого интервала, как показано для оси x на рис. 4.59, c.

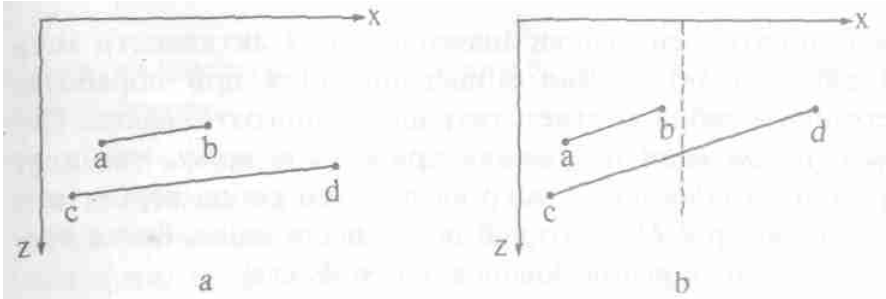

Рис. 4.60. Другой интервальный метод.

Используя более сложные методы порождения интервалов, можно сократить их число, а следовательно, и вычислительную трудоемкость. Однако применение простых средств также может привести к удивительным результатам. Например, Ваткинс [4-25] предложил простой метод разбиения отрезка средней точкой. Простым сравнением глубин концевых точек отрезков *ab* и *cd*, изображенных на рис. 4.60, a, убеждаемся, что отрезок *cd* целиком видим. Однако случай, изображенный на рис. 4.60, b, показывает, что так бывает не всегда (Поскольку глубины точек *b,d* здесь равны). Деление этой сцены прямой, проходящей через середину отрезка *cd*, сразу делает очевидным, что оба куска *cd* видимы.

Далее, иногда удается вообще избежать вычислений глубины. Ромни и др. [4-27] показали, что если многоугольники не могут протыкать друг друга и если текущая сканирующая строка пересекает те же самые многоугольники, что и предыдущая, и если порядок следования точек пересечения ребер при этом не изменится, то не изменяется и приоритет глубины отрезков в пределах каждого интервала. Значит, не нужно вычислять приоритет глубины отрезков для новой сканирующей строки. Хэмлин и Джир [4-28] показали, как при некоторых условиях можно поддерживать приоритет глубины даже тогда, когда порядок следования точек пересечения ребер изменяется.

Основная структура алгоритма построчного сканирования с z-буфером применима также в случае интервального алгоритма построчного сканирования типа Уоткинса. Необходимо изменить только внутренний цикл, т. е. способ обработки отдельной сканирующей строки и содержимое списка активных ребер. Здесь не нужно следить за парностью пересечений ребер многоугольника со сканирующей строкой. Ребра заносятся в список активных ребер индивидуально. Этот список упорядочивается по возрастанию x. Для определения левого и правого ребер из пары здесь используются идентификатор многоугольника и флаг активности многоугольника. В начале сканирующей строки значение флага активности многоугольника равно нулю, и оно модифицируется при обработке каждого очередного ребра соответствующего многоугольника. Появление левого ребра многоугольника приведет к тому, что этот флаг станет равным единице, а встреча правого ребра вернет ему значение нуль. Пример 4.23, который приводится ниже, более подробно проиллюстрирует использование этого флага.

Интервалы, занимаемые каждым многоугольником, можно определять по мере обработки сканирующих строк. Если многоугольники не могут протыкать друг друга, то пересечение каждого ребра из списка активных ребер определяет границу интервала. Как указывалось выше, число активных многоугольников в пределах интервала определяет способ его обработки. Вычисление глубины проводится только тогда, когда в интервале содержится более одного активного многоугольника. Если же протыкание допустимо и в пределах интервала, определенного пересечением ребер, имеется более чем один активный многоугольник, то необходимо проверить возможность пересечения отрезков внутри этого интервала (рис. 4.59, с). Для осуществления этой проверки удобен метод, заключающийся в сравнении знаков разностей глубин пар отрезков в концевых точках интервала. Необходимо проверить каждую пару отрезков в таком интервале. Например, если глубины двух отрезков на левом и правом концах интервала равны $z_{1л}, z_{1п}, z_{2л}, z_{2п}$ то

$$\text{if } \operatorname{sign}(z_{1л} - z_{2л}) \neq \operatorname{sign}(z_{1п} - z_{2п}) \quad (4.9)$$

значит, эти отрезки пересекаются. Если отрезки пересекаются, то интервал подразбивается точкой пересечения. Этот процесс повторяется с левым подынтервалом до тех пор, пока он не станет свободным от пересечений. Для свободных интервалов вычисления глубин производятся в их серединах.

Если один из знаков равен нулю, то отрезки пересекаются в конце интервала. В таком случае достаточно определить глубину на противоположном конце интервала вместо того, чтобы проводить его разбиение.

Структура интервального алгоритма построчного сканирования такова:

Подготовка данных:

Определить для каждого многоугольника самую верхнюю сканирующую строку, пересекающую его.

Занести многоугольник в группу y, соответствующую этой сканирующей строке.

Запомнить в связном списке для каждого многоугольника как минимум следующую информацию: Δy - число сканирующих строк, которые пересекаются этим многоугольником; список ребер многоугольника; коэффициенты уравнения несущей его плоскости (a,b,c,d); визуальные атрибуты многоугольника.

Решение задачи удаления невидимых поверхностей: Для каждой сканирующей строки:

Проверить появление в группе у сканирующей строки новых многоугольников. Добавить все новые многоугольники к списку активных многоугольников.

Проверить появление новых многоугольников в списке активных многоугольников. Добавить все ребра новых активных ребер. Для каждого ребра многоугольника, которое пересекается со сканирующей строкой, содержится следующая информация:

x — абсцисса пересечения ребра с текущей сканирующей строкой.
Δx — приращение по x между соседними сканирующими строками.
Δy — число сканирующих строк, пересекаемых данным ребром.
P — идентификатор многоугольника.

Флаг — признак, показывающий активность данного многоугольника на текущей сканирующей строке.

Упорядочить список активных ребер по возрастанию x.

Обработать список активных ребер. Подробности обработки даны на блок-схеме, приведенной на рис. 4.61, и ее модификациях на рис. 4.62 и 4.63.

Скорректировать список активных ребер:

Для каждого пересечения ребра определить Δy. Если $\Delta y < 0$, то удалить это ребро из списка активных ребер.

Вычислить новую абсциссу пересечения:

$z_{нов} = x_{стар} + \Delta x$

Сократить список активных многоугольников:

Для каждого многоугольника уменьшить Δy . Если $\Delta y < 0$ для какого-то многоугольника, то удалить этот многоугольник из списка.

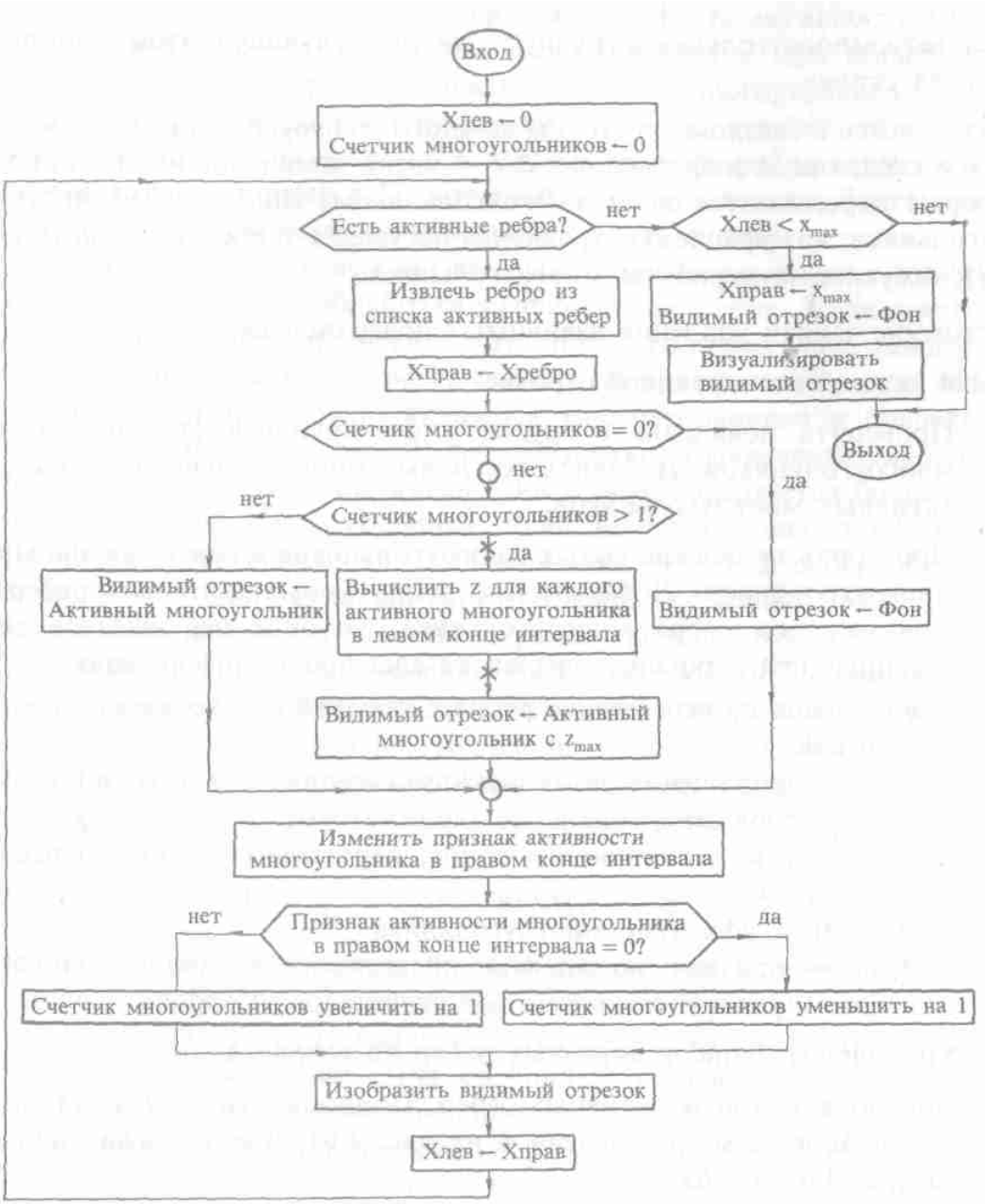

Рис. 4.61. Блок-схема интервального алгоритма для непротыкающих многоугольников.

