7.5. Метод сканирующей сетки Энкарнако

Этот алгоритм можно применять к произвольным **криволинейным поверхностям**, изображаемым в виде сетки uv-линии. Куски криволинейной поверхности определяются только четырьмя вершинами; ребра являются отрезками прямых. Криволинейная поверхность внутри куска аппроксимируется четырьмя плоскостями, образующимися при проведении диагоналей между противоположными вершинами куска. Алгоритм оперирует в **картинной плоскости** и использует тесты, сопоставляющие точку с поверхностью. Трехмерные поверхности, описанные в объектном пространстве, сначала отображаются на картинную плоскость с помощью операции **ортогонального проецирования.**

На проекцию поверхностей в картинной плоскости накладывается прямоугольная двумерная сетка - сканирующая сетка. Для каждой ячейки сканирующей сетки составляется список кусков поверхностей, проекции которых могут иметь общие точки с данной ячейкой. Такая предварительная сортировка кусков по ячейкам сканирующей сетки позволяет ограничиться при определении видимости тестовой точки только теми кусками поверхности, которые лежат в той же ячейке, что и тестовая точка. Если сканирующая сетка имеет слишком крупные ячейки, то с каждой ее ячейкой будет связано много кусков и выполнение тестов видимости займет много времени. Если используется слишком частая сетка, то это либо приведет к излишнему расходу памяти, либо увеличит время начальной сортировки и тем самым ликвидирует выигрыш времени на тестах видимости.

Для определения ячеек сетки, которые могут перекрыться заданным куском поверхности, используется минимаксный тест. Быстродействие и простота минимаксного теста с избытком компенсируют дополнительные затраты времени, обусловливаемые применением тестов принадлежности к тестовым точкам из ячейки c и к куску поверхности P.

Пусть заданы два выпуклых многоугольника в плоскости изображения. Каждый многоугольник определяется множеством своих вершин, т. е. обозначим многоугольники как

$$F_1 = P_{1,1}, P_{1,2}, \text{K}\ , P_{1,m} \text{ и } F_2 = P_{2,1}, P_{2,2}, \text{K}\ , P_{2,m} \text{ где } P_{i,j} = (x_{i,j}, y_{i,j}).$$

Многоугольники F_1 и F_2 не перекрываются друг с другом, если выполняется следующее минимаксное условие:

$$j \in [1,m] \quad i \in [1,n]: \max(x_{1,j}) < \min(x_{2,k}) \textit{ или } \max(x_{2,k}) < \min(x_{1,j}) \textit{ или}$$
$$\max(y_{1,j}) < \min(y_{2,k}) \textit{ или } \max(y_{2,k}) < \min(y_{1,j}).$$

После завершения предварительной сортировки и формирования списков кусков, связанных с ячейками сетки, выполняется основной алгоритм, определяющий видимость u- и v-линий. Для этого отрезок u- или v-линии (ребро куска поверхности) разбивается на цепочку тестовых точек, равномерно распределенных между вершинами куска. Каждая точка сопоставляется с кусками поверхности, лежащими в той же ячейке сетки, что и тестовая точка. Если ни один из кусков не покрывает тестовую точку, то она видима; если же обнаруживается кусок, покрывающий ее, то она невидима. Если видимы все тестовые точки, лежащие на ребре куска, то рисуется все ребро. Если же некоторые точки невидимы, то рисуются только те части ребра, которые лежат между видимыми точками .

Видимость каждой тестовой точки вычисляется в несколько этапов. Из списка, который связан с ячейкой сетки, содержащей тестовую точку, выбирается кусок поверхности. Если это как раз тот кусок, из которого взята тестовая точка, то его рассмотрение прекращается. В противном случае выполняется тест принадлежности, определяющий, может ли данный кусок покрыть точку.

В геометрии имеются процедуры, позволяющие выполнить эту проверку. Это, например, вычисление суммы углов. Пусть $F = (p_1, p_2, \text{K}\ , p_n)$ - многоугольник в картинной плоскости с вершинами $p_i = (x_i, y_i), i = 1, \text{K}\ , n, \quad p_n = p_1, \quad p_t$ - точка, для которой необходимо определить, принадлежит ли она F; $p_t p_i$ - отрезок, соединяющий p_t и p_i; p_t со-

единена такими отрезками со всеми вершинами F; α_i - угол между $p_t p_i$ и $p_t p_{i+1}$. Тогда точка p_t находится вне F, если $\sum_i \alpha_i = 0$ и внутри F, если $\sum_i \alpha_i = 2\pi$.

