Алтайский
государственный технический
университет им. И. И. Ползунова

А.Ю. Андреева

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебное пособие

Барнаул 2015

Министерство образования и науки Российской Федерации
федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. И.И. ПОЛЗУНОВА»

А.Ю. Андреева

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебное пособие

Барнаул 2015 г.

УДК 681.3

Андреева А.Ю. Компьютерная графика: Учебное пособие по дисциплине “Компьютерная графика” для студентов ИТ-направлений. – Барнаул: Изд-во АлтГТУ, 2015 г.- 118 с.

Пособие включает основы алгоритмической графики: классические алгоритмы аффинных преобразований, удаления невидимых поверхностей, основы создания трехмерных изображений. Кроме того, рассмотрены базовые алгоритмы растровой графики и способы коррекции растровых изображений.

Рекомендовано к печати кафедрой Прикладная математика Алтайского государственного технического университета им. И.И. Ползунова

Содержание

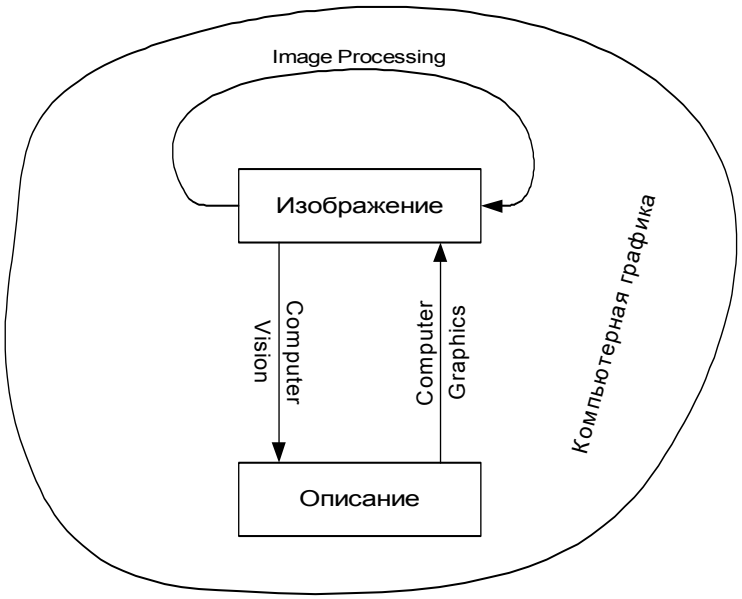
Введение

При обработке информации, связанной с изображением на мониторе, принято выделять три основных направления: распознавание образов, обработку изображений и машинную графику.

Основная задача распознавания образов состоит в преобразовании уже имеющегося изображения на формально понятный язык символов. ***Распознавание образов или система технического зрения (COMPUTER VISION)*** – совокупность методов, позволяющих получить описание изображения, поданного на вход, либо отнести заданное изображение к некоторому классу (так поступают, например, при сортировке почты). Одной из задач COMPUTER VISION является так называемая склетизация объектов, при которой восстанавливается некая основа объекта, его «скелет».

Обработка изображений (IMAGE PROCESSING) рассматривает задачи в которых и входные и выходные данные являются изображениями. Например, передача изображения с устранением шумов и сжатием данных, переход от одного вида изображения к другому (от цветного к черно белому) и т.д.

Компьютерная (машинная) графика (COMPUTER GRAPHICS) воспроизводит изображение в случае, когда исходной является информация неизобразительной природы. Например, визуализация экспериментальных данных в виде графиков, гистограмм или диаграмм, вывод информации на экран компьютерных играх, синтез сцен на тренажерах.



Компьютерная графика – это наука, предметом изучения которой является создание, хранение и обработка моделей и их изображений с помощью ЭВМ. В том случае, если пользователь может управлять характеристиками объектов, говорят об интерактивной компьютерной графике

1 Основные понятия компьютерной графики

1.1 Способы задания геометрических объектов

Бесспорно, геометрическое моделирование является одним из наиболее важных областей человеческой деятельности, поскольку оно часто используется при проектировании. Например, при проектировании архитектурных строений, механических узлов, нового дизайна изделий бытового назначения и других предметов. Основой точного геометрического моделирования является прикладная геометрия. Однако в силу возрастающей мощности компьютерных систем эта область пополнилась новыми технологиями, такими как твердотельное моделирование, B-сплайны, лофтинг, скиннинг и др. Они успешно применяются на практике. Однако, несмотря на большие успехи в этом направлении, вопросы синтеза моделей геометрических объектов все еще остаются актуальными. Основные из них это:

- повышение быстродействия выполнения операций с геометрическими моделями;
- разработка технологий, упрощающих построение сложных геометрических объектов;
- интеграция разнородных геометрических моделей.

Рассмотрим распространенную классификацию существующих способов представления трехмерных геометрических объектов (моделей):

- Простейшие
- Поверхностные
- Объемные

1.1.1 Простейшие способы

К простейшим способам задания трехмерных объектов относятся точечное и проволочное (каркасное) представления.
В точечном представлении объект задан совокупностью вершин, принадлежащих поверхности объекта V = {V1,…,Vn}.
Проволочная модель является расширением предыдущего способа. Объекты задаются совокупностью вершин и соединяющих их ребер (отрезков прямой или кривой соединяющей вершины vi, vj.):
V = {v1,…,vn}, E = {vi, vj },
Основное преимущество этих способов - простота представления. Потому они применяются на промежуточных стадиях работы с геометрическим объектом: предварительная визуализация и как исходные модели, для синтеза более сложных моделей.

1.1.2 Поверхностное представление (Boundary representation, B-rep)

В поверхностном представлении объект создается при помощи набора тонких поверхностей, составляющих его границу. Как правило, поверхностное представление используется в тех областях, где нет необходимости обрабатывать каким-либо образом внутренность тела: дизайнерские проекты, моделирование обводов какого-либо изделия, создание объектов с нестандартными элементами (например, скругление с изменяемым радиусом, винтообразная улитка) и т. д.
Существует несколько основных способов задания границы тела.

Неявная функция
Задание 3D объекта при помощи неявной функции состоит в том, что мы указываем некую функцию *F(x,y,z)*, нулями которой будут точки (x_i, y_i, z_i), образующие нашу поверхность, либо дающие ее приближение. Так, например, неявная функция для сферы радиуса r (см. Рис. 1.1), с центром в точке (x_0, y_0, z_0) имеет вид:

$$F(x,y,z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2$$

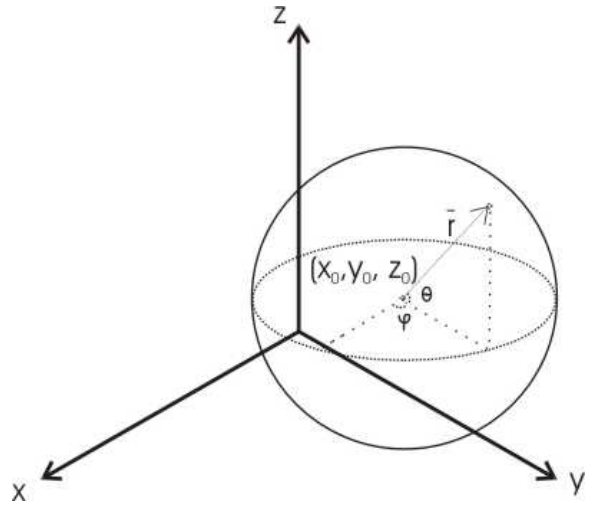


Рис. 1.1. Сфера, заданная неявной функцией

Параметрическое задание, сплайны.

При параметрическом задании координаты точек поверхности рассматриваются как некие функции от двух параметров, пробегающих некоторый набор значений.

$$\begin{cases} x = x(u,v) \\ y = y(u,v) \\ z = z(u.v) \end{cases}$$

где, для параметров u и v, как правило, определяют область определения прямоугольного (*ua* < *u* < *ub, va* < *v* < *vb*) либо треугольного (*ua* < *u* < *ub, va* < *u* + *v* < *vb*) вида.

Так, например, параметризация той же сферы (см. Рис. 1.7) относительно двух углов φ (долгота) и θ (широта) будет иметь вид:

$$\begin{cases} x = r\cos\theta\cos\varphi \\ y = r\cos\theta\sin\varphi, \quad 0 \le \varphi < 2\pi, \ -\dfrac{\pi}{2} \le \theta \le \dfrac{\pi}{2} \\ z = r\sin\theta \end{cases}$$

Помимо этого используются различного рода особые параметрические поверхности, например, поверхности Безье, B-сплайны и др.

Параметрические поверхности очень широко используются в различных CAD - системах (в России используется термин САПР - системы автоматического проектирования). Так, при помощи них моделируются и рассчитываются обводы автомобилей, формы деталей и т.п.

Полигональные поверхности

При использовании данного метода поверхность представляется в виде набора некоторых многоугольников в пространстве (рис. 1.2).

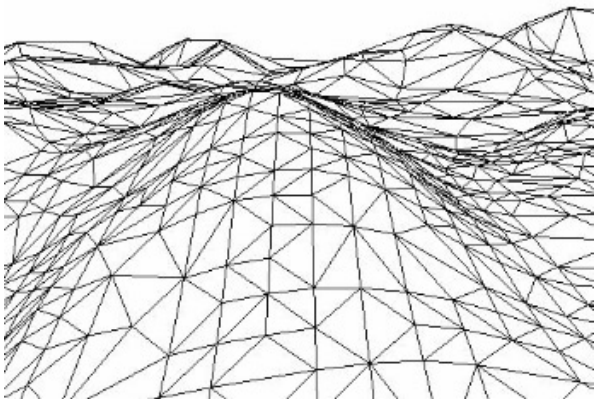


Рис. 1.2. Триангуляция поверхности.

Часто в качестве многоугольников используются треугольники, в этом случае само разбиение поверхности (приближенное представление треугольниками) называется триангуляцией поверхности. Топология полученной при этом сетки описывается следующим образом:

1. Объектами сетки являются вершины, ребра и треугольники (задаваемые тремя вершинами или тремя ребрами), а в более общем случае – многоугольники.
2. У любой вершины есть свойство валентности (т.е. число многоугольников, содержащих ее).
3. Для любого треугольника существует не более одного другого треугольника, инцидентного для первого по фиксированному ребру. Т.е., в частности, не может быть ситуации, приведенной на рисунке 1.3.

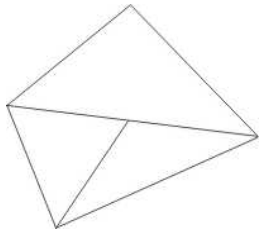


Рис. 1.3 Нарушение условия об инцидентности.

При таком способе задания мы можем, например, организовать обход в некотором направлении по треугольникам, содержащим фиксированную вершину, просто переходя каждый раз к треугольнику, инцидентному по ребру для предыдущего.
Разумеется, при триангуляции качество получаемой поверхности тем лучше, чем больше треугольников мы будем для этого использовать

Полигональное задание трехмерных объектов является наиболее распространенным и для него сформировались следующие разновидности топологий:

1. список вершин. В этой топологии грань выражается через вершины:
   $V = \{vi\}$ – вершины |V| = n;
   $F = \{(vj1, vj2, \ldots, vjk [, fj(u,v)])\}$ – грань (или параметрическая поверхность fj), состоящая из k вершин (k >= 3)
2. список ребер. Здесь грань выражена через ребра:
   $V = \{vi\}$ – вершины |V| = n;
   $E = \{ ek = (vi, vj [, fk(u)])\}$ – ребро (или параметрическое представление линии fk);
   $F = \{ (ej1, ej2, \ldots, ejk [, fj(u,v)])\}$ – грань (или параметрическая поверхность fj) состоящая из k ребер (k >= 3).

3. "крылатое" представление. Эта модель является развитием модели, основанной на информации о ребрах. Отличие состоит в том, что в структуру, описывающую ребра, добавляется информация о взаимном расположении граней

"Крылатое" представление является наиболее удобным для реализации важнейших алгоритмов над геометрическими объектами:
- проверка правильности задания;
- алгоритмы для полигональных моделей, связанные с обходом ребер (выделение плоских контуров, упрощение модели путем удаления граней и другие);
- возможность, не более, чем за линейное время, восстановить любую другую топологию, следуя по цепочкам связей между элементами.

1.1.3 Объемное представление

Методы объемного представления тел сохраняют информацию о любых, не обязательно видимых, частях тела. Все они, как правило, требуют гораздо больше места для хранения объекта, но как уже было сказано, дают дополнительную информацию об объектах, которая может нам потребоваться. Также эти методы необходимы, когда тела не имеют как таковой границы (например, туман, взвесь в жидкости и т.д.)

Конструктивная геометрия тел (Constructive Solid Geometry, CSG)

Метод конструктивной геометрии тел заключается в том, что мы получаем нужную нам поверхность как результат применения последовательности различных множественных операций (объединения, пересечения, разности и т.д.) к некоторым примитивам (т.е. минимальным объектами для построения). В качестве примитивов могу выступать параллелепипеды, шары, конусы, пирамиды и т.д. Сама поверхность задается при помощи дерева построения (см. рис 1.4).