В данном алгоритме не используются преимущества, обеспечиваемые когерентностью приоритетов по глубине, которая была предложена Ромни. Если многоугольники не могут протыкать друг друга, то модификация этого алгоритма, построенная с учетом когерентности по глубине, приводит к значительному его улучшению.

Рис. 4.62. Модификация блока вычисления глубины для блок-схемы с рис. 4.61.

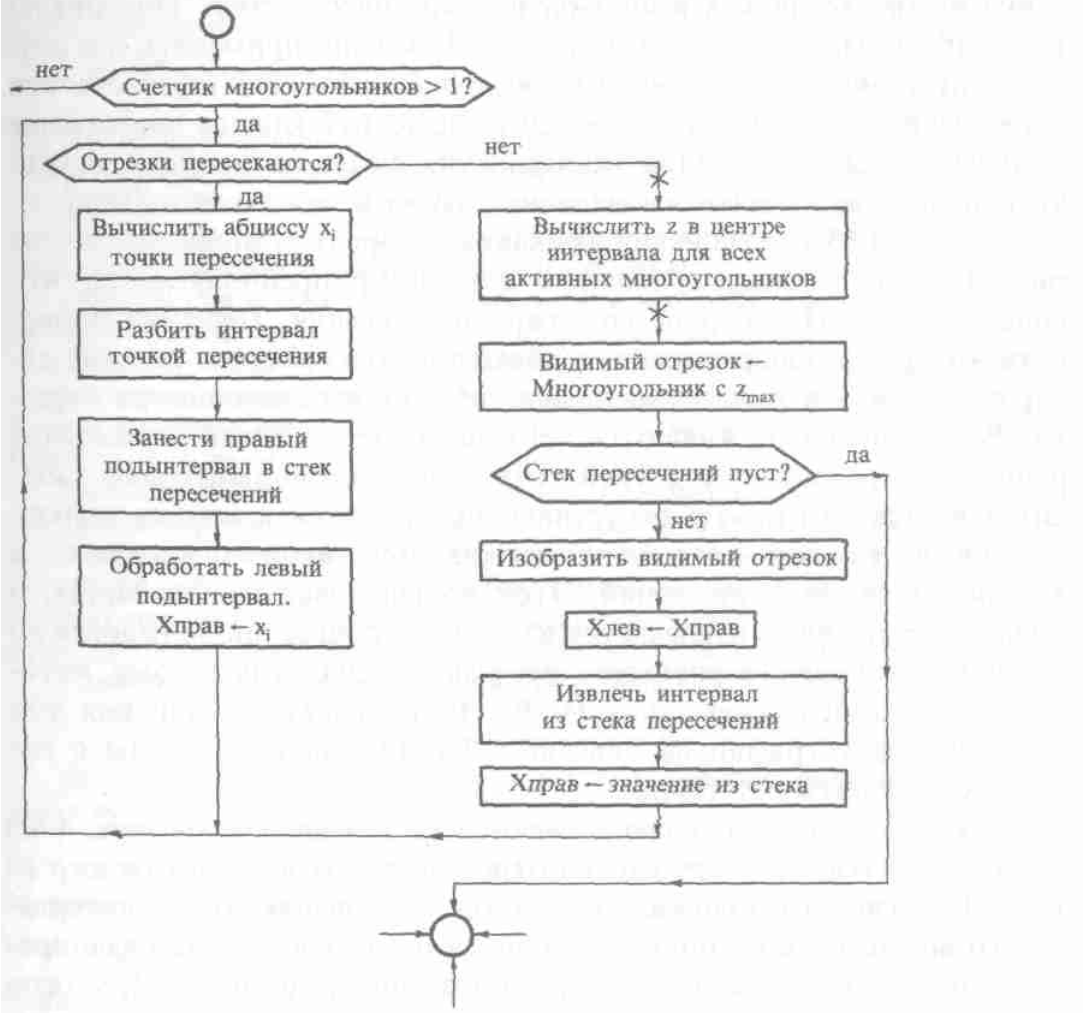

Рис. 4.63. Модификация блок-схемы с рис. 4.61 для протыкающих многоугольников.

В простом интервальном алгоритме, приведенном на рис. 4.61, предполагается, что отрезки многоугольников в пределах одного интервала не пересекаются. Если же отрезки пересекаются в концевых точках интервала, то, как указывалось ранее, вычисление глубины для этих отрезков производится в противоположных им концевых точках данного интервала. Простая модификация блока вычисления глубины в блок-схеме, показанной на рис. 4.61, приведена на рис. 4.62.

Если отрезки пересекаются внутри интервала, т. е. если многоугольники протыкают друг друга, то либо нужно использовать более сложный алгоритм разбиения на интервалы, либо точки пересечения нужно вставлять в упорядоченный список ребер. Интервальный алгоритм, показанный на рис. 4.61, можно применять и в случае допустимости пересечения многоугольников, если включить эти пересечения в список активных ребер,

пометить каждое пересечение флагом, изменить порядок модификации флага активности многоугольника и выполнять вычисления глубин в центре интервала.

На рис. 4.63 показана модификация алгоритма, приведенного на рис. 4.61. В модифицированном алгоритме предполагается, что список активных ребер не содержит пересечения. Пересекающиеся отрезки нужно обнаруживать и обрабатывать сразу же по ходу алгоритма. Здесь в каждом интервале ищутся пересекающиеся отрезки. Если они обнаруживаются, что вычисляется точка пересечения и интервал разбивается в этой точке. Правый подынтервал заносится в стек. Алгоритм рекурсивно применяется к левому подынтервалу до тех пор, пока не обнаруживается такой подынтервал, в котором уже нет пересечений. Этот подынтервал изображается, и новый подынтервал извлекается из стека. Процесс продолжается до тех пор, пока стек не опустеет. Этот метод аналогичен тому, который предлагался Джексоном [4-29]. Заслуживает упоминания тот факт, что фотографии на вклейке (12 и 13) были получены с помощью алгоритма Ваткинса.

Для простоты в модифицированной версии алгоритма (рис. 4.63) предполагается, что пересечения отрезков лежат внутри каждого из них. Поскольку отрезки могут касаться в концевых точках интервала, то вычисление глубины проводится в его центре. Модифицированный блок вычисления глубины, показанный на рис. 4.62, можно подставить в схему на рис. 4.63, чтобы сократить объем вычислений. Дополнительная модификация этого алгоритма реализует сортировку по глубине интервалов, содержащих точки пересечения отрезков, чтобы определить видимость пересекающихся отрезков прежде, чем производить разбиение интервала. Эта модификация позволяет сократить количество подынтервалов и повысить эффективность алгоритма. Если необходимо, то для повышения эффективности используется также предварительное удаление нелицевых граней.

Пример 4.23. Интервальный алгоритм построчного сканирования

Рассмотрим опять прямоугольник и протыкающий его треугольник, которые уже обсуждались в примерах 4.19 и 4.22. Возьмем сканирующую строку 15. Будем использовать соглашение о половине интервала между сканирующими строками. Пересечение сканирующей плоскости при y = 15.5 с многоугольниками изображено на рис. 4.64. На рис. 4.58 показана проекция этих многоугольников на плоскость xy. Непосредственно перед началом обработки сканирующей строки 15 список активных ребер, упорядоченный по возрастанию значения x, содержит следующие величины:

10, 0, 10, 1, 0, 15 1/2, -1,0, 2, 0, 25, 0, 10, 1, 0, 28 1/6, 1/3, 5, 2, 0

где числа сгруппированы в пятерки, соответствующие значениям (x, Δx, Δy, P, Флаг), которые были определены в алгоритме выше. На рис. 4.64 показаны пять интервалов, порожденных четырьмя точками пересечения активных ребер со сканирующей строкой. Из этого же рисунка видно, что отрезки, соответствующие многоугольникам, пересекаются внутри третьего интервала.

Сканирующая строка обрабатывается слева направо в порядке, определяемом растровым сканированием. В первом интервале нет многоугольников. Он изображается (пикселы от 0 до 9) с фоновым значением атрибутов. Во втором интервале становится активным прямоугольник. Его флаг изменяется на 1:

Флаг = - Флаг + 1.

В этом интервале нет других многоугольников. Следовательно, он изображается (пикселы от 10 до 14) с атрибутами прямоугольника.

Третий интервал начинается при x = 15.5. Становится активным треугольник, и его флаг изменяется на 1, счетчик многоугольников возрастает до двух, значением левой абсциссы интервала (Хлев) становится 15.5, и следующее ребро при x = 25 выбирается из списка активных ребер. Значением правой абсциссы интервала (Хправ) становится 25. Поскольку значение счетчика многоугольников больше единицы, то соответствующие им отрезки проверяются на пересечение (рис. 4.63).

Уравнение плоскости треугольника (см. пример 4.19) таково:

$3x + y + 4z - 120 = 0$

Для сканирующей строки 15 значение y = 15.5; а глубина треугольника для пиксела с абсциссой x равна

$z = (120 - y - 3x)/4=(120 - 15.5 - 3x)/4 = (104.5 - 3x)/4$

Поэтому в центрах пикселов, т. е. при x + 1/2, на концах интервала:

$z_{2Л} = [104.5 - (3)\ (15.5)]/4\text{-}14.5$; $z_{2П} = [104.5 - (3)\ (25.5)]/4\text{-}7.0$

Поскольку глубина прямоугольника постоянна, то

$z_{1Л} = 10$; $z_{1П} = 10$

Рис. 4.64. Сканирующая плоскость для примера 4.23.