Если данный кусок может покрыть точку, то необходимо с помощью теста глубины проверить, закрыта ли точка куском.

В тесте точка / поверхность строится уравнение плоскости, содержащей грань, и в него подставляются xy-координаты проверяемой точки.

Каждая плоскость многогранника описывается уравнением

$$aX + bY + cZ = 0$$

где *X, Y* и *Z* - мировые координаты. В матричной форме это уравнение имеет вид

$$[X \quad Y \quad Z \quad 1][P]^T = 0$$

где $P^T = [a \quad b \quad c \quad d]$ - транспонированный вектор коэффициентов плоскостей.

Любая точка на плоскости однозначно определяется двумя координатами, а точка в пространстве в однородных координатах описывается вектором

$$[S] = [X \quad Y \quad Z \quad 1]$$

Если точка лежит на плоскости, то скалярное произведение

$$[S][P]^T = 0$$

Если точка не лежит на плоскости, то по знаку этого скалярного произведения можно судить, по какую сторону от плоскости расположена точка

Процедура усложняется, если допускается существование проникающих граней или циклического перекрытия . В этом случае результат теста зависит от местонахождения проверяемой точки. Для разрешения такой неоднозначности необходимо принять специальные меры, например разделить грань F_1 вдоль штриховой линии на две части и выполнить тест глубины отдельно для каждой части.

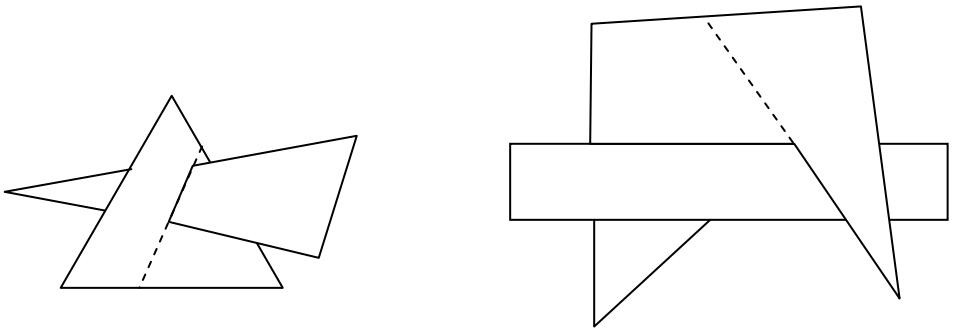

Рис. 7.5.1. Проникающие грани и циклическое перекрытие

Прямая, проходящая через точки $P_1 = (x_1, y_1, z_1)$ и $P_2 = (x_2, y_2, z_2)$ может быть представлена уравнением

$$\frac{x - x_1}{\cos\alpha} = \frac{y - y_1}{\cos\beta} = \frac{z - z_1}{\cos\gamma},$$

где :

$$\cos\alpha = \frac{x_2 - x_1}{d}, \quad \cos\beta = \frac{y_2 - y_1}{d}, \quad \cos\gamma = \frac{z_2 - z_1}{d},$$

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \ .$$

Или для двух прямых $\begin{cases} y = k_1 x + a_1 \ , z = h_1 x + b_1, \\ y = k_2 x + a_2 \ , z = h_2 x + b_2. \end{cases}$

Точка пересечения имеет координаты

$$x_0 = \frac{a_2 - a_1}{k_1 - k_2} = \frac{b_2 - b_1}{h_1 - h_2}, \quad y_0 = \frac{k_1 a_2 - k_2 a_1}{k_1 - k_2}, \quad z_0 = \frac{h_1 a_2 - h_2 a_1}{h_1 - h_2}.$$

Поскольку куски поверхности криволинейны, можно разбить кусок на два треугольника, проведя диагональ, и в тесте глубины использовать плоскости, определяемые этими треугольниками.

Уравнение плоскости по трём точкам

$P_1(x_1, y_1, z_1),\ P_2(x_2, y_2, z_2),\ P_3(x_3, y_3, z_3)$ имеет вид: $\begin{vmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{vmatrix} = 0$

Однако возникает вопрос, какую именно диагональ следует предпочесть. В алгоритме используются обе диагонали - рассматриваются треугольники, образуемые каждой диагональю