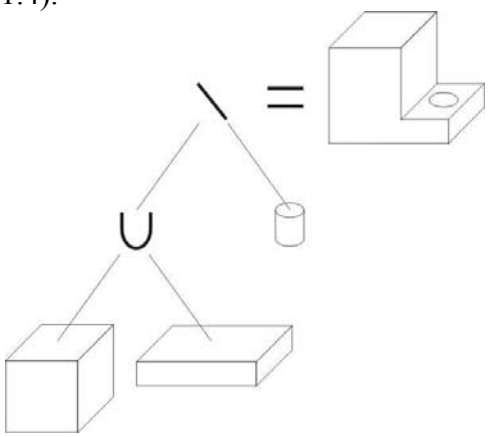


Рис. 1.4. Дерево построения.

3D растр (Воксельное представление)

3D растр представляет собой набор кубиков (см. рис. 1.5) (voxel'ов, от «volume element») в пространстве. Каждый воксел может иметь некоторое числовое значение, являющееся атрибутом соответствующей точки в пространстве.

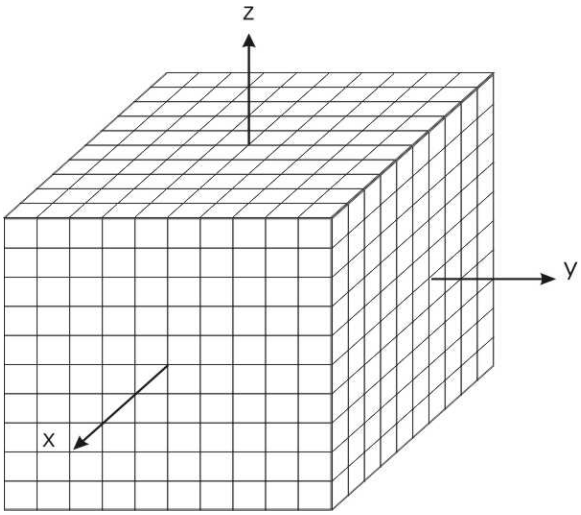


Рис. 1.5. 3D растр.

Данный метод, несмотря на свою простоту, имеет серьезный недостаток: для хранения даже небольшого объекта в приемлемом качестве требуется очень много места. Например, если мы храним кубик с ребром 1024 и выделяем по одному байту (8 бит) на атрибут воксела, то нам потребуется $1024 \times 1024 \times 1024 \times 1 байт = 1 Гб$. Учитывая тот факт, что хранить объект нам, скорее всего, надо будет в оперативной памяти компьютера, применение этого метода в чистом виде становится крайне затруднительным.

Воксельное представление является очень удобным для реализации пространственных алгоритмов и теоретико-множественных операций над объектами (объединение, вычитание, пересечение), но обладает рядом недостатков, которые ограничивают область его применения:

- низкая точность для представления для большинства объектов;
- большой объем занимаемой памяти;

Восьмеричное дерево

В большинстве случаев в 3D растре есть некоторые области вокселов, имеющих одинаковые атрибуты. Этим можно воспользоваться для того, чтобы сократить объем занимаемого объектом места. Будем хранить растр в виде дерева (см. 1.6), вершины которого имеют степень 0 или 8. Сначала проверим, не лежат ли во всех вокселах в растре одинаковые значения, если да, то нам достаточно будет сохранить это значение (*p)* и обозначить, что дальше дробить кубик не надо, например, положим в вершину это значение *p* и 0 (обозначение терминальности вершины). Если у нас есть вокселы с разными значениями, то продолжим процесс. А именно, разделим растр на октанты, соответствующие координатным плоскостям. Для каждого октанта снова проверим, не содержит ли он одинаковые вокселы. И т.д. Получим дерево, в вершинах которого лежат 1 (что обозначает, что соответствующий октант не однороден) или 0 и некоторые числа p_i, соответствующие значениям всех вокселов в данном октанте.

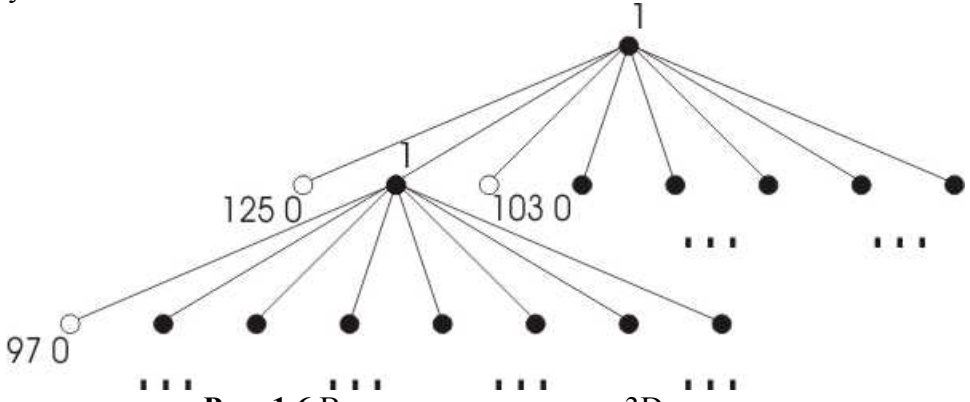


Рис. 1.6 Восьмеричное дерево 3D растра.

Таким образом, если растр состоит из вокселов, имеющих один и тот же атрибут, то мы сэкономим $1\,Гб - (1байт + 1бит)$; если же все воксели различны, то лишнее место, которое мы потратим (на 1 и 0), составит $(1+8+8^2+...+8^{10})\,битов = 1227133513\,битов \approx 146\,мб$. Но последняя ситуация встречается крайне редко, поэтому использование восьмеричных деревьев в большинстве случаев позволяет значительно сократить объем памяти для хранения объекта.

Двоичное дерево

Идея построения двоичного дерева полностью аналогична построению восьмеричного дерева. Но в данном случае растр последовательно делится не на октанты, а на половинки: сначала параллельно плоскости OXY, потом OYZ и т.д. Соответственно, аналогично строится дерево, вершины которого имеют степень 0 или 2.

1.2 Растровые представления изображений.

1.2.1 Виды растра

Цифровое изображение – набор точек (пикселей) изображения; каждая точка изображения характеризуется координатами x и y и яркостью V(x,y), это дискретные величины, обычно целые. В случае цветного изображения, каждый пиксель характеризуется координатами x и y, и тремя яркостями: яркостью красного, яркостью синего и яркостью зеленого (V_R , V_B , V_G). Комбинируя данные три цвета можно получить большое количество различных оттенков.

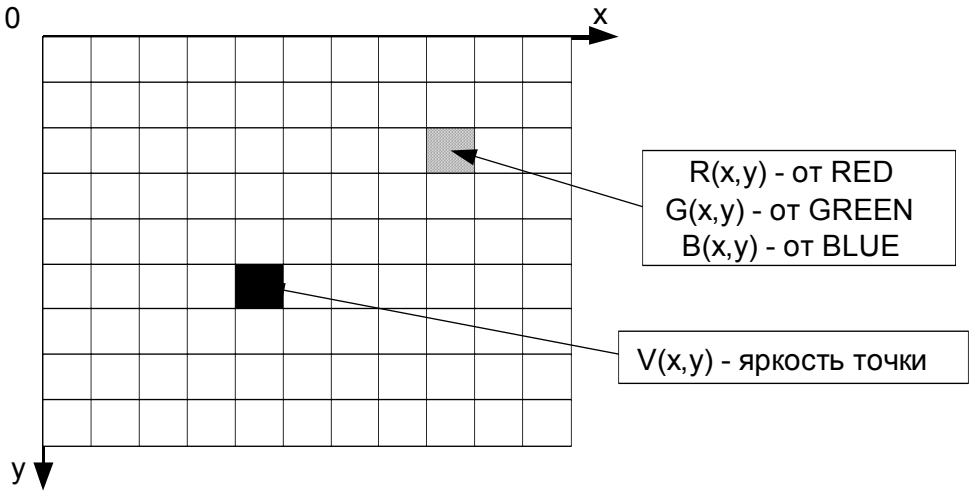


Рис. 1.7 Квадратный растр

Растр – это порядок расположения точек (растровых элементов). На рис. 1.7 изображен растр элементами которого являются квадраты, такой растр называется ***квадратным***, именно такие растры наиболее часто используются. Хотя возможно использование в качестве растрового элемента фигуры другой формы, соответствующего следующим требованиям:

1. Все фигуры должны быть одинаковые;
2. Должны полностью покрывать плоскость без наезжания и дырок.

Так в качестве растрового элемента возможно использование равностороннего треугольника, правильного шестиугольника (гексаэдра) рис. 1.8. Можно строить растры, используя неправильные многоугольники, но практический смысл в подобных растрах отсутствует.

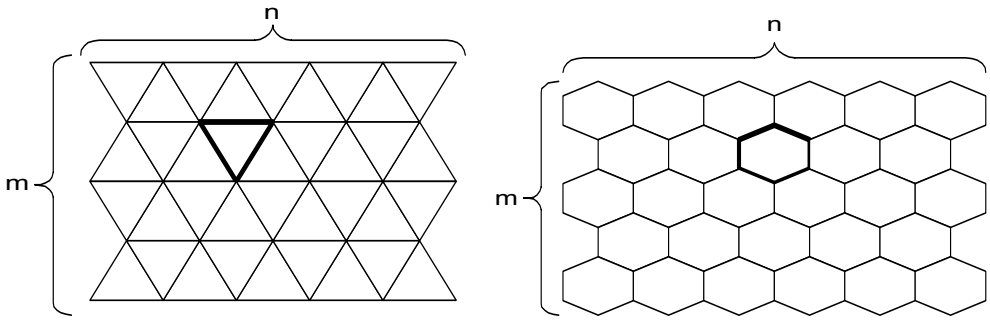


Рис. 1.8 Треугольный и гексагональный растр

Рассмотрим способы построения линий в прямоугольном и гексагональном растре.
В квадратном растре построение линии осуществляется двумя способами:

1) Результат – восьмисвязная линия. Соседние пиксели линии могут находится в одном из восьми возможных (см. рис. 1.9а) положениях. Недостаток – слишком тонкая линия при угле 45°.
2) Результат – четырехсвязная линия. Соседние пиксели линии могут находится в одном из четырех возможных (см. рис. 1.9б) положениях. Недостаток – избыточно толстая линия при угле 45°.

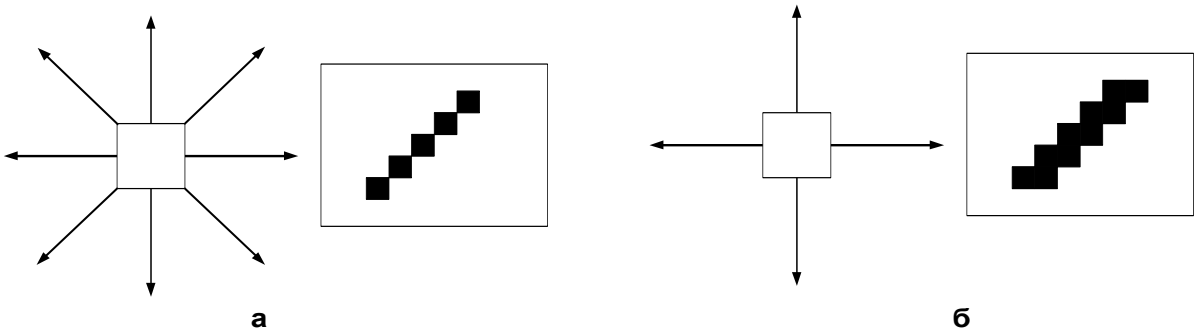


Рис. 1.9. Построение линии в прямоугольном растре

В гексагональном растре линии шестисвязные (см. рис. 1.10) такие линии более стабильны по ширине, т.е. дисперсия ширины линии меньше, чем в квадратном растре.

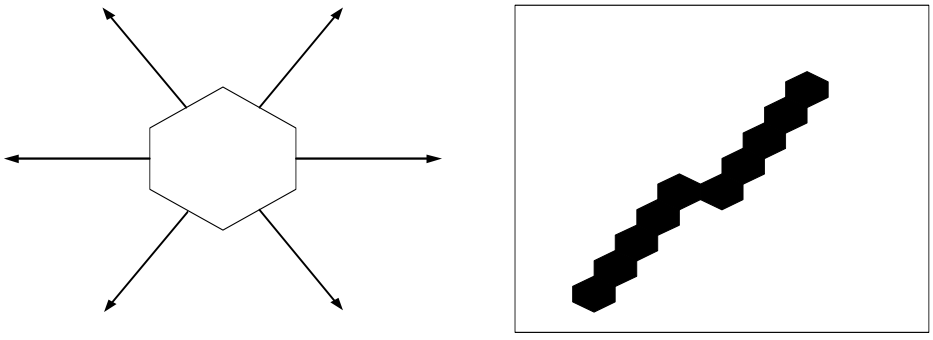


Рис. 1.10. Построение линии в гексагональном растре

Каким образом можно оценить, какой растр лучше?

Одним из способов оценки является передача по каналу связи кодированного, с учетом используемого растра, изображения с последующим восстановлением и визуальным анализом достигнутого качества. Экспериментально и математически

доказано, что гексагональный растр лучше, т.к. обеспечивает наименьшее отклонение от оригинала. Но разница не велика.

1.2.2 Построение линии в квадратном растре.