Из уравнения (4.9) имеем:

$\text{Sign}(z_{1Л} - z_{2Л}) = \text{Sign}(10 - 14.5) < 0.$

$\text{Sign}(z_{1П} - z_{2П}) = \text{Sign}(10 - 7) > 0$

Поскольку $\text{Sign}(z_{1Л} - z_{2Л}) \neq \text{Sign}(z_{1П} - z_{2П})$, эти отрезки пересекаются. Координаты точки пересечения таковы:

$z = (120- 15.5\text{-}3x)/4 = 10\ x_i = 21.5$

Интервал разбивается точкой с абсциссой x = 21.5. Значение Хправ заносится в стек. Теперь значением Хправ становится x_i т. е. 21.5.

В подынтервале от x = 15.5 до x = 21.5 нет пересечений. Глубина треугольника в центре этого подынтервала, т. е. при x = 18.5, равна:

$$z_2 = (104.5 - 3x)/4 = [104.5 - (3)(18.5)]/4 = 12.25$$

что больше, чем $z_1 = 10$ для треугольника. Поэтому в этом подынтервале изображается треугольник (пикселы от 15 до 20).

Затем значением Хлев становится значение Хправ, и правый подынтервал извлекается из стека. Значением Хправ становится величина, извлеченная из стека, т. е. x = 25. В подынтервале от x = 21.5 до x = 25 нет пересечений. Глубина треугольника в центре этого подынтервала, т. е. при x = 23.25 равна

$z_2 = (104.5 - 3x)/4 = [104.5 - (3)(23.25)]/4 = 8.69$

что меньше, чем $z_1 = 10$ для прямоугольника. Поэтому в этом подынтервале (пикселы от 21 до 24) видимым является прямоугольник.

Стек пересечений теперь пуст. Процедура, приведенная на рис. 4.63, передает управление процедуре, изображенной на рис. 4.61. В правом конце интервала находится прямоугольник. Он и перестает быть активным. Его флаг становится равным 0, что приводит к уменьшению счетчика многоугольников на 1. Сегмент изображается с атрибутами прямоугольника. Хлев опять заменяется на Хправ.

Следующим ребром, извлеченным из списка активных ребер, будет ребро треугольника со значением x = 28 1/6. Четвертый интервал простирается от x = 25 до x = 28 1/6.

Счетчик многоугольников здесь равен 1. Активен в этом интервале только треугольник. Поэтому отрезок изображается с атрибутами треугольника (пикселы от 25 до 27). В правом конце интервала находится треугольник. Его флаг становится равным 0, и он теперь неактивен. Счетчик многоугольников становится равным 0. Хлев заменяется на Хправ, т. е. на 26 1/6.

Теперь в списке активных ребер ничего нет. Здесь $x_{max} = 32$, поэтому Хлев < x_{max} .Значит, Хправ заменяется на x_{max} и последний интервал (пикселы от 28 до 31) изображается с фоновыми атрибутами. Затем Хлев заменяется на Хправ. Опять нет активных ребер, но Хлев = x_{max} , и обработка сканирующей строки завершается.

Окончательный результат совпадает с тем, что показан на рис. 4.19. Алгоритмы построчного сканирования можно реализовать и как алгоритмы удаления невидимых линий. Например, Арчулета [4-30] предложил такую версию алгоритма Ваткинса, которая занимается

4.4 Алгоритм Варнока

Основные идеи, на которые опирается алгоритм Варнока, обладают большой общностью. Они основываются на гипотезе о способе обработки информации, содержащейся в сцене, глазом и мозгом человека. Эта гипотеза заключается в том, что тратится очень мало времени и усилий на обработку тех областей, которые содержат мало информации. Большая часть времени и труда затрачивается на области с высоким информационным содержимым.

В качестве примера рассмотрим поверхность стола, на которой нет ничего, кроме вазы с фруктами. Для восприятия цвета, фактуры и других аналогичных характеристик всей поверхности стола много времени не нужно. Все внимание сосредоточивается на вазе с фруктами. В каком месте стола она расположена? Велика ли она? Из какого материала сделана: из дерева, керамики, пластика, стекла, металла? Каков цвет вазы: красный, синий, серебристый; тусклый или яркий и т. п.? Какие фрукты в ней лежат: персики, виноград, груши, бананы, яблоки? Каков цвет яблок: красный, желтый, зеленый? Есть ли у яблока хвостик? В каждом случае, по мере сужения сферы интереса, возрастает уровень

требуемой детализации. Далее, если на определенном уровне детализации на конкретный вопрос нельзя ответить немедленно, то он откладывается на время для последующего рассмотрения. В алгоритме Варнока и его вариантах делается попытка извлечь преимущество из того факта, что большие области изображения однородны, например поверхность стола в приведенном выше примере. Такое свойство известно как когерентность, т. е. смежные области (пикселы) вдоль обеих осей x и y имеют тенденцию к однородности.

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое подокна не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения. В последнем случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом. Устранение лестничного эффекта можно реализовать, доведя процесс разбиения до размеров, меньших, чем разрешение экрана на один пиксел, и усредняя атрибуты подпикселов, чтобы определить атрибуты самих пикселов (см. разд. 2.25).

Конкретная реализация алгоритма Варнока зависит от метода разбиения окна и от деталей критерия, используемого для того, чтобы решить, является ли содержимое окна достаточно простым. В оригинальной версии алгоритма Варнока [4-10, 4-11] каждое окно разбивалось на четыре одинаковых подокна. В данном разделе обсуждаются эта версия алгоритма, а также общий вариант, позволяющий разбивать окно на полигональные области. Другая разновидность алгоритма, предложенная Вейлером и Азертоном [4-12], в которой окно тоже разбивается на полигональные подокна, обсуждается в следующем разделе. Кэтмул [4-13, 4-14] применил основную идею метода разбиения к визуализации криволинейных поверхностей. Этот метод обсуждается в разд. 4.6.

На рис. 4.32 показан результат простейшей реализации алгоритма Варнока. Здесь окно, содержимое которого слишком сложно изображать, разбито на четыре одинаковых подокна. Окно, в котором что-то есть, подразбивается далее то тех пор, пока не будет достигнут предел разрешения экрана. На рис. 4.32, a показана сцена, состоящая из двух простых многоугольников. На рис. 4.32, b показан результат после удаления невидимых линий. Заметим, что горизонтальный прямоугольник частично экранирован вертикальным. На рис. 4.32, c и d показан процесс разбиения окон на экране с разрешением 256x256. Поскольку $256 = 2^8$, требуется не более восьми шагов разбиения для достижения предела разрешения экрана. Пусть подокна рассматриваются в следующем порядке: нижнее левое, нижнее правое, верхнее левое, верхнее правое. Будем обозначать подокна цифрой и буквой, цифра — это номер шага разбиения, а буква — номер квадранта. Тогда для окна 1a подокна 2a, 4a, 4c, 5a, 5b оказываются пустыми и изображаются с фоновой интенсивностью в процессе разбиения. Первым подокном, содержимое которого не пусто на уровне пикселов, оказывается 8a. Теперь необходимо решить вопрос о том, какой алгоритм желательно применить: удаления невидимых линий или удаления невидимых поверхностей. Если желательно применить алгоритм удаления невидимых линий, то пиксел, соответствующий подокну 8a, активируется, поскольку через него проходят видимые ребра. В результате получается изображение видимых ребер многоугольников в виде последовательности точек размером с пиксел каждая (рис. 4.32, e).

Следующее рассмотрение окна, помеченного как 8d на рис. 4.32, d, лучше всего проиллюстрирует различие между реализациями алгоритмов удаления невидимых линий и поверхностей. В случае удаления невидимых линий окно 8d размером с пиксел не содержит ребер ни одного многоугольника сцены. Следовательно, оно объявляется пустым и изображается с фоновой интенсивностью или цветом. Для алгоритма удаления

невидимых поверхностей проверяется охват этого окна каждым многоугольником сцены. Если такой охват обнаружен, то среди охватывающих пиксел многоугольников выбирается ближайший к точке наблюдения на направлении наблюдения, проходящем через данный пиксел. Проверка проводится относительно центра пиксела. Затем этот пиксел изображается с интенсивностью или цветом ближайшего многоугольника. Если охватывающие многоугольники не найдены, то окно размером с пиксел пусто.