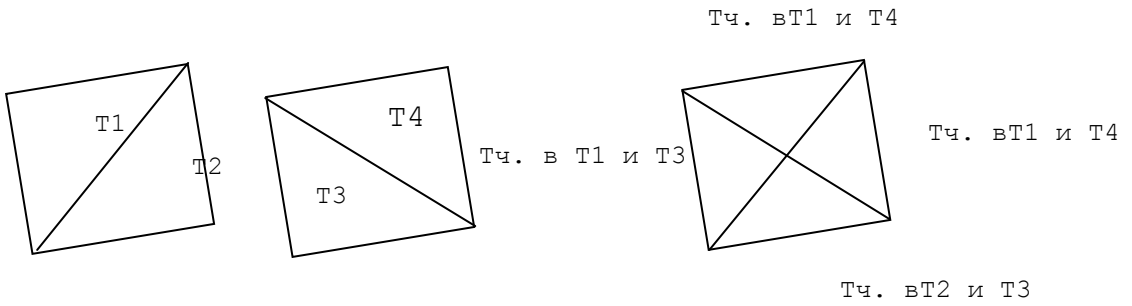

Рис. 7.5.2. Аппроксимация плоскостями (первые две, вторые две и все четыре плоскости.

Тестовая точка должна лежать только в двух из этих четырех треугольников. Выполняются два теста принадлежности, чтобы определить треугольники, содержащие точку, и затем выполняются два теста глубины для определения ее видимости. Точка считается невидимой только в том случае, когда она закрывается обоими треугольниками. Если она закрывается только одним треугольником, ошибка аппроксимации криволинейной поверхности плоскостями оказывается больше, чем расстояние между точкой и двумя плоскостями, содержащими точку, и решения о видимости или невидимости становятся равновероятными.

В исходном алгоритме список кусков, связанный с некоторой ячейкой сканирующей сетки, просматривался сначала для каждой последующей тестовой точки вдоль $u-$ или v-линии. Такая стратегия пригодна при проверке последовательности видимых точек,

поскольку для определения видимости необходимо проверить все куски. Однако для последовательности невидимых тестовых точек эта стратегия неэффективна, поскольку для определения невидимости точек достаточно иметь в среднем лишь половину списка кусков. Алгоритм может быть модифицирован следующим образом: когда обнаруживается кусок, закрывающий тестовую точку, он переносится в голову списка, поскольку вероятность того, что этот же кусок закроет несколько соседних точек вдоль $u-$ или v-линии, весьма высока. Такое простое изменение приводит к уменьшению времени работы алгоритма примерно на порядок.

Может оказаться, что некоторая ячейка сетки содержит слишком большое число кусков и разумнее разбить ее на более мелкие ячейки. Осуществляется это следующим образом: ячейка сетки, в списке которой содержится более восьми кусков, подразделяется на четыре подъячейки и для конкретной подъячейки, содержащей проверяемую точку, формируется временный список кусков . Если и в новом списке более восьми кусков, подъячейка делится еще раз и составляется еще один список. Этот процесс повторяется, если в $1/16$ ячейки находится более 16 кусков. Пороги 8 и 16 оказались оптимальными для одной конкретной реализации алгоритма. Оптимальность определяется в основном отношением времени выполнения минимаксной процедуры и процедуры принадлежности. Общая цель, преследуемая таким локальным делением ячеек сканирующей сетки, состоит в том, чтобы чаще использовать быстрые минимаксные тесты и за счет этого уменьшить число более медленных тестов принадлежности.

7.6. Алгоритмы построчного сканирования

Эти алгоритмы разработаны Уайли, Ромни, Эвансом и Эрдалом [1], Букнайтом [2] . Они функционируют в **пространстве изображения**, причем образ в них генерируется построчно. Алгоритм сводит 3-х мерную задачу удаления невидимых линий к 2-х мерной. Пересечение сканирующей плоскости и 3-х мерной сцены определяет окно размером в

одну строку. Задача решается в пределах этого окна. Ниже описан общий подход, который с некоторыми вариациями используется в названных выше алгоритмах. Этот подход является расширением алгоритма преобразования многоугольника в растровую форму. Различие заключается в том, что в этом случае мы имеем дело не с одним многоугольником, а сразу со всеми многоугольниками, определяющими объект.

На первом шаге создается таблица ребер (ТР), содержащая все негоризонтальные ребра многоугольников. Элементы ТР отсортированы по группам на основе меньшей y - координаты каждого ребра, а внутри групп — в зависимости от величины тангенса угла наклона. Эти элементы содержат:

1) x - координату крайней точки ребра с меньшей y-координатой;

2) y - координату другой крайней точки ребра;

3) приращение Δx координаты x, используемое для перехода от каждой сканирующей строки к следующей (Δx обратно тангенсу угла наклона ребра);

4) идентификатор многоугольника, указывающий, какому многоугольнику принадлежит данное ребро.