Поскольку экран растрового дисплея можно рассматривать как матрицу дискретных элементов (пикселей), каждый из которых может быть подсвечен, нельзя непосредственно провести отрезок из одной точки в другую. Процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок, называется разложением в растр. В сочетании с процессом построчной визуализации изображения он известен как преобразование растровой развертки. Для горизонтальных, вертикальных и наклоненных под углом 45° отрезков выбор растровых элементов очевиден. При любой другой ориентации выбрать нужные пиксели труднее, что показано на рис. 1.11

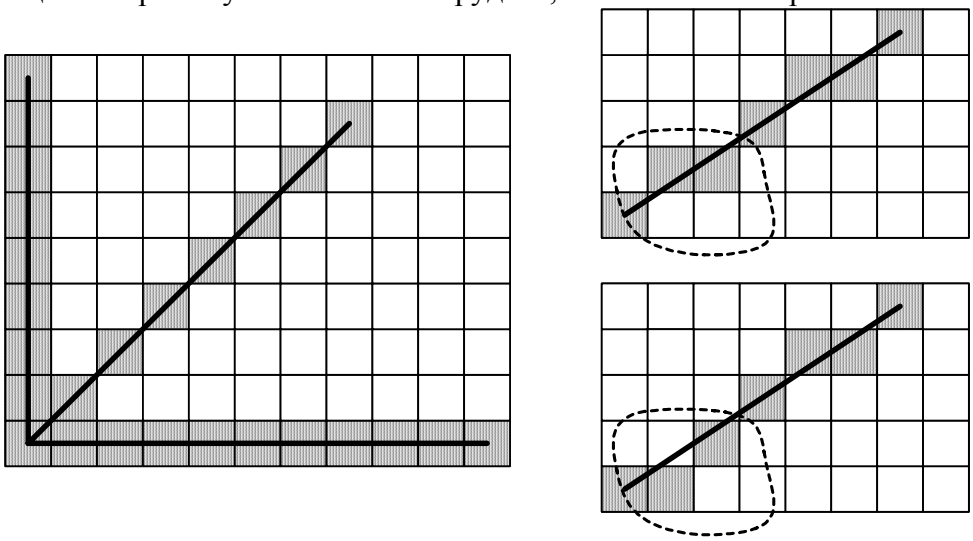


Рис. 1.11. Разложение в растр отрезков

Общие требования к изображению отрезка.

- концы отрезка должны находиться в заданных точках;
- отрезки должны выглядеть прямыми,
- яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона.

Ни одно из этих условий не может быть точно выполнено на растровом дисплее в силу того, что изображение строится из пикселов конечных размеров, а именно:

- концы отрезка в общем случае располагаются на пикселах, лишь наиболее близких к требуемым позициям и только в частных случаях координаты концов отрезка точно совпадают с координатами пикселов;
- отрезок аппроксимируется набором пикселов и лишь в частных случаях вертикальных, горизонтальных и отрезков под 45° они будут выглядеть прямыми, причем гладкими прямыми, без ступенек только для вертикальных и горизонтальных отрезков;
- яркость для различных отрезков и даже вдоль отрезка в общем случае различна, так как, например, расстояние между центрами пикселов для вертикального отрезка и отрезка под 45° различно (см. рис. 1.11).

Объективное улучшение аппроксимации достигается увеличением разрешения дисплея, но в силу существенных технологических проблем разрешение для растровых систем приемлемой скорости разрешение составляет порядка 1280×1024.

Субъективное улучшение аппроксимации основано на психофизиологических особенностях зрения и, в частности, может достигаться просто уменьшением размеров экрана. Другие способы субъективного улучшения качества аппроксимации основаны на различных программных ухищрениях по "размыванию" резких границ изображения.

1.2.3 Параметрический алгоритм рисования линии.

Необходимо провести линию из точки (x_1, y_1) в точку (x_2, y_2) с линейной интерполяцией по яркости (рис. 1.12).

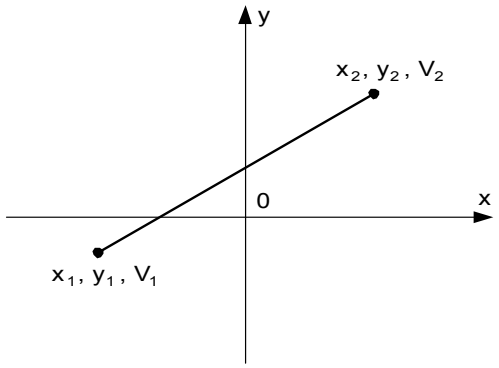


Рис. 1.12 Параметрическое рисование линии

Любую точку на этой линии можно представить в виде

$$x_{i+1} =]x_i + t \cdot (x_2 - x_1)[$$
$$y_{i+1} =]y_i + t \cdot (y_2 - y_1)[$$
$$V_{i+1} =]V_i + t \cdot (V_2 - V_1)[$$

; где $t \in [0;1]$,][– знак округления до целого.

$$t = \frac{1}{\max\{|x_2 - x_1|, |y_2 - y_1|\}} = \frac{1}{N-1};$$

N – длина линии в пикселях.

Можно проводить вычисления через приращение координат.

$$\Delta x = \frac{x_2 - x_1}{N-1}; \Delta y = \frac{y_2 - y_1}{N-1}; \Delta V = \frac{V_2 - V_1}{N-1}$$

Значения приращений считаются в начале функции и не входят в цикл построения линии на экране, за счет чего повышается быстродействие.

Недостатки алгоритма:

- Необходимость работать с вещественными числами.
- В алгоритме есть операция деления, что значительно усложняет аппаратную организацию и увеличивает время работы алгоритма..

Достоинства алгоритма:

- Простота программной реализации.
- Простота реализации линейной интерполяции по яркости.

1.2.4 Алгоритм Брезенхема рисования линии.

В 1965 году Брезенхеймом был предложен простой целочисленный алгоритм для растрового построения отрезка. В алгоритме используется управляющая переменная d_i, которая на каждом шаге пропорциональна разности между s и t (см. рис. 1.13). На рис.1.18 приведен i-ый шаг, когда пиксел P_{i-1} уже найден как ближайший к реальному изображаемому отрезку, и теперь требуется определить, какой из пикселов должен быть установлен следующим: T_i или S_i.

Если s<t, то S_i ближе к отрезку и необходимо выбрать его; в противном случае ближе будет T_i. Другими словами, если s-t<0, то выбирается S_i; в противном случае выбирается T_i.

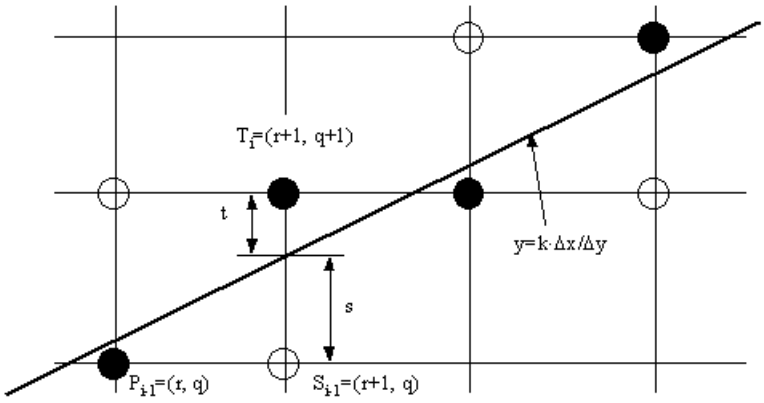


Рис. 1.13. Алгоритм Брезенхейма

Изображаемый отрезок проводится из точки (x_1, y_1) в точку (x_2, y_2). Пусть первая точка находится ближе к началу координат, тогда перенесем обе точки, T(x_1, y_1) так, чтобы начальная точка отрезка оказалась в начале координат, тогда конечная окажется в (Δ x, Δ y), где Δ x= x_2- x_1 , Δ y= x_2- x_1. Уравнение прямой теперь имеет вид y=x· Δ y/Δx. Из рисунка следует, что $s = \frac{\Delta y}{\Delta x}(r+1) - q; t = q + 1 - \frac{\Delta y}{\Delta x}(r+1)$.

поэтому $s - t = 2\frac{\Delta y}{\Delta x}(r+1) - 2q - 1$.

помножим на Δ x: $\Delta x(s-t) = 2(\Delta y \cdot r - q \cdot \Delta x) + 2\Delta y - \Delta x$

так как Δ x>0, величину Δ x(s-t) можно использовать в качестве критерия для выбора пиксела. Обозначим эту величину d_i : $d_i = 2(\Delta y \cdot x_{i-1} - y_{i-1} \cdot \Delta x) + 2\Delta y - \Delta x$

так как r = x_{i-1}, q = y_{i-1}, получаем: $\Delta_i = d_{i+1} - d_i = 2\Delta y(x_i - x_{i-1}) - 2\Delta x(y_i - y_{i-1})$

Известно, что x_I - x_{i-1}=1.

Если d_i >= 0, то выбираем T_i, тогда $\Delta_i = 2(\Delta y - \Delta x)$

Если d_i <0, то выбираем S_i, тогда $\Delta_i = 2\Delta y$

Таким образом, мы получили итеративную формулу для вычисления критерия d_i.

Начальное значение d_1=2Δ y-Δ x.

Можно построить блок схему алгоритма:

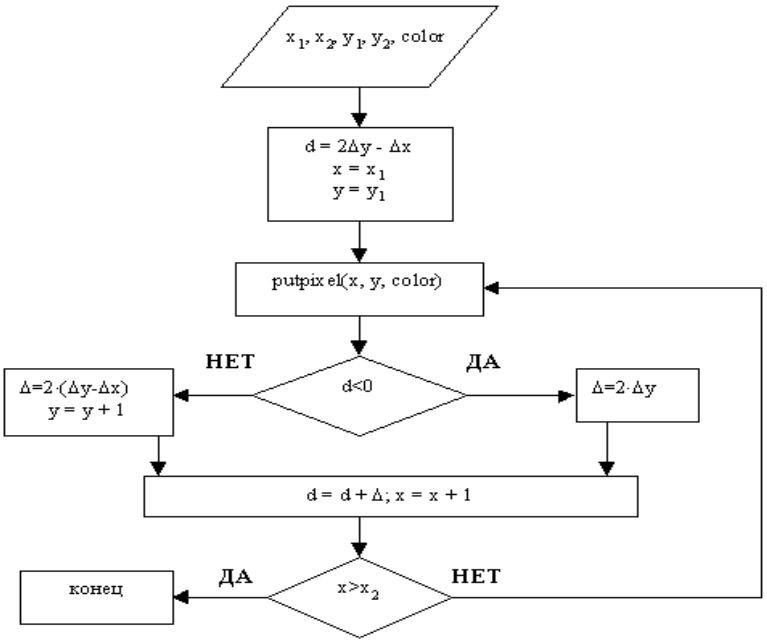


Рис. 1.14 Структурная схема алгоритма

Линейная интерполяция по яркости при построении отрезка по алгоритму Брезенхема получается параметрическим способом, т.е.

$V = V_{i-1} + \Delta V; \Delta V = \frac{V_2 - V_1}{N - 1}$, где N – длина отрезка в пикселях.

1.2.5 Алгоритмы построения окружности.

Рассмотрим окружность с центром в начале координат, для которой $x^2+y^2=R^2$, или в параметрической форме:

x=R· cos(a);
y=R· sin(a).

То есть легко написать программу рисования окружности:

```
void Circle (int x, int y, int R, int color)
{
    int a;
    int x1;
    int x2;
    int y1;
    int y2;
    x2=x+R;
    y2=y;
    for ( int a=1; a<=360; a++)
    {
        x1=x2; y1=y2;
        x2=round(R*cos(a))+x;
        y2=round(R*sin(a))+x;
        Line (x1, y1, x2, y2, color);
    }
}
```

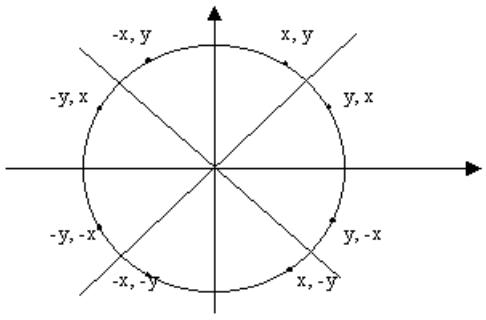


Рис. 1.15. Параметрический алгоритм рисования окружности

Если воспользоваться симметрией окружности, то можно построить более эффективный алгоритм. Если точка (x, y) лежит на окружности, то легко вычислить семь точек, принадлежащих окружности, симметричных этой. То есть, имея функцию вычисления значения y по *x=0..R/SQRT(2)* для построения дуги от 0^0 до 45^0. Построим процедуру, которая будет по одной координате ставить восемь точек, симметричных центру окружности.

```
void Circle_Pixel(int x0, int y0, int x, int y, int color);
{
    putpixel(x0 + x, y0 + y, color);
    putpixel(x0 + y, y0 + x, color);
    putpixel(x0 + y, y0 - x, color);
    putpixel(x0 + x, y0 - y, color);
```

```
    putpixel(x0 - x, y0 - y, color);
    putpixel(x0 - y, y0 - x, color);
    putpixel(x0 - y, y0 + x, color);
    putpixel(x0 - x, y0 + y, color);
}
```

Таким образом можно написать программу рисование окружности по точкам:

```
void Circle (int x0, int y0, int R, int color)
{
    for ( int x=0; x<=R/sqrt(2); x++)
    {
        int y = (int)(sqrt(sqr(R)-sqr(x)));
        Circle_Pixel (x0, y0, x, y, color);
    }
}
```

1.2.6 Алгоритм Брезенхема генерации окружности

Брезенхем разработал алгоритм более эффективный, чем каждый из рассмотренных выше. Здесь используется тот же принцип, что и для рисования линии.