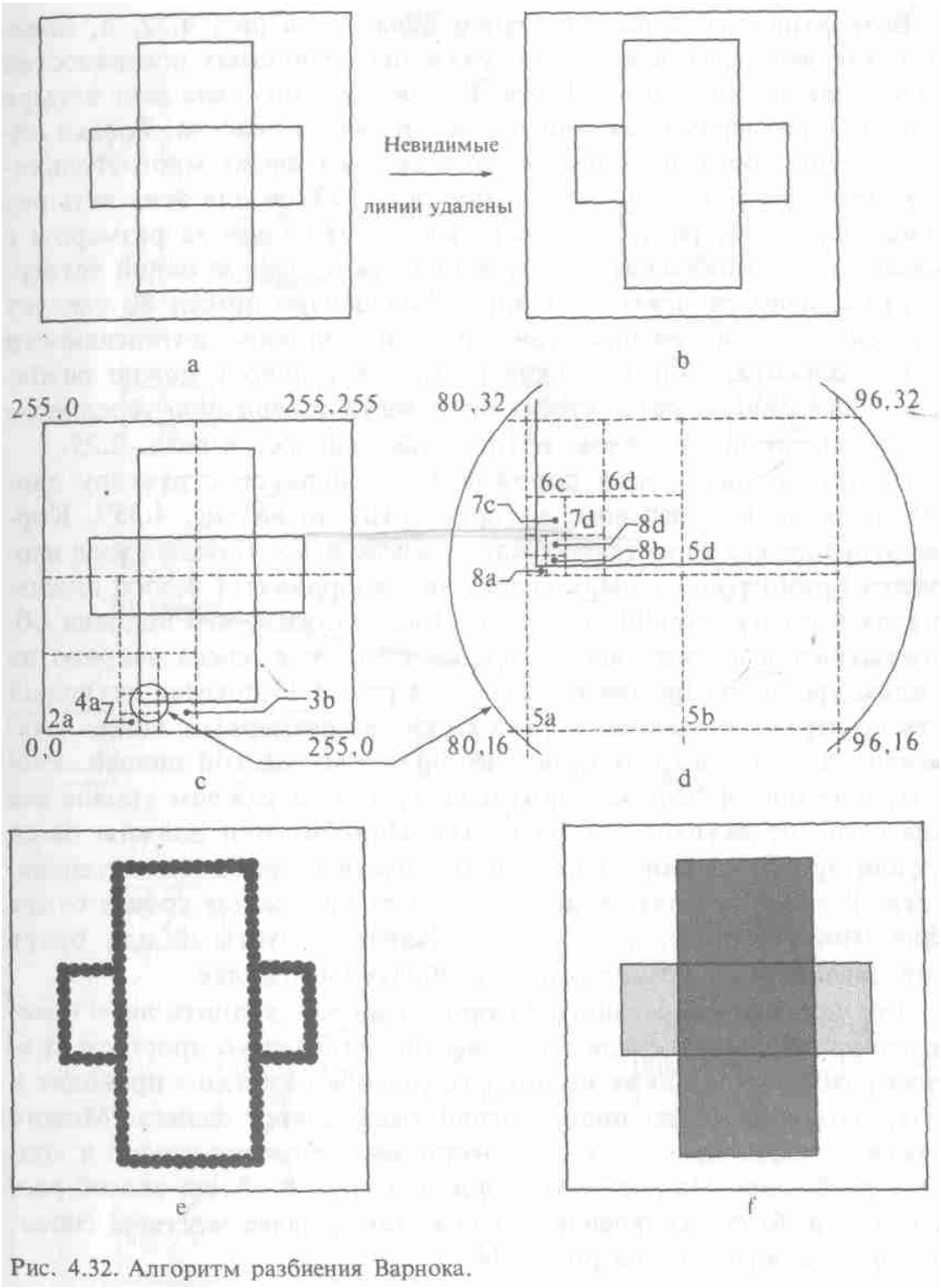

Рис. 4.32. Алгоритм разбиения Варнока.

Поэтому оно изображается с фоновым цветом или интенсивностью. Окно 8d охвачено вертикальным прямоугольником. Поэтому оно изображается с цветом или интенсивностью этого многоугольника. Соответствующий результат показан на рис. 4.32, f.

Возвратившись к рассмотрению окна 8a на рис. 4.32, d, покажем, как включить в алгоритм удаления невидимых поверхностей устранение лестничного эффекта. Разбиение этого окна дает четыре подокна с размерами, меньшими, чем размеры пиксела. Только одно из

этих подокон - правое верхнее - охвачено многоугольником. Три других пусты. Усреднение результатов для этих четырех подпикселов (см. разд. 2.25) показывает, что окно 8а размером с пиксел нужно изображать с интенсивностью, равной одной четверти интенсивности прямоугольника. Аналогично пиксел 8b следует высвечивать с интенсивностью, равной половине интенсивности прямоугольника. Конечно, окна размером с пиксел можно разбивать более одного раза, чтобы произвести взвешенное усреднение характеристик подпикселов, которое обсуждалось в разд. 2.25.

Процесс подразбиения порождает для подокон структуру данных, являющуюся деревом, которое показано на рис. 4.33. Корнем этого дерева является визуализируемое окно. Каждый узел изображен прямоугольником, содержащим координаты левого нижнего угла и длину стороны подокна. Предположим, что подокна обрабатываются в следующем порядке: *abcd*, т. е. слева направо на каждом уровне разбиения в дереве. На рис. 4.33 показан активный путь по структуре данных дерева к окну 8а размером с пиксел. Активные узлы на каждом уровне изображены толстой линией. Рас смотрение рис. 4.32 и 4.33 показывает, что на каждом уровне все окна слева от активного узла пусты. Поэтому они должны были визуализироваться ранее с фоновым значением цвета или интенсивности. Все окна справа от активного узла на каждом уровне будут обработаны позднее, т. е. будут объявлены пустыми или будут подразделены при обходе дерева в обратном порядке.

При помощи изложенного алгоритма можно удалить либо невидимые линии, либо невидимые поверхности. Однако простота критерия разбиения, а также негибкость способа разбиения приводят к тому, что количество подразбиений оказывается велико. Можно повысить эффективность этого алгоритма, усложнив способ и критерий разбиения. На рис. 4.34, а показан другой общий способ разбиения и дано его сравнение с изложенным ранее жестким способом, представленным на рис. 4.34, b.

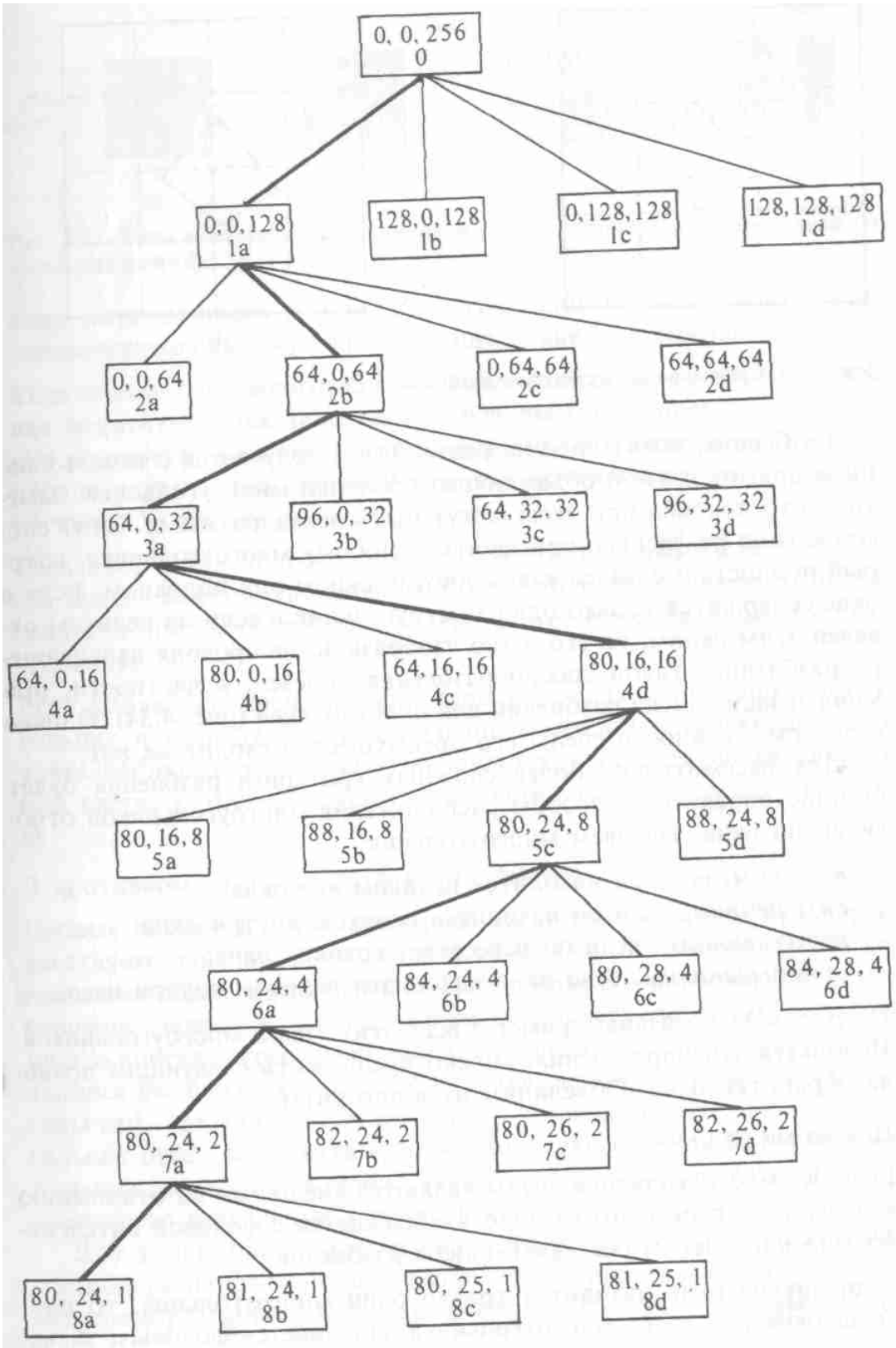

Рис. 4.33. Дерево структуры подокон.

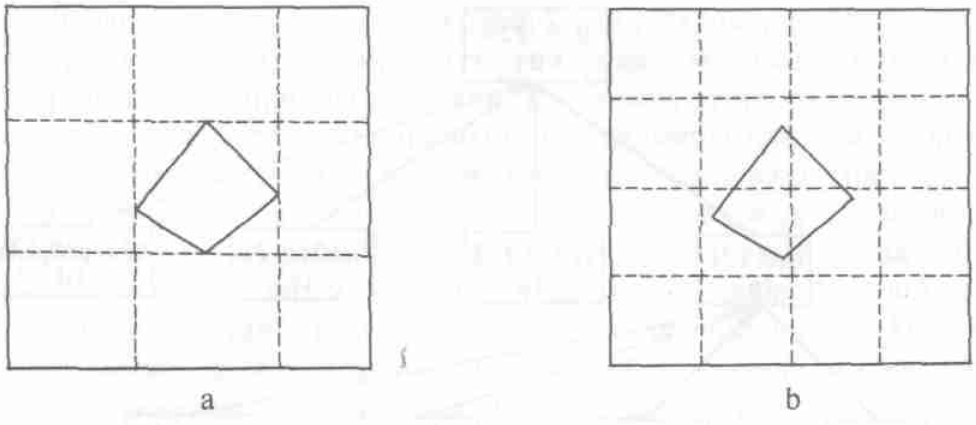

Рис. 4.34. Сравнение двух способов разбиения.