На втором шаге создается таблица многоугольников (ТМ), в которой содержится по крайней мере следующая информация о каждом многоугольнике:

1) коэффициенты уравнения плоскости (a, b, c, d);

Уравнение плоскости описывается формулой $aX + bY + cZ + d = 0$, где коэффициент d вычисляют с использованием произвольной точки на плоскости. Например, если эта точка имеет координаты (x_1, y_1, z_1), то

$$d = -(ax_1 + by_1 + cz_1)$$

Для определения коэффициентов уравнений плоскостей многогранников синтезируемого объекта поступают следующим образом. Определяют нормали к каждой вершине многоугольника с помощью векторного произведения прилежащих ребер, а затем усреднении результатов. Для определения нормали к плоскости вычисляют нормали к каждой

вершине многоугольника с помощью векторного произведения прилежащих ребер. Например, для треугольника получают:

$$A(x_1, y_1, z_1), \quad B(x_2, y_2, z_2), \quad C(x_3, y_3, z_3);$$

$$n_A = \begin{bmatrix} i & j & k \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{bmatrix} \quad u \quad m.д.$$

Таким образом, коэффициенты плоскости вычисляют по следующей системе уравнений:

$$\begin{cases} a = \sum_{i=1}^{3} (y_i - y_j)(z_i + z_j); \\ b = \sum_{i=1}^{3} (z_i - z_j)(x_i + x_j); \\ c = \sum_{i=1}^{3} (x_i - x_j)(y_i + y_j). \end{cases}$$

где $j=i+1$, если $i=3$, то $j=1$, а коэффициент d определяют с использованием любой точки на плоскости.

2) информация о закраске или цвете многоугольника;

3) булева переменная (флаг), определяющая положение внутри/вне многоугольника (она используется при обработке сканирующей строки; вначале этой переменной присваивается значение false}.

На рис.7.6.1 показаны проекции двух треугольников на плоскость *xy*; невидимые линии изображены пунктиром. В отсортированной ТР для этой фигуры содержатся элементы для AB, AC, FD, FE, CB и DE. В ТМ входят элементы для ABC и DEF.

На третьем шаге создается таблица активных ребер (ТАР), эта таблица всегда упо-

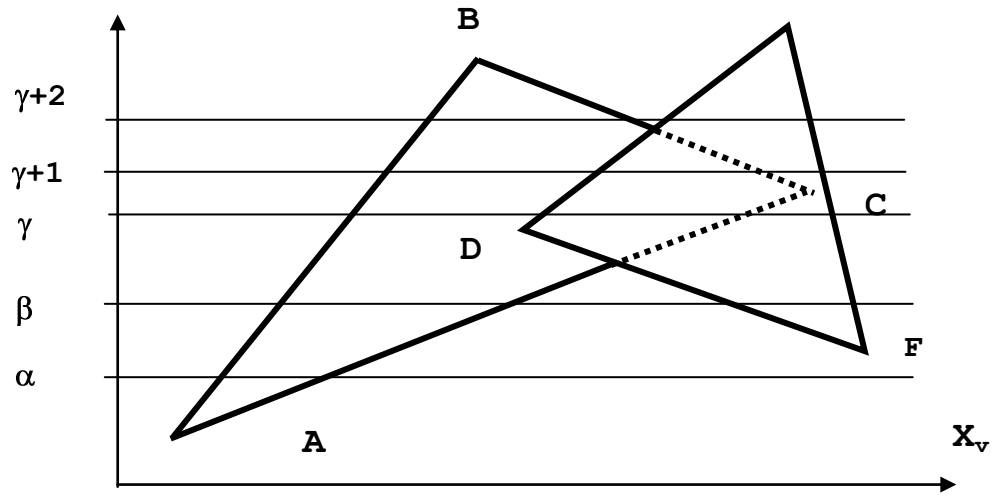

Рис. 7.6.1.

рядочена по возрастанию координаты *x*. К тому времени, когда алгоритм дойдет до сканирующей строки $y = \alpha$, ТАР будет содержать AB и AC в указанном порядке. Ребра рассматриваются в направлении слева направо. Приступая к обработке AB, инвертируем прежде всего флаг внутри/вне треугольника ABC. Тогда флаг примет значение true и, следовательно, мы окажемся «внутри» треугольника, который необходимо рассмотреть. Теперь, поскольку мы находимся внутри только одного треугольника, последний должен быть видимым и, следовательно, на интервале от ребра AB до следующего ребра AC в ТАР будет вестись закраска, требуемая для треугольника ABC. При прохождении этого ребра флаг ABC меняет значение на false, и, следовательно, мы теперь не находимся «внутри» ни одного из треугольников. Поскольку AC является последним ребром в ТАР, обработка сканирующей строки завершается. В ТАР вносятся изменения из ТР, она снова упорядочивается по *x* и производится переход к обработке следующей строки.