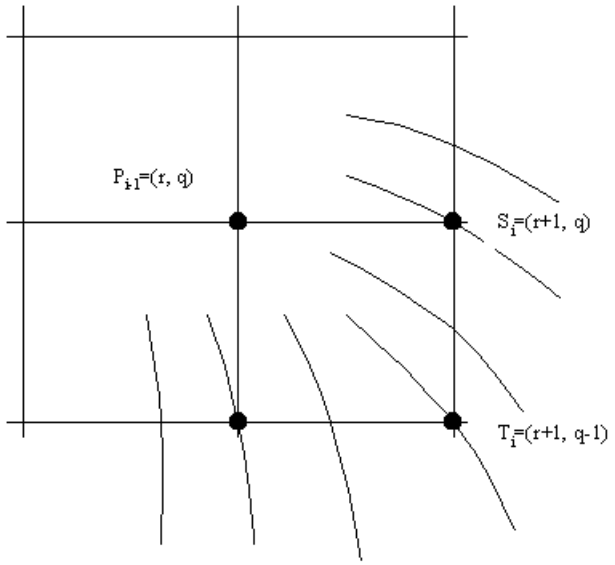


Рис. 1.16. Алгоритм Брезенхейма рисования окружности

Будем рассматривать сегмент окружности, соответствующий $x=x_0.. x_0+R/sqrt(2)$. На каждом шаге выбираем точку, ближайшую к реальной окружности. В качестве ошибки возьмем величину $D(P_i)=(x^2_i+ y^2_i) – R^2$.

Рассмотрим рис. 2.6.1 , на котором показаны различные возможные способы прохождения истинной окружности через сетку пикселей. Пусть пиксел P_{i-1} уже найден как ближайший к реальной изображенной окружности, и теперь требуется определить, какой из пикселей должен быть установлен следующим: T_i или S_i. Для этого определим точку, которой соответствует минимальная ошибка:

$$D(S_i)=((q+1)^2+ p^2) – R^2,\ D(T_i)=((q+1)^2+ (p+1)^2) – R^2.$$

Если $|D(S_i)| < |D(T_i)|$, то выбираем S_i, иначе - T_i. Введем величину $d_i = |D(S_i)| - |D(T_i)|$, тогда S_i выбирается при $d_i < 0$, иначе выбирается T_i. Если рассматривать только часть окружности, дугу от 0^0 до 45^0, то $D(S_i) > 0$ так как точка S_i лежит за пределами

окружности, а $D(T_i) < 0$, так как T_i находится внутри окружности, поэтому $d_i = D(S_i) + D(T_i)$.

Алгебраические вычисления, аналогичные тем, которые проводились для линии, приводят к результату:

$$d_1=3-2R.$$

Если выбираем S_i (когда $d_i < 0$),

$$\Delta_i=4x_{i-1}+6;$$

если выбираем T_i (когда $d_i >=0$),

$$\Delta_i=4(x_{i-1} * y_{i-1}) +10.$$

Блок-схема этого алгоритма:

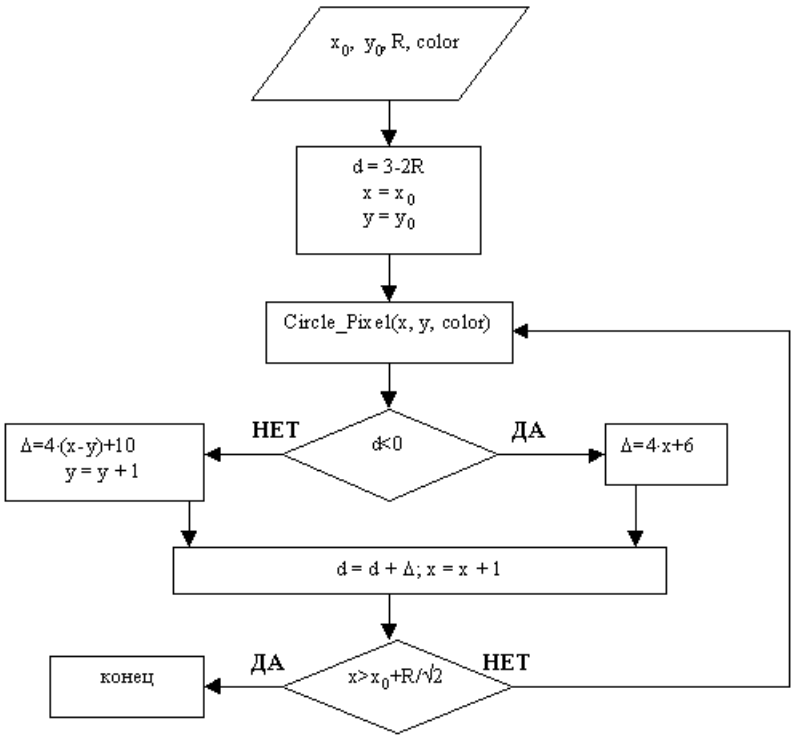


Рис. 1.17. Структурная схема алгоритма

Алгоритм хорош тем что отсутствуют операции с плавающей точкой, а также операции деления и извлечения корня.

2 Отсечение и заливка двумерных многоугольников

2.1 Отсечение сцены по границам окна

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

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

Программное исполнение отсечения достаточно медленный процесс, поэтому, естественно, в мощные дисплеи встраивается соответствующая аппаратура. Здесь мы рассмотрим программные реализации алгоритма отсечения.

Отсекаемые отрезки могут быть трех классов - целиком видимые, целиком невидимые и пересекающие окно. Очевидно, что целесообразно возможно более рано, без выполнения большого объема вычислений принять решение об видимости целиком или отбрасывании. По способу выбора простого решения об отбрасывании невидимого отрезка целиком или принятия его существует два основных типа алгоритмов отсечения - алгоритмы, использующие кодирование концов отрезка или всего отрезка и алгоритмы, использующие параметрическое представление отсекаемых отрезков и окна отсечения. Представители первого типа алгоритмов - алгоритм Коэна-Сазерленда (Cohen-Sutherland, CS-алгоритм) и FC-алгоритм (Fast Clipping - алгоритм). Представители алгоритмов второго типа - алгоритм Кируса-Бека (Curus-Beck, CB - алгоритм) и более поздний алгоритм Лианга-Барски (Liang-Barsky, LB-алгоритм).

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

Вначале мы рассмотрим алгоритм Коэна-Сазерленда, являющийся стандартом де-факто алгоритма отсечения линий и обладающий одним из лучших быстродействий при компактной реализации. После рассмотрим алгоритм Кируса-Бека, который использует параметрическое представление и позволяет отсекать произвольным выпуклым окном.

2.2 Двумерный алгоритм Коэна-Сазерленда

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

- большинство примитивов содержится целиком в большом окне,

- большинство примитивов лежит целиком вне относительно маленького окна.

Для решения задачи определения принадлежности отрезка полю вывода используется следующий метод. Пространство разбивается на 9 областей, каждая из областей кодируется бинарным 4-х битным кодом рис. 2.7.2.

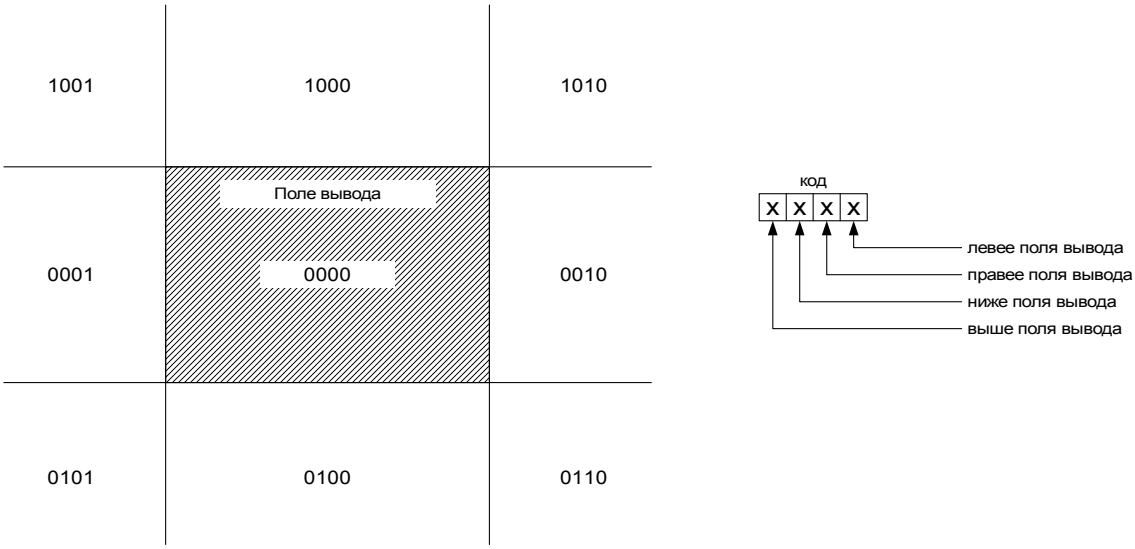


Рис. 2.1 Кодирование пространства для алгоритма Сазерленда-Коэна

Для каждого отрезка рассчитываются коды концов ($К_1$, $К_2$) затем производится экспресс анализ:

- Если $K_1 \wedge K_2 \neq 0$, тогда отрезок лежит вне поля вывода – отрезок отбрасывается
- Если $K_1 = K_2 = 0$, тогда отрезок полностью лежит внутри поля вывода – отсечение не нужно, отрезок полностью прорисовывается
- Если $K_1 \wedge K_2 = 0$, отрезок может частично лежать внутри поля вывода – необходимо отсечение по полю вывода.

Когда $K_1 \wedge K_2 = 0$ необходимо отсекать отрезок по границам поля вывода, отсечение происходит последовательно по всем сторонам рис. 2.2.

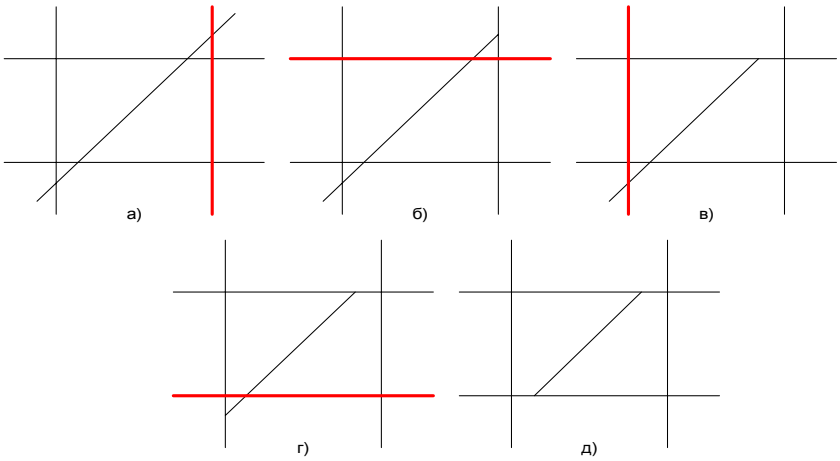


Рис. 2.2. Отсечение отрезка по прямоугольной области

На рис.2.2, жирным выделено ребро по которому происходит отсечение. Также надо отметить, что точки лежащие на границе поля вывода принадлежат полю вывода.

На каждом шаге отсечения вычисляются новые координаты одной из точек, найдем формулы для вычисления новых координат.

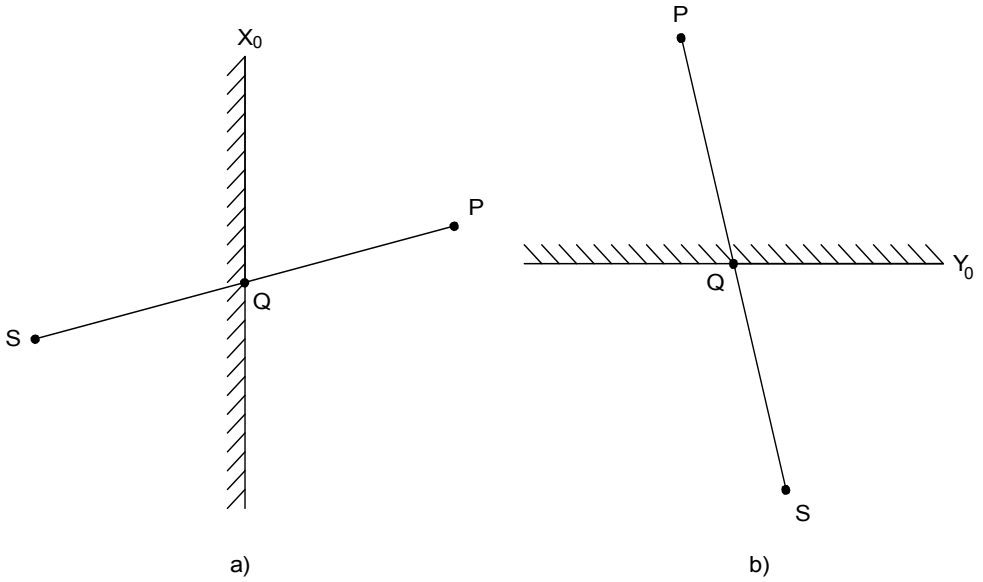


Рис.2.3. Вычисление координат точек пересечения

Формулы для расчета новых координат:

а)

$$Q(x_0, y_Q, V_Q)$$

$$y_Q = y_S + \frac{x_0 - x_S}{x_p - x_S} \cdot (y_P - y_S)$$

$$V_q = V_S + \frac{x_0 - x_S}{x_p - x_S} \cdot (V_P - V_S)$$

;

б)

$$Q(x_Q, y_0, V_Q)$$

$$x_Q = x_S + \frac{y_0 - y_S}{y_p - y_S} \cdot (x_P - x_S)$$

$$V_q = V_S + \frac{y_0 - y_S}{y_p - y_S} \cdot (V_P - V_S)$$

2.3 Двумерный алгоритм Кируса-Бека

Рассмотренный выше алгоритм, проводил отсечение по прямоугольному окну, стороны которого параллельны осям координат. Это, конечно, наиболее частый случай отсечения. Однако, во многих случаях требуется отсечение по произвольному многоугольнику, например, в алгоритмах удаления невидимых частей сцены. В этом случае наиболее удобно использование параметрического представления линий, не зависящего от выбора системы координат.