Разбиение, показанное на рис. 4.34, а, получается с использованием прямоугольной объемлющей оболочки многоугольника. Заметим, что подокна при этом могут быть неквадратными. Этот способ можно рекурсивно применить к любому многоугольнику, который полностью охвачен каким-нибудь окном или подокном. Если в окне содержится только один многоугольник и если он целиком охвачен этим окном, то его легко изобразить, не проводя дальнейшего разбиения. Такой способ разбиения полезен, в частности, при минимизации числа разбиений для простых сцен (рис. 4.34). Однако с ростом сложности сцены его преимущество сходит на нет.

Для рассмотрения более сложных критериев разбиения будет полезно определить способы расположения многоугольников относительно окна. Назовем многоугольник:

внешним, если он находится целиком вне окна;

внутренним, если он находится целиком внутри окна;

пересекающим, если он пересекает границу окна;

охватывающим, если окно находится целиком внутри него.

На рис. 4.35 показаны примеры всех этих типов многоугольников. Используя эти определения, можно предложить следующие правила обработки окна. Объединим их в алгоритм.

Для каждого окна:

Если все многоугольники сцены являются внешними по отношению к окну, то это окно пусто. Оно изображается с фоновой интенсивностью или цветом без дальнейшего разбиения.

Если внутри окна находится только один многоугольник, то площадь окна вне этого многоугольника заполняется фоновым значением интенсивности или цвета, а сам многоугольник заполняется соответствующим ему значением интенсивности или цвета

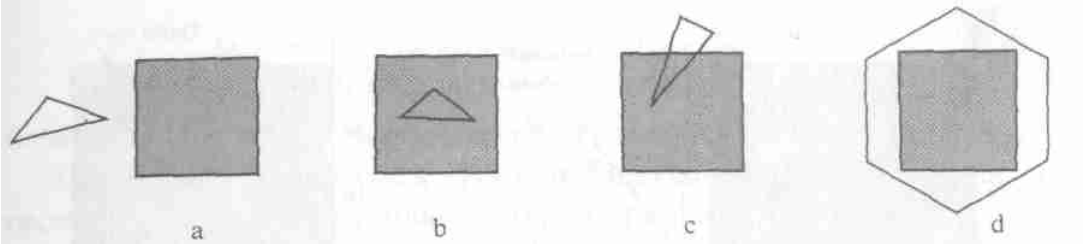

Рис. 4.35. Типы многоугольников: внешний (a), внутренний (b), пересекающий (c), охватывающий (d).

.

Если только один многоугольник пересекает окно, то площадь окна вне многоугольника заполняется фоновым значением интенсивности или цвета, а та часть пересекающего многоугольника, которая попала внутрь окна, заполняется соответствующей ему интенсив

Если окно охвачено только одним многоугольником и если в окне нет других многоугольников, то окно заполняется значением интенсивности или цвета, которое соответствует охватывающему многоугольнику.

Если найден по крайней мере один охватывающий окно многоугольник и если этот многоугольник расположен ближе других к точке наблюдения, то окно заполняется значением интенсивности или цвета, которое соответствует охватывающему многоугольнику.

В противном случае проводится разбиение окна.

Первые четыре из этих критериев касаются ситуаций, в которых участвуют один прямоугольник и окно. Они используются для сокращения числа подразбиений. Пятое правило является ключом к решению задачи удаления невидимых поверхностей. В нем речь идет о поиске такого охватывающего многоугольника, который находился бы ближе к точке наблюдения, чем все остальные многоугольники, связанные с этим окном. Очевидно, что этот многоугольник будет закрывать или экранировать все остальные многоугольники, связанные с окном. Поэтому в сцене будут фигурировать именно его визуальные характеристики.

Для реализации этих правил требуются методы определения способа расположения многоугольника относительно окна. В случае прямоугольных окон можно воспользоваться минимаксными

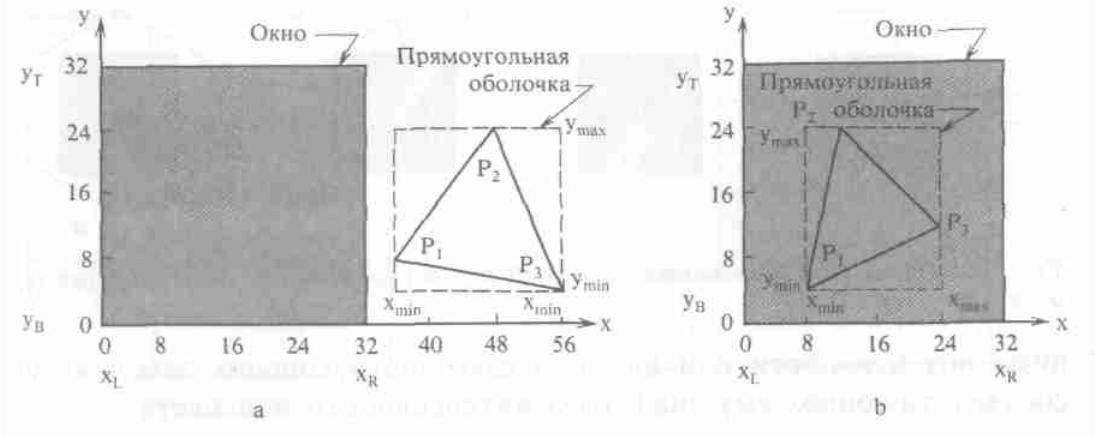

Рис. 4.36. Проверка с помощью прямоугольной оболочки для внешнего и внутреннего многоугольников.

или габаритными, с объемлющей прямоугольной оболочкой, тестами для определения, является ли многоугольник внешним по отношению к окну (см. разд. 2.13 и 3.1). В частности, если $x_Л$, $x_П$, $y_Н$, $y_В$ определяют четыре ребра окна, а x_{min} , x_{max} , y_{min} , y_{max} - ребра прямоугольной объемлющей оболочки многоугольника, то этот многоугольник будет внешним по отношению к окну, если выполняется любое из следующих условий:

$x_{min} > x_П$, $x_{max} < x_Л$,$y_{min} > y_В$,$y_{max} < y_Н$

как показано на рис. 4.36, а. Кроме того, многоугольник является внутренним по отношению к окну, если его оболочка содержится внутри этого окна, т. е. если

$x_{min} >= x_Л$ и $x_{max} <= x_П$ и $y_{min} >= y_Н$ и $y_{max} < y_В$

как показано на рис. 4.36,b.

Пример 4.14. Внутренний и внешний многоугольники

Рассмотрим квадратное окно, заданное значениями $x_Л$, $x_П$, $y_Н$, $y_В$, равными соответственно 0, 32, 0, 32. На рис. 4.36, а показаны два многоугольника, для которых нужно найти способ их расположения относительно окна. Первый из этих многоугольников задан вершинами P_1(36,8),P_2(48,24) и P_3(56,4), а второй - вершинами P_1 (8, 4), P_2 (12, 24) и P_3 (24, 12).

Прямоугольная оболочка для первого многоугольника определяется величинами x_{min} , x_{max} , y_{min} , y_{max} равными соответственно 36, 56, 4, 24. Поскольку (x_{min}=36)>($x_П$=32), этот многоугольник является внешним по отношению к окну.

Аналогично прямоугольная оболочка для второго многоугольника определяется x_{min} , x_{max} , y_{min} , y_{max} равными 8, 24, 4, 24, как показано на рис. 4.36, b. Здесь выполняются условия (x_{min} = 8) > ($x_Л$ = 0) и (x_{max} = 24) < ($x_П$ = 32) и (y_{min} = 4) > ($y_Н$ = 0) и (y_{max} = 24) < ($y_В$ = 32) Следовательно, этот многоугольник является внутренним по отношению к окну.

Можно воспользоваться простой подстановкой для проверки пересечения окна многоугольником. Координаты вершин окна подставляются в пробную функцию, заданную уравнением прямой, несущей ребро многоугольника (см. разд. 3.16 и пример 3.23). Если знак этой пробной функции не зависит от выбора вершины окна, то все его вершины лежат по одну сторону от несущей прямой и на указанной прямой нет точек пересечения. Если же эти знаки различны, то многоугольник пересекает окно. Если ни одно из ребер многоугольника не пересекает окна, то этот многоугольник либо является внешним по отношению к окну, либо охватывает его. Если уравнение прямой, проходящей через две вершины $P_1(x_1 ,y_1)$ и $P_2(x_2, y_2)$, имеет вид y = mx + b, то пробная функция (T.F. - test function) такова:

$$T.F. = y - mx- b$$

$$m = (y_2 -y_1)/(x_2 -x_1)\ x_2 -x_1 \neq 0$$

$$b = y_2 -mx_2$$

$$T.F. = x - x_1\ x_2 - x_1 = 0$$

Продемонстрируем этот метод на примере.