Когда встретится сканирующая строка $y = \beta$, в отсортированной ТАР будут находиться AB, AC, FD и FE. Обработка продолжается в основном так же, как прежде. Со сканирующей строкой пересекаются два треугольника, но мы в каждый момент будем находиться «внутри» только одного из них.

Особый интерес представляет сканирующая строка $y = \gamma$. При входе в ABC флаг треугольника становится равным true. На интервале от точки входа до следующего ребра DE ведется закраска, требуемая для ABC. В точке пересечения с DE флаг DEF также устанавливается в true, и, следовательно, мы будем находиться «внутри» сразу двух треугольников (полезно иметь отдельный список многоугольников, флаги внутри/вне которых принимают значение true, а также счетчик, показывающий сколько многоугольников находится в списке). Очевидно, что теперь нам надо решить, какой из треугольников расположен ближе к наблюдателю - ABC или DEF. Определение производится путем вычисления значений *z* из уравнений плоскости для обоих треугольников при $y = \gamma$ и при *x*, задаваемом пересечением строки $y = \gamma$ с ребром DE. Эта величина *x* содержится в элементе для DE в TAP. В нашем примере треугольник DEF имеет меньшую координату *z* и поэтому мы его видим. Таким образом, правило закраски DEF действует на всем интервале вплоть до пересечения с ребром BC, где флаг ABC примет значение false, после чего интервал снова будет «внутри» только одного треугольника DEF. Поэтому правило закраски, установленное для DEF, продолжает действовать вплоть до пересечения с ребром FE.

Предположим, что за треугольниками ABC и DEF находится большой многоугольник GHIJ (рис. 7.6.1). Тогда, если при движении вдоль строки $y = \gamma$, мы пересечем ребро CB, будем все еще находиться «внутри» многоугольников DEF и GHIJ и, следовательно, вычисления, связанные с определением глубины, будут производиться снова. Этих затрат процессорного времени можно избежать, если предположить, что многоугольники не могут проникать друг в друга (при моделировании реальных объектов такое допущение вполне приемлемо). Из этого предположения вытекает, что соотношение глубин между DEF и GHIJ после выхода из ЛВС измениться не может, и, мы, следовательно, будем продолжать оставаться «внутри» DEF. Поэтому при выходе из закрываемого многоугольника глубину можно не вычислять, ее требуется определять только в том случае, если мы выходим из закрывающего многоугольника.

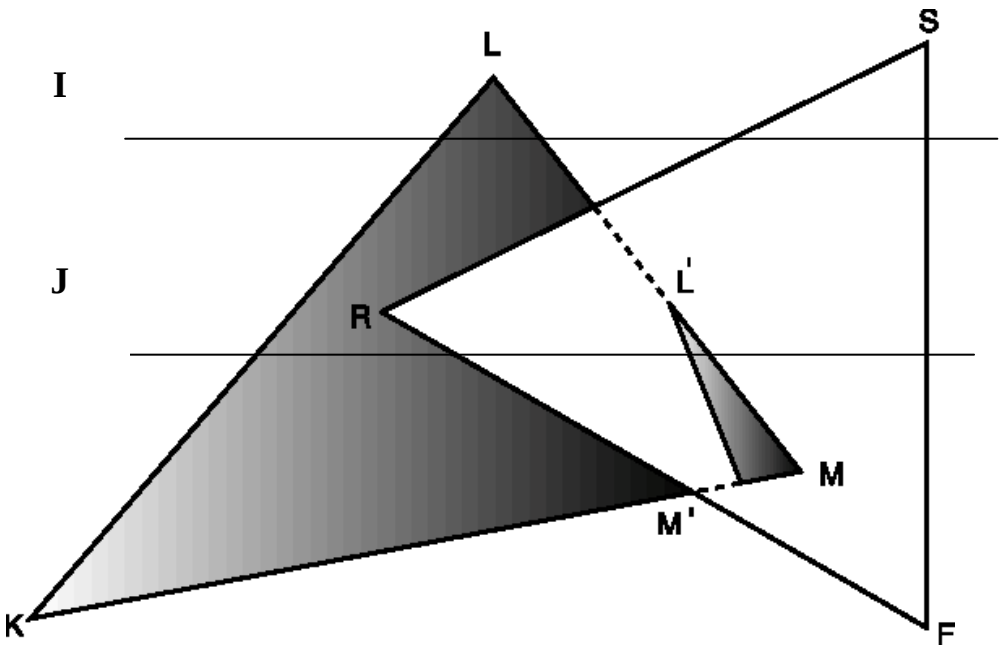

Рис. 7.6.2. Проникающие многоугольники.