Продолжим каждую из четырех границ окна до бесконечных прямых. Каждая из таких прямых делит плоскость на 2 области. Назовем "видимой частью" ту, в которой находится окно отсечения, как это показано на рис. 2.4. Видимой части соответствует внутренняя сторона линии границы. Невидимой части плоскости соответствует внешняя сторона линии границы.

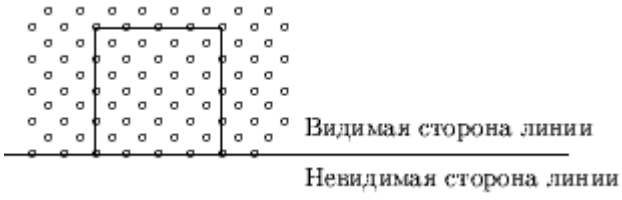


Рис. 2.4. Видимая часть линии границы

Таким образом, окно отсечения может быть определено как область, которая находится на внутренней стороне всех линий границ.

Отсекаемый отрезок прямой может быть преобразован в параметрическое представление следующим образом. Пусть конечные точки отрезка есть V_0 и V_1 с координатами (x0,y0) и (x1,y1), соответственно. Тогда параметрическое представление линии может быть задано следующим образом:

$$x = x0 + dx \cdot t; \qquad y = y0 + dy \cdot t,$$

$$\text{где} \quad dx = x1 - x0; \qquad dy = y1 - y0.$$

Или в общем виде для отрезка, заданного точками V_0 и V_1:

$$V(t) = V_0 + (V_1 - V_0) \cdot t$$

Для точек V_0 и V_1 параметр t равен 0 и 1, соответственно. Меняя t от 0 до 1 перемещаемся по отрезку V_0V_1 от точки V_0 к точке V_1. Изменяя t в интервале от $-\infty$ до $+\infty$, получаем бесконечную (далее удлиненную) прямую, ориентация которой - от точки V_0 к точке V_1.

Для выполнения отсечения в параметрическом представлении необходимо иметь способ определения ориентации удлиненной линии, содержащей отсекаемый отрезок, относительно линии границы - с внешней стороны на внутреннюю или с внутренней на внешнюю, а также иметь способ определения расположения точки, принадлежащей отрезку, относительно окна - вне, на границе, внутри.

Для этих целей в алгоритме Кируса-Бека , реализующем отсечение произвольным выпуклым многоугольником, используется вектор внутренней нормали к ребру окна.

Внутренней нормалью $\mathbf{N}_в$ в точке A к стороне окна называется нормаль, направленная в сторону области, задаваемой окном отсечения.

Рассмотрим основные идеи алгоритма Кируса-Бека.

Так как многоугольник предполагается выпуклым, то может быть только две точки пересечения отрезка с окном. Поэтому надо найти два значения параметра t, соответствующие начальной и конечной точкам видимой части отрезка.

Пусть $\mathbf{N}_i$ - внутренняя нормаль к i-й граничной линии окна, а $\mathbf{P} = \mathbf{V_1} - \mathbf{V_0}$ - вектор, определяющий ориентацию отсекаемого отрезка, тогда ориентация отрезка относительно i-й стороны окна определяется знаком скалярного произведения $P_i = \mathbf{N}_i \cdot \mathbf{V}$, равного произведению длин векторов на косинус наименьшего угла, требуемого для поворота вектора $\mathbf{N}_i$ до совпадения по направлению с вектором $\mathbf{V}$:

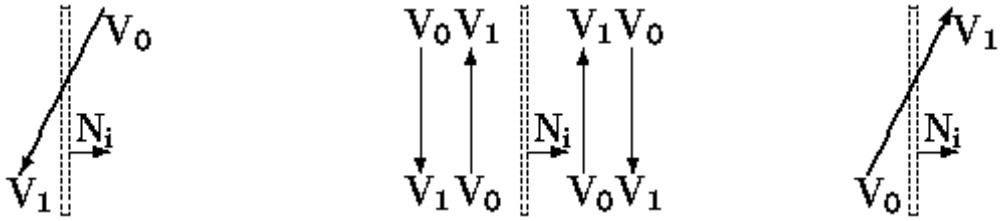
$$P_i = \mathbf{N}_i \cdot \mathbf{P} = \mathbf{N}_i \cdot (\mathbf{V}_1 - \mathbf{V}_0). \qquad (2.1)$$

При $P_i < 0$ отсекаемый отрезок направлен с внутренней на внешнюю стороны i-й граничной линии окна (см. рис. 2.5а).

При $P_i = 0$ точки V_0 и V_1 либо совпадают, либо отсекаемый отрезок

параллелен i-й граничной линии окна (см. рис. 2.5б). (2.2)

При $P_i > 0$ отсекаемый отрезок направлен с внешней на внутреннюю сторону i-й граничной линии окна (см. рис. 2.5в).



а) Изнутри наружу б) Параллельно границе в) Снаружи внутрь

Рис. 2.5. Ориентация отсекаемого отрезка относительно окна

Для определения расположения точки относительно окна вспомним параметрическое представление отсекаемого отрезка:

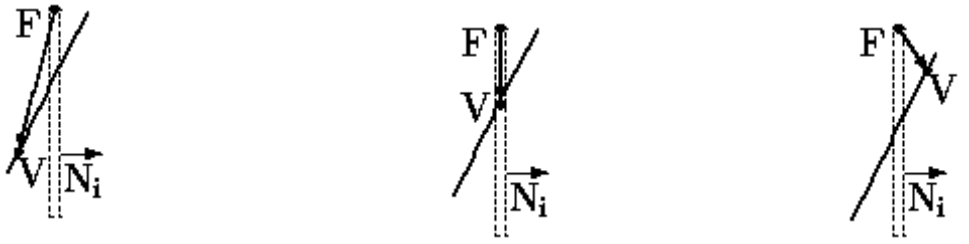
$$V(t) = V_0 + (V_1 - V_0)\cdot t; \qquad 0 \geq t \geq 1. \qquad (2.3)$$

Рассмотрим теперь скалярное произведение внутренней нормали $\mathbf{N}_i$ к i-й границе на вектор $\mathbf{Q}(t) = \mathbf{V}(t) - \mathbf{F}_i$, начинающийся в начальной точке ребра окна и заканчивающийся в некоторой точке V(t) удлиненной линии.

$$Q_i = \boldsymbol{N}_i \cdot \boldsymbol{Q} = \boldsymbol{N}_i \cdot [\boldsymbol{V}(t) - \boldsymbol{F}_i] \quad для \quad i = 1,2,3 \ \ldots \qquad (2.4)$$

Аналогично предыдущему имеем (рис.2.6):

При $Q_i < 0$ точка **V**(t) лежит с внешней стороны границы
При $Q_i = 0$ точка я**V**(t) лежит на самой границе
При $Q_i > 0$ точка **V**(t) лежит с внутренней стороны границы



а) Точка V вне б) Точка V на границе в) Точка V внутри

Рис. 2.6. Расположение точки относительно окна

Подставляя в (2.4) параметрическое представление (2.3), получим условие пересечения отрезка с границей окна:

$$\boldsymbol{N}_i \cdot [\ \boldsymbol{V}_0 + (\boldsymbol{V}_1 - \boldsymbol{V}_0)\cdot t - \boldsymbol{F}_i\ ] = 0 \qquad (2.5)$$

Раскрывая скобки, получим:

$$N_i \cdot [\ V_0 \ - \ F_i \ ] \ + \ N_i \cdot [\ V_1 \ - \ V_0 \ ] \cdot t \ = \ 0. \qquad (2.6)$$

Используя (2.1) и (2.4) перепишем (2.5):

$$(\ N_i \ \cdot \ P \ ) \cdot t \ + \ N_i \ \cdot \ Q \ = \ P_i \ \cdot \ t \ + \ Q_i. \qquad (2.7)$$

Разрешая (2.6) относительно t, получим:

$$t \ = \ - \frac{Q_i}{P_i} \ = \ - \frac{N_i \cdot Q}{N_i \cdot P} \quad \text{при} \quad P_i \neq 0, \quad i \ = \ 1,2,3,\ldots \qquad (2.8)$$

Это уравнение и используется для вычисления значений параметров, соответствующих начальной и конечной точкам видимой части отрезка.

Как следует из (2.2), P_i равно нулю если отрезок либо вырожден в точку, либо параллелен границе. В этом случае следует проанализировать знак Q_i и принять или не принять решение об отбрасывании отрезка целиком в соответствии с условиями (2.2).

Если же P_i не равно 0, то уравнение (2.8) используется для вычисления значений параметров t, соответствующих точкам пересечений удлиненной линии с линиями границ.

Алгоритм построен следующим образом:

Искомые значения параметров t_0 и t_1 точек пересечения инициализируются значениями 0 и 1, соответствующими началу и концу отсекаемого отрезка.

Затем в цикле для каждой i-й стороны окна отсечения вычисляются значения скалярных произведений, входящих в (2.7).

Если очередное P_i равно 0, то отсекаемый отрезок либо вырожден в точку, либо параллелен i-й стороне окна. При этом достаточно проанализировать знак Q_i. Если $Q_i < 0$, то отрезок вне окна и отсечение закончено иначе рассматривается следующая сторона окна.

Если же P_i не равно 0, то по (2.8) можно вычислить значение параметра t для точки пересечения отсекаемого отрезка с i-й границей. Так как отрезок V_0V_1 соответствует диапазону $0 \le t \le 1$, то все решения, выходящие за данный диапазон следует отбросить. Выбор оставшихся решений определяется знаком P_i.

Если $P_i < 0$, т.е. удлиненная линия направлена с внутренней на внешнюю стороны граничной линии, то ищутся значения параметра для конечной точки видимой части отрезка. В этом случае определяется минимальное значение из всех получаемых решений. Оно даст значение параметра t_1 для конечной точки отсеченного отрезка. Если текущее полученное значение t_1 окажется меньше, чем t_0, то отрезок отбрасывается, так как нарушено условие $t_0 \ \le \ t_1$.

Если же $P_i \ > \ 0$, т.е. удлиненная линия направлена с внешней на внутреннюю стороны граничной линии, то ищутся значения параметра для начальной точки видимой части отрезка. В этом случае определяется максимальное значение из всех получаемых решений. Оно даст значение параметра t_0 для начальной точки отсеченного отрезка. Если текущее полученное значение t_0 окажется больше, чем t_1, то отрезок отбрасывается, так как нарушено условие $t_0 \ \le \ t_1$.

На заключительном этапе алгоритма значения t_0 и t_1 используются для вычисления координат точек пересечения отрезка с окном. При этом, если $t_0 = 0$, то начальная точка осталась V_0 и вычисления не нужны. Аналогично, если $t_1 = 1$, то конечная точка осталась V_1 и вычисления также не нужны.

*По вычисленному $Q(t_{пер})$ определяем, лежит ли точка пересечения внутри отсекающего отрезка. Для этого достаточно сравнить длины векторов **Q и S**. (**S**=F_1-F_0, где F_1,F_0 концы отсекающего отрезка)*

Все эти случаи пояснены на блок-схеме, представленной на рис. 2.7.