Пример 4.15. Пересекающие многоугольники

Возьмем квадратное окно с $x_{Л}$,,$x_{П}$,$y_{Н}$,$y_{В}$, равными соответственно 8, 32, 8, 32, и пару многоугольников, первый с вершинами $P_1(8, 4)$, P_2 (12, 24), и $P_3(40, 12)$ и второй - с вершинами $P_1(4, 4)$, P_2 (4, 36), P_3 (40, 36) и P_4 (32, 4). Они показаны на рис. 4.37. Пробная функция ребра P_1P_2 у многоугольника на рис. 4.37, а получается из следующих соотношений:

$$m = (y_2 - y_1)/(x_2 - x_1) = 20/4 = 5$$

$$b = y_1 - mx_1 = 4 - 5(8) = -36$$

$$T.F. = y - mx - b = y - 5x + 36$$

Подстановка координат каждой угловой точки окна в пробную функцию дает:

$$T.F.(8,8) = 8 - 5(8) + 36 = 4$$

$$T.F.(8, 32) = 32 - 5(8) + 36 = 28$$

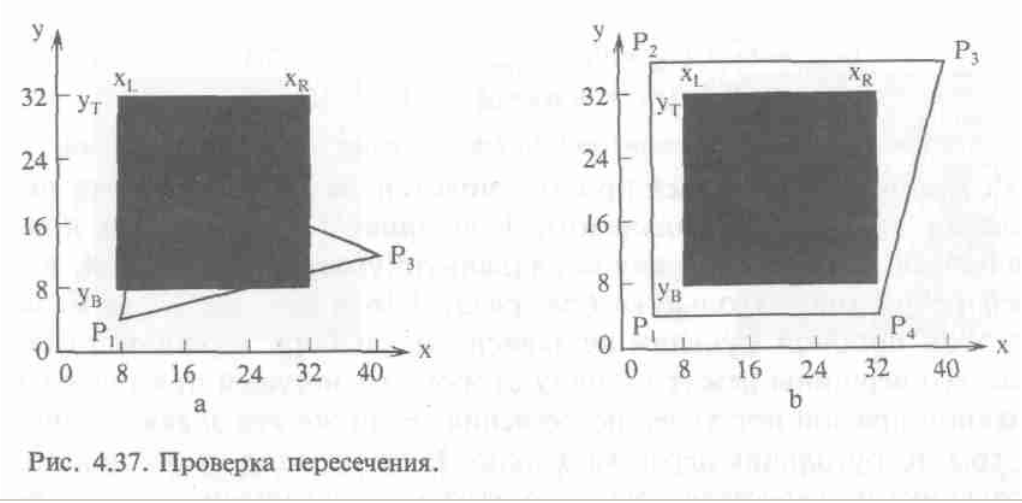

Рис. 4.37. Проверка пересечения.

Таблица 4.9.

Ребро многоугольника	Пробная функция	Координаты окна	Результат подстановки	Примечание
P_1P_2	x - 4	(8,8)	4	Нет пересечения
		(8,32)	4	
		(32,32)	28	
		(32,8)	28	
P_2P_3	y - 36	(8,8)	-28	Нет пересечения
		(8,32)	-4	
		(32,32)	-4	
		(32,8)	-28	
P_3P_4	y - 4x + 124	(8,8)	100	Нет пересечения
		(8,32)	124	
		(32,32)	28	
		(32,8)	4	
P_4P_1	y - 4	(8,8)	4	Нет пересечения
		(8,32)	28	
		(32,32)	28	

		(32,8)	4	

T.F.(32,32) = 32 - 5(32) + 36 = -92 T.F.(32,8) = 8 - 5(32) + 36 = - 116

Поскольку знаки подстановок в пробную функцию различны, это ребро многоугольника пересекает окно, что и показано на рис. 4.37, a. Значит, заданный многоугольник является пересекающим. Проверять остальные ребра этого многоугольника нет необходимости.

Результаты проверки многоугольника, изображенного на рис. 4.37, b, сведены в табл. 4.9. Ни одно из ребер этого многоугольника не пересекает окна. Следовательно, этот многоугольник либо внешний, либо охватывающий. На рис. 4.37 видно, что он охватывающий.

Простым габаритным тестом с прямоугольной оболочкой, который рассматривался выше, нельзя идентифицировать все виды внешних многоугольников. Примером может служить многоугольник, огибающий угол окна (рис. 4.38, a). Для этого нужны более сложные тесты. Особый интерес представляют тест с бесконечной прямой и тест с подсчетом угла. В обоих тестах предполагается, что внутренние и пересекающие многоугольники уже были идентифицированы. Оба теста могут быть использованы для определения внешних и охватывающих многоугольников.

В тесте с бесконечной прямой проводится луч из любой точки окна, например из угла в бесконечность. Подсчитывается число пересечений этого луча с заданным многоугольником. Если это число четное (или нуль), то многоугольник внешний; если же оно нечетное, то многоугольник охватывает окно, как показано на рис. 4.38, a. Если луч проходит через вершину многоугольника, как показано на рис. 4.38, b, то результат неопределен. Эта неопределенность устраняется, если считать касание за два пересечения (P_2 на рис. 4.38, b), а протыкание - за одно (P_4 на рис. 4.38, b) (см. также разд. 2.17). Изменение угла наклона луча тоже позволяет устранить неопределенность.

Тест с подсчетом угла проиллюстрирован на рис. 4.39. Совершим обход по ребрам многоугольника по или против часовой стрелки. При этом просуммируем углы, образованные лучами, начинающимися в любой точке окна и проходящими через начало и конец проходимого в данный момент ребра многоугольника. Как показано на рис. 4.39, удобной точкой является центр окна. Получаемая сумма углов интерпретируется следующим образом:

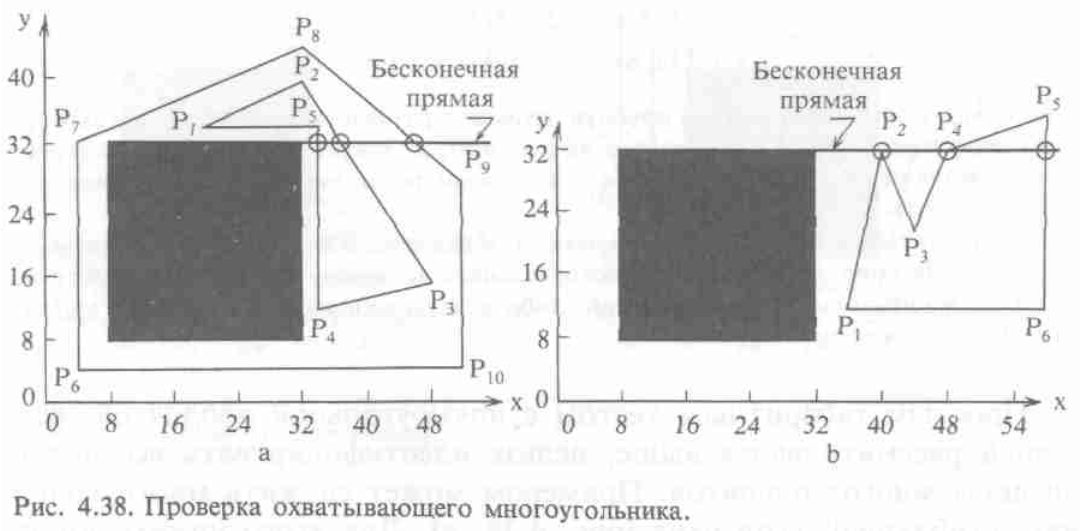

Рис. 4.38. Проверка охватывающего многоугольника.

Сумма = 0, многоугольник, внешний по отношению к окну.
Сумма = ± 360n, многоугольник охватывает окно n раз

(Простой многоугольник ,не имеющий самопересечений, может охватить точку только один раз).

Практическое вычисление этой суммы значительно упрощается, если учесть, что точность вычисления каждого угла не обязана быть высокой. Фактически нужная точность получается, если считать только целые октанты (приращения по 45°), покрытые отдельными углами, как показано на рис. 4.40. Техника здесь напоминает ту, что использовалась для кодирования концов отрезка при

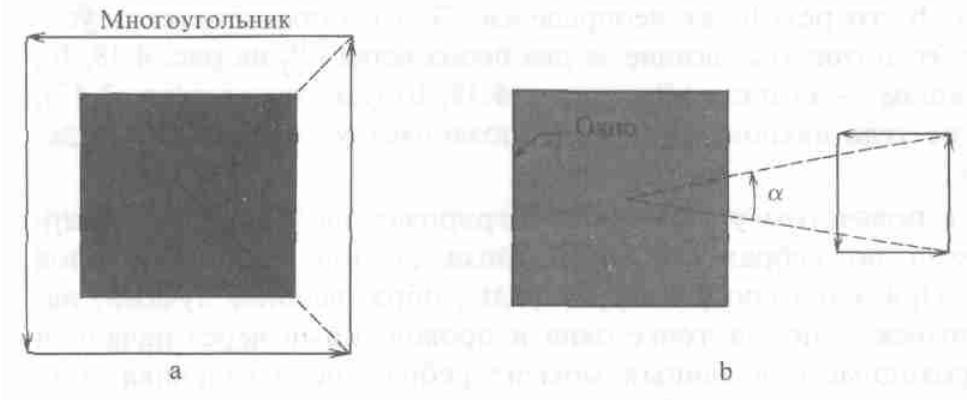

Рис. 4.39. Проверка с подсчетом угла.

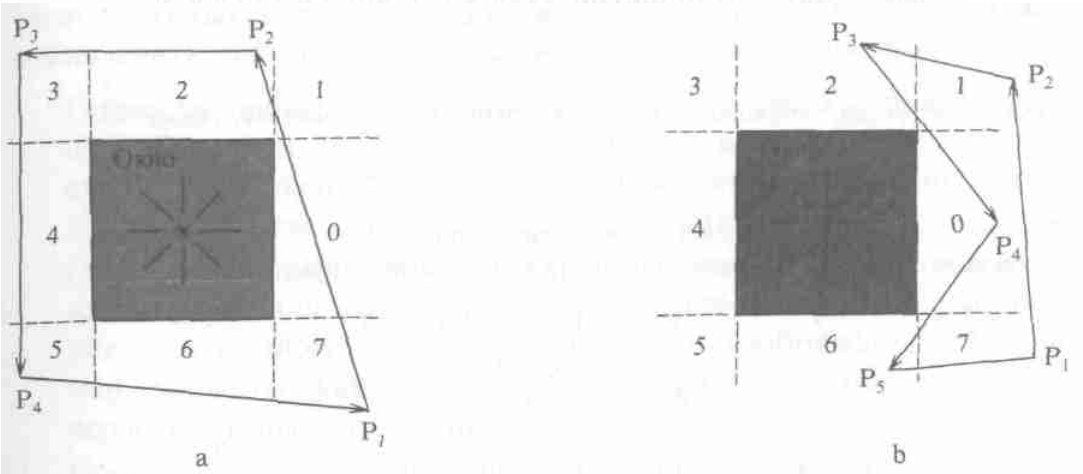

Рис. 4.40. Угловой тест для охватывающего и внешнего многоугольников.