На рис. 7.6.2 показаны проникающие друг в друга треугольники. Чтобы правильно использовать алгоритм, введем в рассмотрение ложное ребро M'L' и тем самым разобьем треугольник KLM на KLL'M' и L'MM'.

Кроме того, можно модифицировать алгоритм и при обработке сканирующей строки на ней определять точку проникновения. В другой модификации алгоритма используется когерентность по глубине. Если при обработке некоторой сканирующей строки в ТАР находятся те же ребра, что и при рассмотрении непосредственно предшествующей строки, причем в том же порядке, то соотношения глубин остаются прежними и их не надо вычислять. В этом случае видимые интервалы, найденные при обработке предыдущей строки (от AB до DE и от DE до EF на рис.7.6.1), сохраняются и на следующей сканирующей строке.

На рис. 7.6.1 когерентность по глубине теряется при переходе от $y = \gamma$ к $y = \gamma + 1$, поскольку в ТАР меняется взаимная упорядоченность ребер DE и CB (эта ситуация в ал-

горитме должна быть предусмотрена). Поэтому видимыми станут другие интервалы, в нашем случае от AB до CB и от DE до FE. Можно показать, как иногда может сохраняться когерентность по глубине, даже если меняется порядок вхождения ребер в TAP. В случае буфера регенерации простейшим способом является начальное присвоение пикселям некоторого заданного значения; следовательно, алгоритм должен обрабатывать лишь те сканирующие строки, которые пересекают ребра. Другой способ состоит в том, что в описание объекта включается большой квадрат, расположенный дальше от точки зрения, чем все остальные многоугольники. Этот квадрат параллелен картинной плоскости и имеет желаемую окраску. Наконец, еще один способ заключается в такой модификации алгоритма, при которой значения пикселей, соответствующие фону, можно непосредственно помещать в буфер регенерации в тех случаях, когда мы не находимся «внутри» многоугольника.

7.7. Алгоритм, построенный на использовании Z - буфера

Данный алгоритм удаления невидимых поверхностей является одним из простейших. Этот алгоритм работает в **пространстве изображения**. Здесь используется идея о буфере кадра [1,2,4,6]. Буфер кадра (регенерации) используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения. Для него требуется буфер регенерации, в котором запоминаются значения яркости, а также Z-буфер (буфер глубины), куда можно помещать информацию о координате z для каждого пикселя. Вначале в Z-буфер заносятся максимально возможные значения z, а буфер регенерации заполняется значениями пикселя, описывающими фон. Затем каждый многоугольник преобразуется в растровую форму и записывается в буфер регенерации, при этом, однако, не производится начального упорядочения.

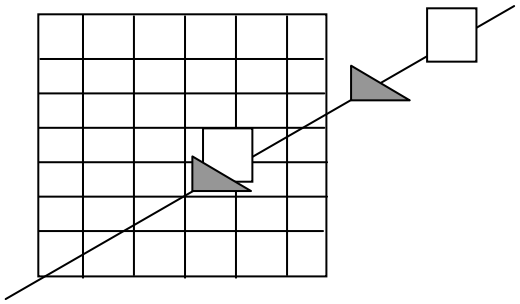

Рис. 7.7.1. Алгоритм Z – буфера.

В процессе работы глубина (значение координаты Z) каждого нового пикселя, который надо занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесён в Z -буфер. Если это сравнение показывает, что .новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра. Кроме того, производится корректировка Z -буфера: в него заносится глубина нового пикселя. Если же глубина (значение координаты Z) нового пикселя меньше глубины хранящегося в буфере, то никаких действий производить не надо. В сущности алгоритм для каждой точки (x, y) ищет наибольшее значение функции $Z(x, y)$(рис. 7.7.1).

Этот алгоритм позволяет удалять сложные поверхности и визуализировать пересечения таких поверхностей. Сцены могут быть произвольной сложности, а поскольку размеры изображения ограничены размером экрана дисплея, то трудоемкость алгоритма зависит линейно от числа рассматриваемых поверхностей. Элементы сцены заносятся в буфер кадра в произвольном порядке, поэтому в данном алгоритме не тратится время на выполнение сортировок, необходимых в других алгоритмах.