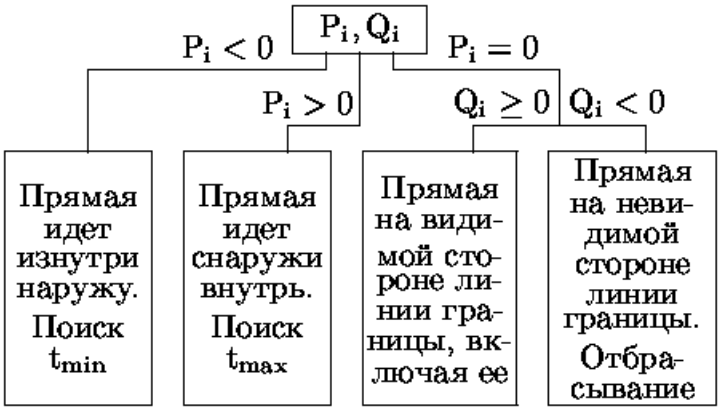


Рис. 2.7. Структурная схема алгоритма Кируса-Бека

Вычисления значений параметров t_0 и t_1 выполняются в соответствии с выражениями (2.9).

$$t_0 \geq \max\left(\{-Q_i/P_i \mid P_i > 0,\ i = 1,2,\ldots\} \cup \{0\}\right),$$
$$t_1 \leq \min\left(\{-Q_i/P_i \mid P_i < 0,\ i = 1,2,\ldots\} \cup \{1\}\right). \qquad (2.9)$$

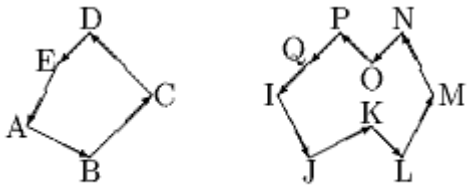
2.4 Проверка выпуклости и определение нормалей

Как видно из описания, алгоритм Кируса-Бека отсекает только по выпуклому окну. Кроме этого требуются значения внутренних нормалей к сторонам окна. Естественно выполнить эти вычисления в момент задания окна, так как следует ожидать, что одним окном будет отсекаться достаточно много отрезков.

2.4.1 Алгоритм с использованием векторных произведений

Проверка на выпуклость может производиться анализом знаков векторных произведений смежных ребер (рис. 2.8).

$$[A] \times [B] = A \cdot B \sin(\angle AB)$$



а) выпуклый б) невыпуклый

Рис. 2.8 Проверка выпуклости и определение нормалей

Если знак векторного произведения равен 0, то вершина вырождена, т.е. смежные ребра лежат на одной прямой (рис. 2.8 б, вершина Q).

Если все знаки равны 0, то многоугольник отсечения вырождается в отрезок.

Если же векторные произведения имеют разные знаки, то многоугольник отсечения невыпуклый (рис. 2.8 б).

Если все знаки неотрицательные, то многоугольник выпуклый, причем обход вершин выполняется против часовой стрелки (рис. 2.8 а), т.е. внутренние нормали ориентированы влево от контура. Следовательно, вектор внутреннего перпендикуляра к стороне может быть получен поворотом ребра на $+90^{\circ}$ (в реализации алгоритма вычисления нормалей на самом деле вычисляется не нормаль к стороне, а перпендикуляр, так как при вычислении значения t по соотношению (2.6) длина не важна).

Если все знаки неположительные, то многоугольник выпуклый, причем обход вершин выполняется по часовой стрелке, т.е. внутренние нормали ориентированы вправо от контура. Следовательно, вектор внутреннего перпендикуляра к стороне может быть получен поворотом ребра на -90°.

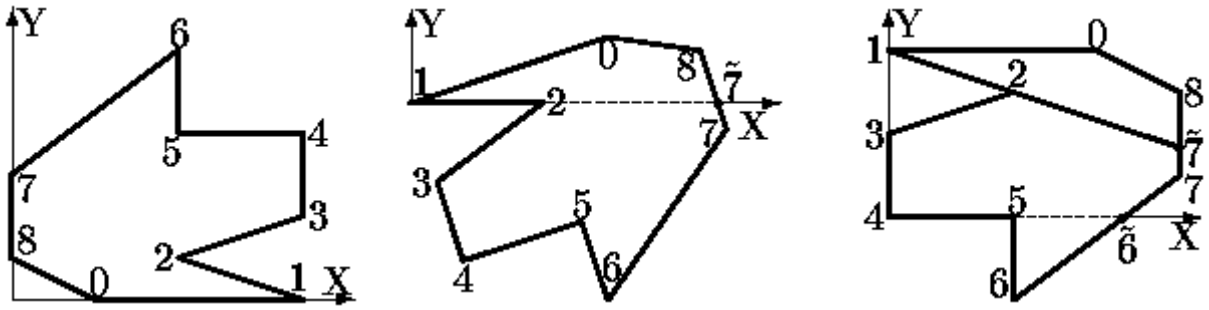
2.4.2 Разбиение невыпуклых многоугольников

Одновременное проведение операций проверки на выпуклость и разбиение простого невыпуклого многоугольника на выпуклые обеспечивается методом переноса и поворотов окна.

Алгоритм метода при обходе вершин многоугольника против часовой стрелки состоит в следующем:

1. Для каждой i-й вершины многоугольник сдвигается для переноса упомянутой вершины в начало координат.
2. Многоугольник поворачивается против часовой стрелки для совмещения (i+1)-й вершины с положительной полуосью X. Вектор внутреннего перпендикуляра к ребру, образованному вершинами i-й и (i+1)-й, вычисляется поворотом ребра на -90° против часовой стрелки.
3. Анализируется знак Y-координаты (i+2)-й вершины.
   Если $Y_{i+2} \geq 0$, то в (i+1)-й вершине выпуклость.
   Если $Y_{i+2} \geq 0$, то в (i+1)-й вершине невыпуклость.
   Если имеется невыпуклость, то многоугольник разрезается на два вдоль положительной полуоси X.
   Для этого вычисляется пересечение положительной полуоси X с первой из сторон.
   Формируются два новых многоугольника: первый многоугольник - вершины с (i+1)-й до точки пересечения - вершины 2, 3, 4, 6, 7, **7** на рис. 2.9 б);
   второй многоугольник - все остальные вершины - вершины **7**, 8, 0, 1 на рис. 2.9 б)

Так как вновь полученные многоугольники могут в свою очередь оказаться невыпуклыми, алгоритм применяется к ним, пока все многоугольники не станут выпуклыми.



а) Исходное окно

б) Невыпуклость после вершины 2

в) Невыпуклость после вершины 5

Рис. 2.9. Проверка выпуклости и разбиение многоугольника

Повторное применение алгоритма в многоугольнику, образованному вершинами 2, 3, 4, 6, 7, [7\tilde], показано на рис. 0.2.18 в).

Данный алгоритм не обеспечивает минимальность числа вновь полученных выпуклых многоульников и некорректно работает если имеется самопересечение сторон, как это показано на рис. 0.2.19.

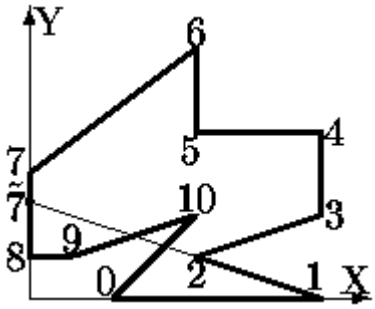


Рис. 2.10. Многоугольник с самопересечением сторон

2.5 Заливка многоугольника

В большинстве приложений используется одно из существенных достоинств растровых устройств - возможность заполнения областей экрана.

Существует две разновидности заполнения:

- первая, связанная как с интерактивной работой, так и с программным синтезом изображения, служит для заполнения внутренней части многоугольника, заданного координатами его вершин.

- вторая, связанная в первую очередь с интерактивной работой, служит для заливки области, которая либо очерчена границей с кодом пиксела, отличающимся от кодов любых пикселов внутри области, либо закрашена пикселами с заданным кодом;

В данном разделе рассмотрим алгоритм заполнения многоугольника. В следующем разделе будут рассмотрены алгоритмы заливки области.

Простейший способ заполнения многоугольника, заданного координатами вершин, заключается в определении принадлежит ли текущий пиксел внутренней части многоугольника. Если принадлежит, то пиксел заносится.

Определить принадлежность пиксела многоугольнику можно, например, подсчетом суммарного угла с вершиной на пикселе при обходе контура многоугольника. Если пиксел внутри, то угол будет равен 360°, если вне - 0° (рис.).

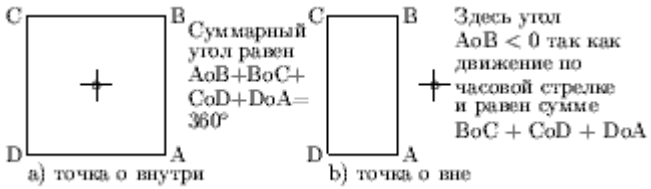


Рис. 2.11. Определение принадлежности пиксела многоугольнику

Вычисление принадлежности должно производиться для всех пикселов экрана и так как большинство пикселов скорее всего вне многоугольников, то данный способ слишком расточителен. Объем лишних вычислений в некоторых случаях можно сократить использованием прямоугольной оболочки - минимального прямоугольника, объемлющего интересующий объект, но все равно вычислений будет много. Другой метод определения принадлежности точки внутренней части многоугольника будет рассмотрен ниже при изучении отсечения отрезков по алгоритму Кируса-Бека.

2.5.1 Построчное заполнение

Реально используются алгоритмы построчного заполнения, основанные на том, что соседние пикселы в строке скорее всего одинаковы и меняются только там где строка пересекается с ребром многоугольника. Это называется когерентностью растровых строк (строки сканирования Y_i, Y_{i+1}, Y_{i+2} на рис.). При этом достаточно определить X-координаты пересечений строк сканирования с ребрами. Пары отсортированных точек пересечения задают интервалы заливки.

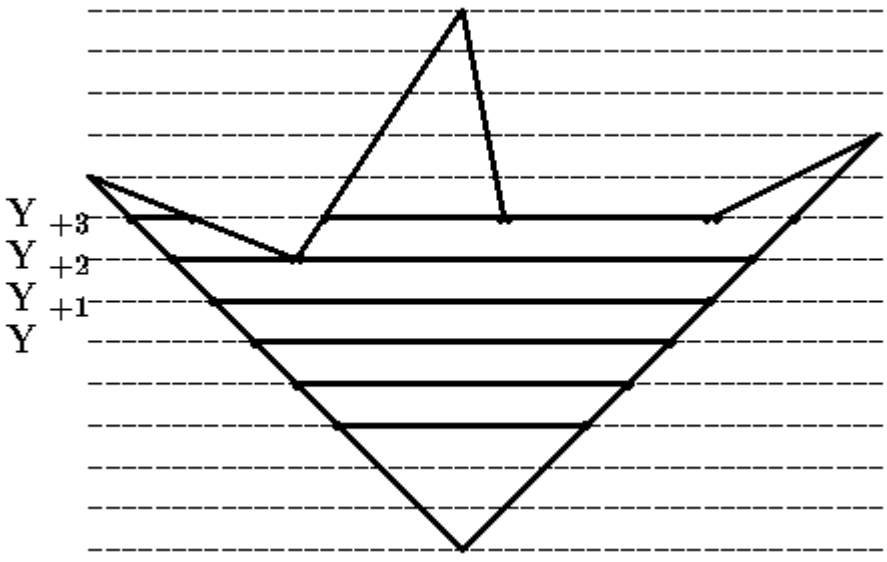


Рис. 2.12. Построчная закраска многоугольника

Кроме того, если какие-либо ребра пересекались i-й строкой, то они скорее всего будут пересекаться также и строкой i+1. (строки сканирования Y_i и Y_{i+1} на рис. 2.12). Это называется когерентностью ребер. При переходе к новой строке легко вычислить новую X-координату точки пересечения ребра, используя X-координату старой точки пересечения и тангенс угла наклона ребра:

$$X_{i+1} = X_i + 1/k$$

(тангенс угла наклона ребра k = dy/dx, так как dy = 1, то 1/k = dx).

Смена же количества интервалов заливки происходит только тогда, когда в строке сканирования появляется вершина.

Учет когерентности строк и ребер позволяет построить для заполнения многоугольников различные высокоэффективные алгоритмы построчного сканирования. Для каждой строки сканирования рассматриваются только те ребра, которые пересекают строку. Они задаются списком активных ребер (САР). При переходе к следующей строке для пересекаемых ребер перевычисляются X-координаты пересечений. При появлении в строке сканирования вершин производится перестройка САР. Ребра, которые перестали пересекаться, удаляются из САР, а все новые ребра, пересекаемые строкой заносятся в него.

Общая схема алгоритма, динамически формирующего список активных ребер и заполняющего многоугольник снизу-вверх, следующая:

1. Подготовить служебные целочисленные массивы Y-координат вершин и номеров вершин.
2. Совместно отсортировать Y-координаты по возрастанию и массив номеров вершин для того, чтобы можно было определить исходный номер вершины.
3. Определить пределы заполнения по оси Y - Y_min и Y_max. Стартуя с текущим значением Y_tek = Y_min, исполнять пункты 4-9 до завершения раскраски.
4. Определить число вершин, расположенных на строке Y_tek - текущей строке сканирования.
5. Если вершины есть, то для каждой из вершин дополнить список активных ребер, используя информацию о соседних вершинах.
   Для каждого ребра в список активных ребер заносятся:
   - максимальное значение Y-координаты ребра,
   - приращение X-координаты при увеличении Y на 1,
   - начальное значение X-координаты.

   Если обнаруживаются горизонтальные ребра, то они просто закрашиваются и информация о них в список активных ребер не заносится.
   Если после этого обнаруживается, что список активных ребер пуст, то заполнение закончено.

6. По списку активных ребер определяется Y_след - Y-координата ближайшей вершины. (Вплоть до Y_след можно не заботиться о модификации САР а только менять X-координаты пересечений строки сканирования с активными ребрами).
7. В цикле от Y_tek до Y_след:
   - выбрать из списка активных ребер и отсортировать X-координаты пересечений активных ребер со строкой сканирования;
   - определить интервалы и выполнить закраску;
   - перевычислить координаты пересечений для следующей строки сканирования.
8. Проверить не достигли ли максимальной Y-координаты. Если достигли, то заливка закончена, иначе выполнить пункт .