отсечении (см. разд. 3.1). Пронумеруем октанты числами от 0 до 7 против часовой стрелки. Число целых октантов, покрытых углом, равно разности между номерами октантов, в которых лежат концы его ребер. Предлагается следующий алгоритм:

Δa = (номер октанта, в котором лежит второй конец ребра) - (номер октанта, в котором лежит первый конец ребра)

if $\Delta a > 4$ **then** $\Delta a = \Delta a - 8$

if $\Delta a < -4$ **then** $\Delta a = \Delta a + 8$

if $\Delta a = 0$ **then** ребро многоугольника расщепляется ребром окна, и процесс повторяется с парой полученных отрезков.

Суммируя вклады от отдельных ребер, получаем:

$\Sigma\Delta a =$ { 0, многоугольник, внешний по отношению к окну,

± 8 n, многоугольник охватывает окно.

Пример 4.16. Угловой тест для охватывающего и внешнего многоугольников

Рассмотрим окно и многоугольники, показанные на рис. 4.40. Для многоугольника с рис. 4.40, а получаем для ребра P_1P_2 :

$$\Delta a_{12} = 2 - 7 = -5 < -4$$
$$= -5 + 8 = 3$$

Аналогично, для других ребер многоугольника имеем:

$$\Delta a_{23} = 3 - 2 = 1$$
$$\Delta a_{34} = 5 - 3 = 2$$
$$\Delta a_{41} = 7 - 5 = 2$$

Сумма всех углов, опирающихся на ребра этого многоугольника, равна:

$$\Sigma\Delta a = 3 + 1 + 2 + 2 = 8$$

Поэтому данный многоугольник охватывает окно.

Для многоугольника, показанного на рис. 4.40, b, имеем:

$$\Delta a_{12} = 1 - 7 = -6 < -4$$
$$= -6 + 8 = 2$$
$$\Delta a_{23} = 2 - 1 = 1$$
$$\Delta a_{34} = 0 - 2 = -2$$
$$\Delta a_{45} = 6 - 0 = 6 > 4$$
$$= 6 - 8 = -2$$
$$\Delta a_{51} = 7 - 6 = 1$$
$$\Sigma\Delta a = 2 + 1 - 2 - 2 + 1 = 0$$

Поэтому данный многоугольник является внешним по отношению к окну.

Иерархическое использование методов, основанных на изложенных вычислительных приемах, дает дополнительный выигрыш. Если реализуется только простейший алгоритм Варнока, то нет смысла определять внутренние или пересекающие многоугольники. Разбиения будут постоянно превращать такие многоугольники во внешние или охватывающие. Все нестандартные ситуации разрешаются на уровне пикселов. В этом простом алгоритме для определения пустых окон достаточно пользоваться только тестом с прямоугольной объемлющей оболочкой. Если этот простой тест дает отрицательный результат, то алгоритм разбивает окно вплоть до достижения уровня пикселов. Поскольку внешний многоугольник в той форме, которая показана на рис. 4.40, b, может встретиться даже на уровне пикселов, то необходимо применить более мощный алгоритм для определения пустоты окна или охвата его одним или более многоугольниками.

Более сложный алгоритм, обсуждавшийся выше, делает попытку определить способ расположения многоугольника относительно окна больших размеров, чтобы избежать его подразбиения. Такие тесты требуют больших вычислительных затрат. Следовательно, существует зависимость между затратами, связанными с процессом разбиения окон, и затратами, связанными с ранним определением окон, готовых к визуализации. Более сложный алгоритм должен реализовать тестирование каждого окна в следующем порядке.

Провести проверку при помощи простого теста с прямоугольной оболочкой для определения как можно большего числа пустых окон и окон, содержащих единственный внутренний многоугольник. Такие окна сразу же изображаются.

Провести проверку при помощи простого теста на пересечение для определения окон, пересекающих единственный многоугольник. Этот многоугольник отсекается и изображается. Например, многоугольник на рис. 4.34, b был бы изображен после первого же шага разбиения.

Провести проверку при помощи более сложных тестов для внешних и охватывающих многоугольников, чтобы определить дополнительные пустые окна и окна, охваченные единственным многоугольником. Такие окна сразу же изображаются.

После реализации этих шагов проводится разбиение окна или делается попытка обнаружить такой охватывающий многоугольник, который был бы ближе к точке наблюдения, чем любой другой многоугольник. Если проводится разбиение, то решение последнего вопроса откладывается до достижения уровня пикселов. В любом случае требуется проводить тест глубины.

Тест глубины проводится путем сравнения глубин (координат z) плоскостей, несущих многоугольники, в угловых точках окна. Если глубина охватывающего многоугольника больше, чем глубины всех остальных многоугольников в углах окна, то охватывающий многоугольник экранирует все остальные многоугольники в этом окне. Следовательно, данное окно можно визуализировать с тем значением интенсивности или цвета, которое соответствует охватывающему многоугольнику. Заметим, что приведенное условие по глубине является достаточным, но не необходимым для того, чтобы охватывающий многоугольник экранировал все остальные многоугольники в окне. На рис. 4.41 показано, что проведение теста глубины в углах окна для несущих грани плоскостей может привести к неудаче при определении охватывающего окно многоугольника, который экранирует все остальные многоугольники в этом окне.

В частности, если плоскость, несущая многоугольник, экранируется охватывающим многоугольником в углах окна, то сам многоугольник тоже экранируется последним в углах окна (как показано на рис. 4.41). Если же несущая плоскость не экранируется охватывающим окно многоугольником, то не очевидно, будет или нет экранирован многоугольник, лежащий в этой плоскости (случай b на рис. 4.41).

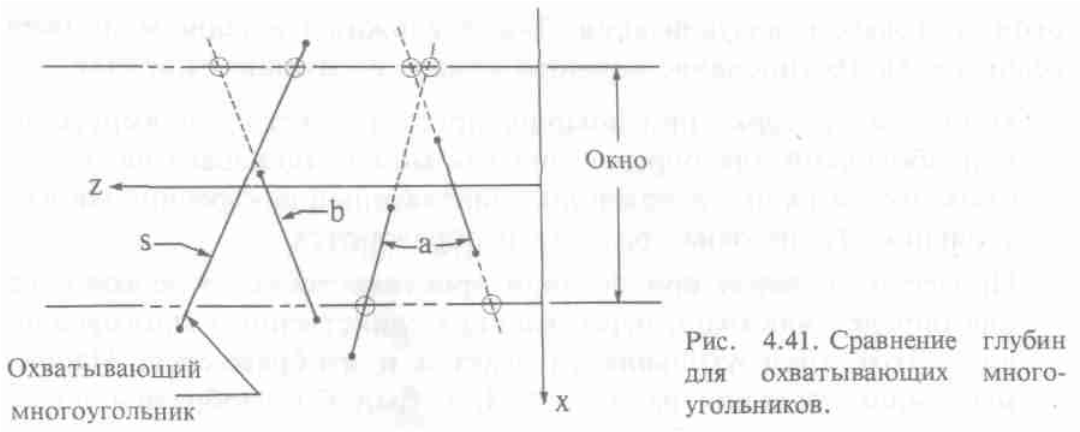

Рис. 4.41. Сравнение глубин для охватывающих многоугольников.

Этот конфликт разрешается путем разбиения окна.

Глубину несущей плоскости в углу окна можно вычислить с помощью ее уравнения (см. разд. 4.3 и пример 4.3). Например, если уравнение этой плоскости имеет вид

$$ax + by + cz + d = 0$$

а координаты угловой точки окна равны (x_w , y_w), то значение

$$z = -(d + ax_w + by_w)/c \quad c \neq 0$$

равно искомой глубине.

Всюду выше предполагалось, что все многоугольники следует сравнивать с каждым окном. Для сложных сцен это весьма неэффективно. Эффективность можно повысить, проведя сортировку по приоритету глубины (сортировку по z). Сортировка проводится по значению z координаты той вершины многоугольника, которая ближе других к точке наблюдения. В правой системе координат многоугольник с максимальным значением такой координаты z будет ближайшим к точке наблюдения. В списке отсортированных многоугольников он появится первым.

При обработке каждого окна алгоритм ищет охватывающие его многоугольники. После обнаружения такого многоугольника запоминается величина z у его самой удаленной от наблюдателя вершины z_{smin}. При рассмотрении каждого очередного многоугольника в списке сравнивается координата z его ближайшей к наблюдателю вершины - z_{pmax} с z_{smin} . Если $z_{pmax} < z_{smin}$, то очевидно, что очередной многоугольник экранируется охватывающим и не должен больше учитываться. Из рис. 4.41 видно, что это условие является достаточным, но не необходимым; например, многоугольники, помеченные буквой a на рис. 4.41, больше можно не учитывать, но многоугольник, помеченный буквой b, учитывать нужно.

Длину списка многоугольников, обрабатываемых для каждого окна, можно сократить, если воспользоваться информацией, полученной этим алгоритмом ранее. В частности, если многоугольник охватывает окно, то очевидно, что он охватывает и все подокна этого окна и его можно больше не тестировать для них. Кроме того, если многоугольник является внешним по отношению к окну, то он будет внешним и для всех его подокон, и его можно не рассматривать при обработке этих подокон. Учитывать далее следует только пересекающие и внутренние многоугольники.