К недостаткам алгоритма следует отнести довольно большие объёмы памяти, а также трудоемкость устранения лестничного эффекта и трудность реализации эффектов прозрачности.

Формально данный алгоритм можно описать следующим образом.:

1. Заполнение буфера кадра фоновый значением интенсивности (цвета).
2. Заполнение Z - буфера минимальным значением Z .
3. Преобразование каждого многоугольника в растровую форму в произвольном порядке.
4. Вычисление для каждого пикселя с координатами (x, y), принадлежащего многоугольнику, его глубины $Z(x, y)$.
5. Сравнение глубины $Z(x, y)$ со значением $Z_{буф}(x, y)$, хранящимся в Z-буфере для пикселя теми же координатами (x, y). Если $Z(x, y) > Z_{буф}(x, y)$, то записать атрибут очередного многоугольника в буфер .кадра и $Z_{буф}(x, y)$ заменить на значение $Z(x, y)$.

Предварительно для выпуклых многогранников целесообразно удалить нелицевые грани.

Для этого сначала определяется нормаль к грани - направленный от тела вектор *N*, перпендикулярный к грани (уравнение нормали легко получается из уравнения плоскости, содержащей грань).

Для определения нормали к плоскости вычисляют нормали к каждой вершине многоугольника с помощью векторного произведения прилежащих ребер. Например, для треугольника имеем:

$$A(x_1, y_1, z_1), \quad B(x_2, y_2, z_2), \quad C(x_3, y_3, z_3);$$
$$n_A = \begin{bmatrix} i & j & k \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{bmatrix} \quad u \quad m.д.$$

а затем усредняют результаты.

Затем строится линия визирования *L* - прямая, проходящая через точку наблюдения и основание нормали, и вычисляется угол между *N* и *L*. Грань потенциально видима, если этот угол меньше или равен $\pi/2$, и невидима, если угол больше $\pi/2$. Поскольку «критическим углом» является угол $\pi/2$, то достаточно вычислить скалярное произведение $L \cdot N = |L| \cdot |N| \cdot \cos \phi_{LN}$ и проверить его знак ($\cos \phi_{LN}$ изменяет знак при $\phi_{LN} = \pi/2$). Как отмечено выше, для одного выпуклого тела такой способ полностью определяет видимость его граней. Если тело не одно, этот способ можно использовать лишь для выделения невидимых граней.

Процедура определения невидимости занимает сравнительно немного времени, поскольку достаточно найти одну грань, заслоняющую другую. Зато процедура определения видимости является относительно медленной, поскольку элемент может считаться видимым только после проверки всех граней, которые могут его заслонять. Поэтому от доли видимых граней в общем числе граней сильно зависит расход машинного времени при решении задачи удаления невидимых линий или поверхностей.

Для вычисления глубины каждого пикселя на сканирующей строке поступают следующим образом: уравнение плоскости, несущей многоугольник записывают в виде

$$Ax + By + Cz + D = 0.$$

Отсюда $Z = -(Ax + By + D)/C$ при $C \neq 0$. Для сканирующей строки $y = const$, глубина пикселя, для которого $x_1 = x + \Delta x$, равна:

$$z_1 = \frac{Ax + By + D}{C} - \frac{A\Delta x}{C} = z - \frac{A}{C}\Delta x.$$

Поскольку $\Delta x = 1$, то $z_1 = z - A/C$. Если $C = 0$, то плоскость многоугольника параллельна оси Z. (Для наблюдателя такой многоугольник вырождается в линии.) Глубина пикселя, являющегося пересечением сканирующей строки с ребром многоугольника, вычисляют следующим образом. Сначала определяют ребра грани, вершины которых лежат по разные стороны от сканирующей строки (одна из вершин ребра может в крайнем случае лежать на сканирующей строке), так как только в этом случае сканирующая отрока пересекает ребро. Затем из найденных точек пересечения выбирают ближайшую к наблюдателю. Глубину точки пересечения определяют по соотношению

$$z_3 = z_2 + \frac{y_3 - y_2}{y_2 - y_1}(z_2 - z_1).$$

где (y_1, z_1) и (y_2, z_2) - координаты вершин проекции ребра на плоскость YOZ; (x_3, z_3) - координаты проекции точки пересечения на ту же плоскость.