Очистить список активных ребер от ребер, закончившихся на строке Y_след и

Сортировка методом распределяющего подсчета

Понятно, что одна из важнейших работ в алгоритме построчного сканирования - сортировка. В связи с заведомо ограниченной разрешающей способностью растровых дисплеев (не более 2048) иногда целесообразно использовать чрезвычайно эффективный алгоритм сортировки методом распределяющего подсчета.

Для рассмотрения алгоритма предположим, что надо отсортировать числа, заданные в массиве с именем "Исходный_массив"; количество сортируемых чисел задается скаляром "Кол-во_чисел"; сортируемые числа J удовлетворяют условию:

$$0 \le J < \text{Max\_число}.$$

Для сортировки потребуются описания:

```
int  Max_число;         /* Верхняя граница значений */
int  *Повтор;           /* Длина этого массива = Max_число */
int  Кол_чисел;         /* Кол-во сортируемых чисел */
int  *Исходный_массив; /* Длина этого массива >= Кол_чисел */
int  *Результат;        /* Длина этого массива >= Кол_чисел */
int  ii,jj, kk;         /* Рабочие переменные */
```

1. Обнуляется служебный массив для подсчета числа повторений исходных кодов.

```
2.      for (ii=0; ii<Max_число; ++ii) Повтор[ii]= 0;
```

3. Сортируемый массив просматривается и вычисляется количество раз повторений каждого числа:

```
4.      for (ii= 0; ii < Кол_чисел; ++ii) {
5.         jj= Исходный_массив[ii];
6.         Повтор[jj]= Повтор[jj] + 1;
7.      }
```

8. Суммируется количество повторений каждого числа, так что значение Повтор[J] даст начальное расположение группы чисел, равных J, в отсортированном массиве:

```
9.      jj= 0;
10.     for (ii=0; ii<Max_число; ++ii) {
11.        jj= jj + Повтор[ii];
12.        Повтор[ii]= jj;
13.     }
```

14. Просматривается исходный массив и числа из него заносятся в массив результатов той же длины. Индекс занесения числа J в массив результатов равен значению J-го элемента массива Повтор. После занесения числа J значение Повтор[J] уменьшается на 1:

```
15.     for (ii= 0; ii < Кол_чисел; ++ii) {
16.        jj= Исходный_массив[ii];
17.        kk= Повтор[jj];
18.        Результат[kk]= jj;
19.        Повтор[jj]= Повтор[jj] - 1;
20.     }
```

2.5.2 Заливка области с затравкой

Как уже отмечалось, для приложений, связанных в основном с интерактивной работой, используются алгоритмы заполнения области с затравкой.

При этом тем или иным образом задается заливаемая (перекрашиваемая) область, код пиксела, которым будет выполняться заливка и начальная точка в области, начиная с которой начнется заливка.

По способу задания области делятся на два типа:

- гранично-определенные, задаваемые своей (замкнутой) границей такой, что коды пикселов границы отличны от кодов внутренней, перекрашиваемой части области. На коды пикселы внутренней части области налагаются два условия - они должны быть отличны от кода пикселов границы и кода пиксела перекраски. Если внутри гранично-определенной области имеется еще одна граница, нарисованная пикселами с тем же кодом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;

• внутренне-определенные, нарисованные одним определенным кодом пиксела. При заливке этот код заменяется на новый код закраски.

В этом состоит основное отличие заливки области с затравкой от заполнения многоугольника. В последнем случае мы сразу имеем всю информацию о предельных размерах части экрана, занятой многоугольником. Поэтому определение принадлежности пиксела многоугольнику базируется на быстро работающих алгоритмах, использующих когерентность строк и ребер (см. предыдущий раздел). В алгоритмах же заливки области с затравкой нам вначале надо прочитать пиксел, затем определить принадлежит ли он области и если принадлежит, то перекрасить.

Заливаемая область или ее граница - некоторое связное множество пикселов. По способам доступа к соседним пикселам области делятся на 4-х и 8-ми связные. В 4-х связных областях доступ к соседним пикселам осуществляется по четырем направлениям - горизонтально влево и вправо и в вертикально вверх и вниз. В 8-ми связных областях к этим направлениям добавляются еще 4 диагональных. Используя связность мы может, двигаясь от точки затравки, достичь и закрасить все пикселы области.

Важно отметить, что для 4-х связной прямоугольной области граница 8-ми связна (рис. а) и наоборот у 8-ми связной области граница 4-х связна (см. рис. б). Поэтому заполнение 4-х связной области 8-ми связным алгоритмом может привести к "просачиванию" через границу и заливке пикселов в примыкающей области.

В общем, 4-х связную область мы можем заполнить как 4-х, так и 8-ми связным алгоритмом. Обратное же неверно. Так область на рис. а мы можем заполнить любым алгоритмом, а область на рис. б, состоящую из двух примыкающих 4-х связных областей можно заполнить только 8-ми связным алгоритмом.

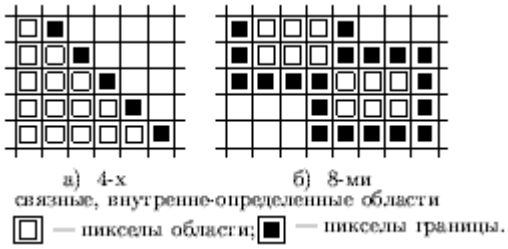


Рис. 0.5.1: Связность областей и их границ

С использованием связности областей и стека можно построить простые алгоритмы закраски как внутренне, так и гранично-определенной области. В [] рассматриваются совсем короткие рекурсивные подпрограммы заливки. В [] - несколько более длинные итеративные подпрограммы.

Простой алгоритм заливки с затравкой

Рассмотрим простой алгоритм заливки гранично-определенной 4-х связной области. Ниже приведена рекурсивная реализация подпрограммы заливки 4-х связной гранично-определенной области:

```
void V_FAB4R (grn_pix, new_pix, x_isx, y_isx)
int grn_pix, new_pix, x_isx, y_isx;
{
if (getpixel (x_isx, y_isx)  grn_pix &&
getpixel (x_isx, y_isx)  new_pix)
{
```

```
putpixel (x_isx, y_isx, new_pix);
V_FAB4R (grn_pix, new_pix, x_isx+1, y_isx);
V_FAB4R (grn_pix, new_pix, x_isx, y_isx+1);
V_FAB4R (grn_pix, new_pix, x_isx-1, y_isx);
V_FAB4R (grn_pix, new_pix, x_isx, y_isx-1);
}
} /* V_FAB4R */
```

Заливка выполняется следующим образом:

• определяется является ли пиксель граничным или уже закрашенным,

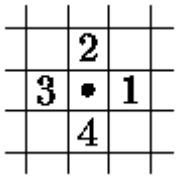
• если нет, то пиксель перекрашивается, затем проверяются и если надо перекрашиваются 4 соседних пикселя.

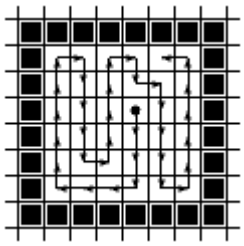
Понятно, что несмотря на простоту и изящество программы, рекурсивная реализация проигрывает итеративной в том, что требуется много памяти для упрятывания вложенных вызовов.

В [Шикин??] приведен итеративный алгоритм закраски 4-х связной гранично-определенной области. Логика работы алгоритма следующая:

- Поместить координаты затравки в стек
- Пока стек не пуст
- Извлечь координаты пиксела из стека.
- Перекрасить пиксел.
- Для всех четырех соседних пикселов проверить является ли он граничным или уже перекрашен.
- Если нет, то занести его координаты в стек.

На рис. а) показан выбранный порядок перебора соседних пикселов, а на рис. б) соответствующий ему порядок закраски простой гранично-определенной области.





а) Порядок перебора соседних пикселов

б) Порядок заливки области

Рис. 2.13. Заливка 4-х связной области итеративным алгоритмом

Ясно, что такой алгоритм экономнее, так как в стек надо упрятывать только координаты.

Рассмотренный алгоритм легко модифицировать для работы с 8-ми связными гранично-определенными областями или же для работы с внутренне-определенными.

Заметим, что рекурсивный алгоритм неэкономичен: при стандартном окне стека в 64 K с помощью рекурсивной программы можно закрасить квадратик не более чем 57×57 пикселей. Итеративная же программа при тех же условиях позволяет закрасить прямоугольник размером 110×110 истратив на массив координат 16382 байта.

Как уже отмечалось, очевидный недостаток алгоритмов непосредственно использующих связность закрашиваемой области - большие затраты памяти на стек, так как на каждый закрашенный пиксел в стеке по максимуму будет занесена информация о еще трех соседних. Кроме того, информация о некоторых пикселах может записываться в стек многократно. Это приведет не только к перерасходу памяти, но и потере быстродействия за счет многократной раскраски одного и того же пиксела. Значительно более экономен далее рассмотренный построчный алгоритм заливки.

Построчный алгоритм заливки с затравкой

Использует свойство пространственной когерентности:

- пиксели в строке меняются только на границах;
- при перемещении к следующей строке размер заливаемой строки скорее всего или неизменен или меняется на 1 пиксель.

Таким образом, на каждый закрашиваемый фрагмент строки в стеке хранятся координаты только одного начального пиксела [], что приводит к существенному уменьшению размера стека.

Последовательность работы алгоритма для гранично-определенной области следующая:

1. Координата затравки помещается в стек, затем до исчерпания стека выполняются пункты 2-4.
2. Координата очередной затравки извлекается из стека и выполняется максимально возможное закрашивание вправо и влево по строке с затравкой, т.е. пока не попадется граничный пиксел. Пусть это Хлев и Хправ, соответственно.
3. Анализируется строка ниже закрашиваемой в пределах от Хлев до Хправ и в ней находятся крайние правые пикселы всех незакрашенных фрагментов. Их координаты заносятся в стек.
4. То же самое проделывается для строки выше закрашиваемой.

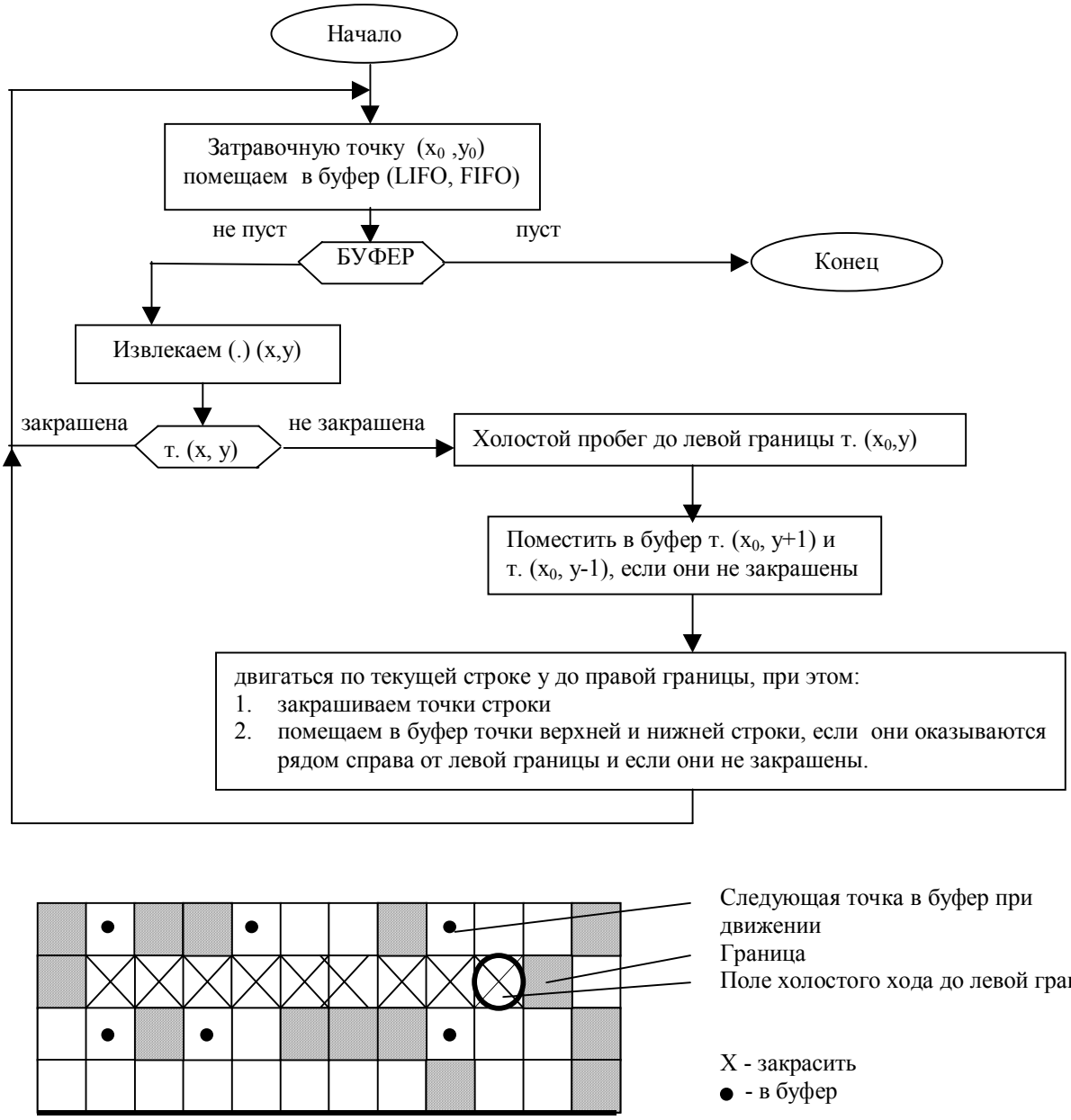


Рис. 2.14. Структурная схема и иллюстрация алгоритма построчной заливки с затравкой.