Чтобы воспользоваться этой информацией, применяют три списка: для охватывающих, для внешних и для пересекающих и внутренних многоугольников [4-11]. По ходу процесса разбиения многоугольники включаются в соответствующий список или удаляются из него. Запоминается также и уровень (шаг разбиения), на котором многоугольник впервые попадает в тот или иной список. Эта информация используется при обходе дерева, изображенного на рис. 4.33, в обратном порядке. На каждом шаге разбиения первым обрабатывается список охватывающих многоугольников для того, чтобы найти ближайший к наблюдателю многоугольник такого типа. Затем обрабатывается список пересекающих и внутренних многоугольников, чтобы проверить, не экранируются ли все они охватывающим многоугольником. Список внешних многоугольников игнорируется.

Итак, были обсуждены основные идеи и методы возможных улучшений алгоритма Варнока. Важно подчеркнуть, что нет единого алгоритма Варнока. Конкретные реализации этого алгоритма различаются в деталях. Запись наиболее фундаментальной версии этого алгоритма на псевдоязыке дана ниже. Если размер окна больше разрешающей способности дисплея и если оно еще содержит нечто, представляющее интерес, то алгоритм всегда будет разбивать окно. Чтобы идентифицировать внешние многоугольники для окон, размер которых больше размера пикселов, проводится простой габаритный тест с прямоугольной оболочкой. А для окон размером с пиксел выполняется более сложный тест, в котором видимый многоугольник определяется путем сравнения

координат z всех многоугольников в центре пиксела. Однако здесь не используется сортировка по приоритету вдоль оси z, а также выигрыш, даваемый ранее накопленной информацией об относительном расположении многоугольников и окон. Алгоритм использует стек, максимальная длина которого равна:

3*(разрешение экрана в битах - 1) + 5

Этот простой алгоритм предназначен для демонстрации основных идей без подробностей реализации структур данных. Для выпуклых многогранников до начала работы данного алгоритма проводится устранение нелицевых граней (см. разд. 4.2). Блок-схема показана на рис. 4.42.

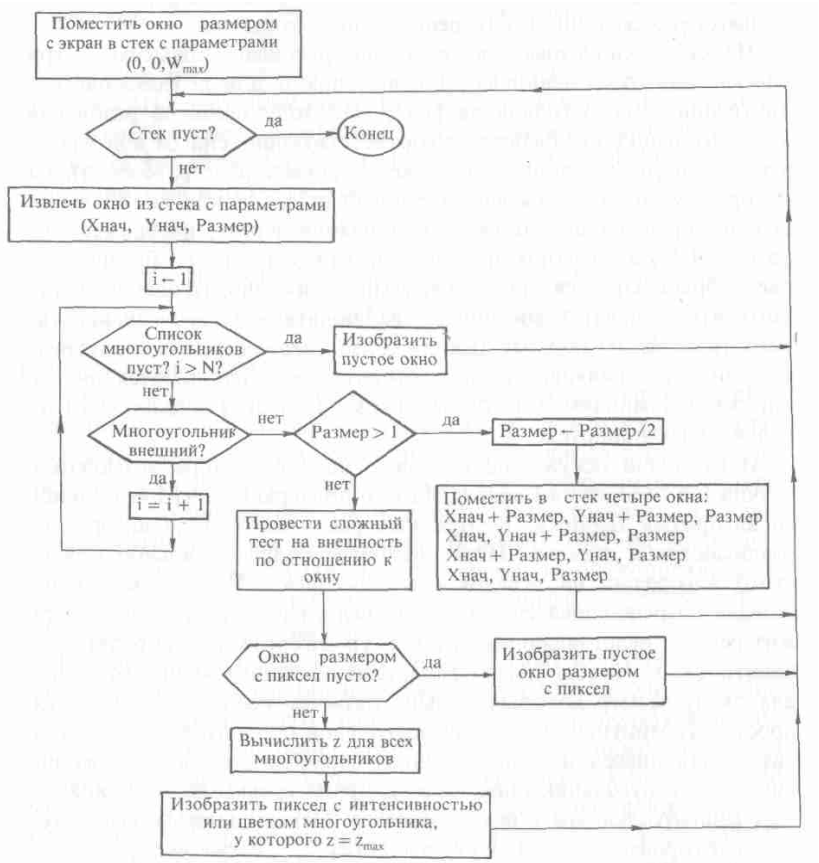

Рис. 4.42. Блок-схема простого алгоритма Варнока.

Пример 4.17. Алгоритм Варнока

Рассмотрим три многоугольника:

1: (10, 3, 20), (20, 28, 20), (22, 28, 20), (22, 3, 20)
2: (5, 12, 10), (5, 20, 10), (27, 20, !0), (27, 12, 20)
3: (15, 15, 25), (25, 25, 5), (30, 10,5)

которые нужно изобразить на экране с разрешением 32x32 пиксела, используя простой алгоритм Варнока, описанный выше. Два первых многоугольника являются прямоугольниками, перпендикулярными оси z соответственно при z = 20 и z = 10. Третий - это треугольник, протыкающий оба прямоугольника, как показано на рис. 4.43, а.

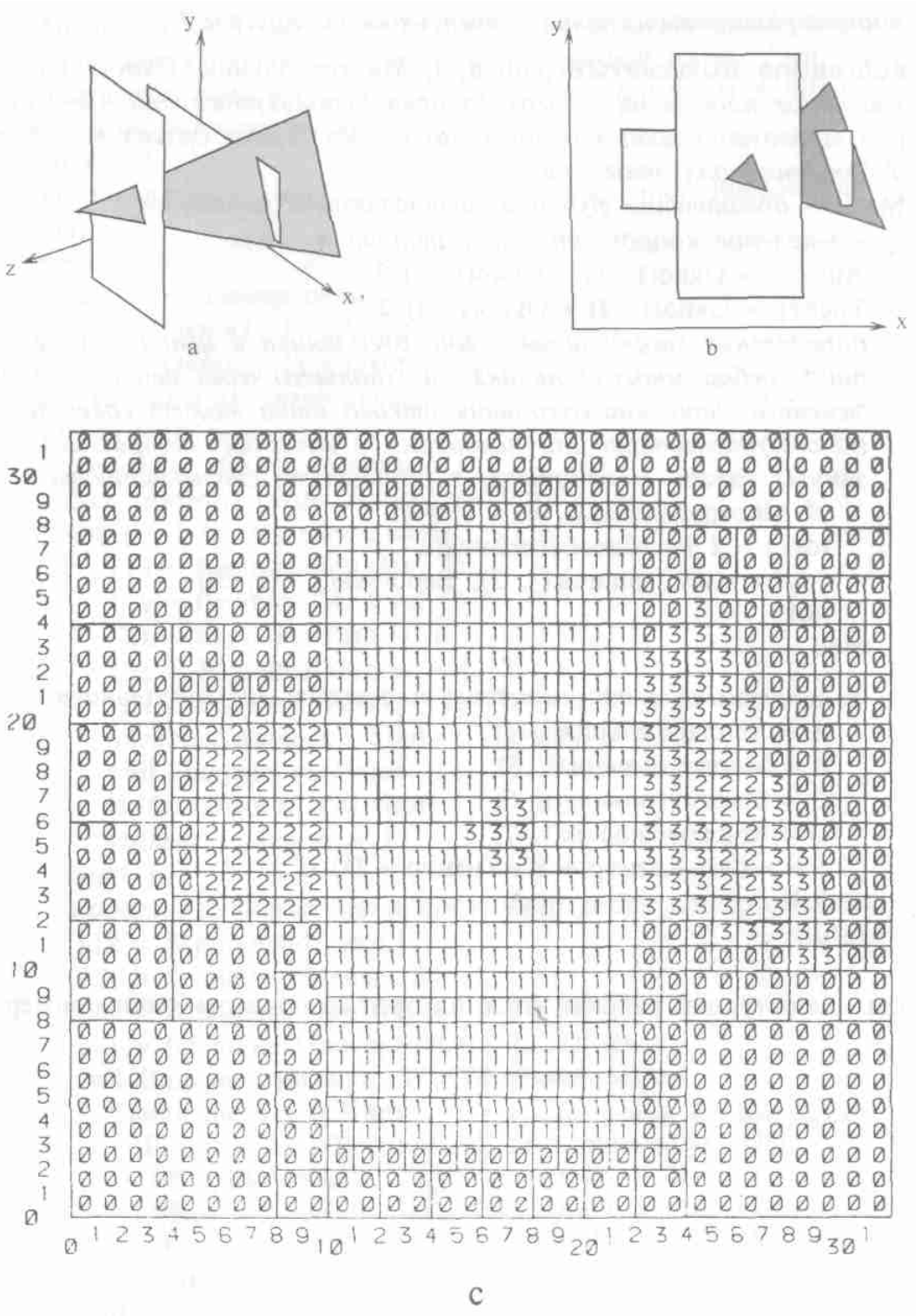

Рис. 4.43. Пример многоугольников для простого алгоритма Варнока.

На рис. 4.43, b показан вид на эту сцену из точки, лежащей в бесконечности на положительной полуоси z; невидимые линии удалены с рисунка. На рис. 4.43, с показано содержимое буфера кадра после завершения работы алгоритма. Цифры в ячейках соответствуют номерам многоугольников. Алгоритм работает, начиная с левого нижнего угла, направо и вверх. Крупные ячейки показывают габариты окон, полученных в результате разбиения экрана на каждом шаге алгоритма. Обратим, например, внимание на большое пустое окно (8x8) в левом нижнем углу экрана. Это окно изображено без дальнейшего подразбиения. На рисунках показано, что треугольник частично экранирован вторым прямоугольником, затем протыкает его, становится частично видимым, потом экранируется первым прямоугольником, и, наконец, протыкает и его, так что делается видимой вершина этого треугольника