Уравнение проекции ребра имеет вид

$$\frac{z - z_2}{z_2 - z_1} = \frac{y - y_2}{y_2 - y_1};$$

уравнение проекции плоскости $y = const(y = y_3)$.. Алгоритм, использующий Z - буфер, можно применять для построения разрезов поверхностей. В этом случае изменяется только операция сравнения глубины пикселя со значением, занесенным в буфер:

$$[Z(x,y) > Z_{буф}(x,y)] \text{ и } (Z(x,y) \le Z_{разр}),$$

где $Z_{разр}$ - глубина искомого разреза.

В этом случае остаются только элементы поверхности, которые лежат на плоскости разреза или сзади нее.

Для построения прозрачных поверхностей можно применять модифицированный вариант Z - буфера ЧГЕЧ -буфер. Если рассматриваются монохромные, почти прозрачные поверхности, то в этом случае достаточно наряду о Z - буфером иметь ещё один буфер с информацией о текущей интенсивности пикселя. При построении непрозрачной поверхности (как и в Z - буфере) используется Z -буфер, при этом обновляется информация и в ЧГЕЧ - буфере. При построении прозрачной поверхности расстояние до очередного пикселя изображения сравнивается с элементом Z -буфера. Он будет построен, если находится ближе к наблюдателю, при этом элемент. Z - буфера не модифицируется, а в элемент ЧГЕЧ - буфера. заносится информация о новом цвете пикселя по правилу:

$$I = T * I_0;$$
$$T = (1 - (4\pi\ D + M))\exp(-S * l),$$

где I интенсивность пикселя; T коэффициент прозрачности; D, M - коэффициенты диффузного и зеркального отражений; S - коэффициент поглощения; l - путь луча в рассматриваемом слое; I_0 - текущее содержимое $ALFA$ - буфера (коэффициент прозрачности должен приближаться к I). Ограничение на значение коэффициента прозрачности T позволяет не учитывать относительное расположение прозрачных поверхностей и вычислять результирующую интенсивность пикселя в линейной приближении. В заключение можно отметить, что эксплуатационные характеристика алгоритма с Z - буфером остаются практически постоянными, так как с ростом числа многоугольников в видимом объеме уменьшается число пикселей, покрываемых одним многоугольником.

7.8. Алгоритмы для криволинейных поверхностей

Рассмотренные выше алгоритмы применимы лишь к объектам, состоящим из плоских граней. Чтобы воспользоваться любым из них для изображения объектов, ограниченных криволинейными поверхностями, необходимо предварительно аппроксимировать эти объекты большим числом мелких граней [5,6]. Хотя это можно сделать, иногда удобнее работать непосредственно с криволинейными поверхностями. Квадратичные поверхности в общем случае являются неплоскими и могут быть записаны в аналитическом виде

$$a_1x^2 + a_2y^2 + a_3z^2 + a_4xy + a_5yz + a_6zx + a_7x + a_8y + a_9z + a_{10} = 0$$

Если коэффициенты $a_1 - a_9$ равны нулю, поверхность вырождается в плоскость. Хорошо известными объектами, состоящими из квадратичных поверхностей, являются сфера (одна поверхность), закрытый цилиндр (три поверхности), закрытый конус (две поверхности) и эллипсоид (одна поверхность). Широко известны алгоритмы, в которых определяются пересечения двух квадратичных поверхностей, это приводит к уравнению четвертого порядка относительно *x, y* и *z*, корни которого находятся численными методами. Можно понизить порядок уравнения до второго путем параметризации кривых пересечения.

Более универсальными являются параметрические бикубические поверхности, поскольку обладают большей общностью и обеспечивают непрерывность касательной на

границах куска. В алгоритм, позволяющих изображать такие поверхности, разбиение куска (по *s* и по *t*) производится до тех пор, пока его проекция на плоскость экрана не станет примерно равной размеру пикселя. Определение видимости такой маленькой области (относительно всех ранее рассмотренных кусков) выполняется с помощью алгоритма, использующего *z*-буфер. Вычисляется закраска этой области, и соответствующее значение помещается в буфер регенерации. Метод быстр, но весьма эффективен.

На сегодняшний день наиболее плодотворным является подход, который основан на адаптивном процессе разбиения бикубических кусков, продолжающемся до тех пор, пока каждый из получающихся в результате деления кусочков не станет достаточно плоским. Допустимые отклонения определяются разрешающей способностью дисплея, а также ориентацией подразделяемой области относительно картинной плоскости. Поэтому разбиения, в которых нет необходимости, не производятся. После завершения процесса разбиения мелкие многоугольники, задаваемые четырьмя угловыми точками каждого из получившихся в результате деления кусков, обрабатываются при помощи алгоритма построчного сканирования. Тем самым мы получаем возможность одновременно использовать многогранные и бикубические поверхности