3 Аффинные преобразхования

3.1 Афинные преобразования на плоскости

В компьютерной графике все, что относится к двумерному случаю принято обозначать символом (2D) (2-dimention).
Допустим, что на плоскости введена прямолинейная координатная система. Тогда каждой точке M ставится в соответствие упорядоченная пара чисел (x, y) ее координат (рис. 1). Вводя на плоскости еще одну прямолинейную систему координат, мы ставим в соответствие той же точке M другую пару чисел – (x*, y*).

Рис. 1

Переход от одной прямолинейной координатной системы на плоскости к другой описывается следующими соотношениями:

$x^* = ax + by + l$, (2.1)

$y^* = gx + by + m$, (2.2)

где a, b, g, l, m -- произвольные числа, связанные неравенством:

$$\begin{vmatrix} \alpha & \beta \\ \gamma & \delta \end{vmatrix} \neq 0$$

Формулы (2.1) и (2.2) можно рассматривать двояко: либо сохраняется точка и изменяется координатная система (рис. 2) – в этом случае произвольная точка M остается той же, изменяются лишь ее координаты (x, y) | (x*, y*), либо изменяется точка и сохраняется координатная система (рис. 3) – в этом случае формулы (2.1) и (2.2) задают отображение, переводящее произвольную точку M (x, y) в точку M* (x*, y*), координаты которой определены в той же координатной системе.

Рис. 2

В дальнейшем, формулы (2.1) и (2.2) будут рассматриваться как правило, согласно которому в заданной системе прямолинейных координат преобразуются точки плоскости.

В аффинных преобразованиях плоскости особую роль играют несколько вжных частных случаев, имеющих хорошо прослеживаемыегеометрические характеристики. При исследовании геометрического смысла числовых коэффицентов в формулах (2.1) и (2.2) для этих случаев удобно считать, что заданная система координат является прямоугольной декартовой.

1. Поворот вокруг начальной точки на угол j (рис. 4) описывается формулами:

$x^* = x \cos \varphi - y \sin \varphi$, (2.3)
$y^* = x \sin \varphi - y \cos \varphi$. (2.4)

2. Растяжение (сжатие) вдоль координатных осей можно задать так:

$x^* = \alpha x$, (2.5)
$y^* = \delta y$, (2.6)
$\alpha > 0, \delta > 0$. (2.7)

Растяжение (сжатие) вдоль оси абсцисс обеспечивается при условии, что $a > 1$ ($a < 1$). На рис.5 $a = d > 1$.

3. Отражение (относительно оси абсцисс) (рис. 6) задается при помощи формул:

$x^* = x$, (2.8)
$y^* = -y$. (2.9)

4. На рис. 7 вектор переноса MM* имеет координаты λ, μ. Перенос обеспечивает соотношения:

$x^* = x + \lambda$, (2.10)
$y^* = y + \mu$. (2.11)

Выбор этих четырех частных случаев определяется двумя обстоятельствами.

1. Каждое из приведенных выше преобразований имеет простой и наглядный геометрический смысл (геометрическим смыслом наделены и постоянные числа, входящие в приведенные формулы).

2. Как известно из курса аналитической геометрии, любое преобразование вида (2.1) всегда можно представить как последовательное исполнение (суперпозицию) простейших преобразований вида 1 – 4 (или части этих преобразований).

Таким образом, справедливо следующее важное свойство аффинных преобразований плоскости: любое отображение вида (2.1) можно описать при помощи отображений, задаваемых формулами (2.3) – (2.11).

Для эффективного использования этих известных формул в задачах компьютерной графики более удобной является их матричная запись. Матрицы, соответствующие случаям 1 – 3, строятся легко и имеют соответственно следующий вид:

$\begin{matrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{matrix}$ $\quad$ $\begin{matrix} \alpha & 0 \\ 0 & \delta \end{matrix}$ $\quad$ $\begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$

Однако, желательно охватить матричным подходом все 4 простейших преобразования (включая перенос), а, значит, и общее аффинное преобразование. Этого можно достичь перейдя к описанию произвольной точки плоскости не парой, а упорядоченной тройкой чисел.

Однородные координаты точки

Пусть M – произвольная точка плоскости с координатами x и y, вычисленными относительно заданной прямолинейной координатной системы. Однородными координатами этой точки называется любая тройка одновременно не равных нулю чисел x1, x2, x3, связанных с заданными числами x и y следующими соотношениями:

$x1 / x3 = x$, $x2 / x3 = y$ (3.1)

При решении задач компьютерной графики однородные координаты обычно вводятся так: произвольной точке М (x, y) плоскости ставится в соответствие точка МЭ (x, y, 1) в пространстве.

Необходимо заметить, что произвольная точка на прямой, соединяющей начало координат, точку О (0, 0, 0), с точкой МЭ (x, y, 1),может быть задана тройкой чисел вида (hx, hy, h).

Будем считать, что h = 0. Вектор с координатами hx, hy, h является направляющим вектором прямой, соединяющей точки О (0, 0, 0) и МЭ (x, y, 1). Эта прямая пересекает плоскость z = 1 в точке (x, y, 1), которая однозначно определяет точку (x, y) координатной плоскости xy.

Тем самым между произвольной точкой с координатами (x, y) и множеством троек чисел вида (hx, hy, h), h = 0, устанавливается взаимно однозначное соответствие, позволяющее считать числа hx, hy, h новыми координатами этой точки.

Широко используемые в проективной геометрии однородные координаты позволяют эффективно описывать так называемые несобственные элементы (по существу, те, которыми проектная плоскость отличается от привычной евклидовой плоскости).

В проективной геометрии для однородных координат принято следующее обозначение:

x : y : 1 (3.2)

или, более общо,

x1 : x2 : x3 (3.3)

(здесь непременно требуется, чтобы числа x1, x2, x3 одновременно в нуль не обращались).

Применение однородных координат оказывается удобным уже при решении простейших задач.

Рассмотрим, например, вопросы, связанные с изменением масштаба. Если устройство отображения работает только с целыми числами (или если необходимо работать только с целыми числами), то для произвольного значения h (например, h = 1) точку с однородными координатами (0.5, 0.1, 2.5) представить нельзя. Однако при разумном выборе h можно добиться того, чтобы координаты этой точки были целыми числами. В частности, при h = 10 для рассматриваемого примера имеем (5, 1, 25).

Рассмотрим другой случай. Чтобы результаты преобразования не приводили к арифметическому переполнению для точки с координатами (80000, 40000, 1000) можно взять, например, h = 0.001. В результате получим (80, 40, 1).

Приведенные примеры показывают полезность использования однородных координат при проведении расчетов. Однако основной целью введения однородных координат в компьютерной графике является их несомненное удобство в применении к геометрическим преобразованиям.

При помощи троек однородных координат и матриц третьего порядка можно описать любое аффинное преобразование плоскости.

Считая, h = 1, сравним две записи:

$$(x * y * 1) = (x\ y\ 1)\begin{matrix}\alpha & \gamma & 0\\ \beta & \delta & 0\\ \lambda & \mu & 1\end{matrix} \qquad (3.4)$$

Нетрудно заметить, что после перемножения выражений, стоящих в правой части последнего соотношения, мы получим формулы (2.1) и (2.2) и верное числовое равенство 1 = 1. Тем самым сравниваемые записи можно считать равносильными.

Элементы произвольной матрицы аффинного преобразования не несут в себе явно выраженного геометрического смысла. Поэтомучтобы реализовать то или иное отображение, то есть найти элементы соответствующей матрицы по заданному геометрическому описанию, необходимы специальные приемы. Обычно построение этой матрицы в соответствии со сложностью поставленной задачи и с описанными выше частными случаями разбивают на несколько этапов.

На каждом этапе пишется матрица, соответствующая тому или иному из выделенных выше случаев 1 – 4, обладающих хорошо выраженными геометрическими свойствами.

Выпишем соответствующие матрицы третьего порядка.

А. Матрица вращения (rotation)

$$[R] = \begin{matrix}\cos\varphi & \sin\varphi & 0\\ -\sin\varphi & \cos\varphi & 0\\ 0 & 0 & 1\end{matrix} \qquad (3.5)$$

Б. Матрица растяжения-сжатия (dilatation)

$$[D] = \begin{matrix}\alpha & 0 & 0\\ 0 & \delta & 0\\ 0 & 0 & 1\end{matrix} \qquad (3.6)$$

В. Матрица отражения (reflection)

$$[M] = \begin{matrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{matrix} \qquad (3.7)$$

Г. Матрица переноса (translation)

$$[T] = \begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ \lambda & \mu & 1\end{matrix} \qquad (3.8)$$

Рассмотрим примеры аффинных преобразований плоскости.

Пример 1. Построить матрицу поворота вокруг точки A (a, b) на угол φ (рис. 9).

1-й шаг. Перенос на вектор – A (-a, -b) для смещения центра поворота с началом координат;

$$[T_{-A}] = \begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ -a & -b & 1\end{matrix} \qquad (3.9)$$

матрица соответствующего преобразования.

2-й шаг. Поворот на угол φ;

$$[\,R_{\varphi}\,] = \begin{matrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{matrix} \qquad (3.10)$$

матрица соответствующего преобразования.

3-й шаг. Перенос на вектор A (a, b) для возвращения центра поворота в прежнее положение;

$$[\,T_A\,] = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{matrix} \qquad (3.11)$$

матрица соответствующего преобразования.

Перемножим матрицы в том же порядке, как они выписаны:

[T-A] [Rφ] [TA].

В результате получим, что искомое преобразование (в матричной записи) будет выглядеть следующим образом:

$$(x^*\ y^*\ 1) = (x\ y\ 1) \begin{matrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ -a\cos\varphi + b\sin\varphi + a & -a\sin\varphi - b\cos\varphi + b & 1 \end{matrix} \qquad (3.12)$$

Элементы полученной матрицы (особенно в последней строке) не так легко запомнить. В то же время каждая из трех перемножаемых матриц по геометрическому описанию соответствующего отображения легко строится.

Пример 2. Построить матрицу растяжения с коэффицентами растяжения αвдоль оси абсцисс и β вдоль оси ординат и с центром в точке A (a, b).

1-й шаг. Перенос на вектор –A (-a, -b) для совмещения центра растяжения с началом координат;

$$[\,T_{-A}\,] = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{matrix} \qquad (3.13)$$

матрица соответствующего преобразования.

2-й шаг. Растяжение вдоль координатных осей с коэффицентами α и β соответственно; матрица преобразования имеет вид

$$[\,D\,] = \begin{matrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{matrix} \qquad (3.14)$$

3-й шаг. Перенос на вектор A (a, b) для возвращения центра растяжения в прежнее положение; матрица соответствующего преобразования:

$$[\,T_A\,] = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{matrix} \qquad (3.15)$$

Перемножив матрицы в том же порядке

[T-A] [D] [TA], получим окончательно

$$(x^*\ y^*\ 1) = (x\ y\ 1) \begin{matrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ (1-\alpha)a & (1-\delta)b & 1 \end{matrix} \qquad (3.16)$$

Рассуждая подобным образом, то есть разбивая предложенное преобразование на этапы, поддерживаемые матрицами [R], [D], [M], [T], можно построить матрицу любого аффинного преобразования по его геометрическому описанию.

3.2 Аффинные преобразования в пространстве

Рассмотрим трехмерный случай (3D) (3-dimension) и сразу введем однородные координаты.

Потупая аналогично тому, как это было сделано в размерности два, заменим координатную тройку (x, y, z), задающую точку в пространстве, на четверку чисел

(x y z 1)

или, более общо, на четверку

(hx hy hz), h = 0.

Каждая точка пространства (кроме начальной точки O) может быть задана четверкой одновременно не равных нулю чисел; эта четверка чисел определена однозначно с точностью до общего множителя.

Предложенный переход к новому способу задания точек дает возможность воспользоваться матричной записью и в более сложных трехмерных задачах.

Любое аффинное преобразование в трехмерном пространстве может быть представленно в виде суперпозиции вращений, растяжений, отражений и переносов. Поэтому вполне уместно сначала подробно описать матрицы именно этих преобразований (ясно, что в данном случае порядок матриц должен быть равен четырем).

A. Матрицы вращения в пространстве.

Матрица вращения вокруг оси абсцисс на угол φ:

$$[\,Rx\,] = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Матрица вращения вокруг оси ординат на угол ψ:

$$[\,Ry\,] = \begin{matrix} \cos\psi & 0 & -\sin\psi & 0 \\ 0 & 1 & 0 & 1 \\ \sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Матрица вращения вокруг оси аппикат на угол χ:

$$[\,Rz\,] = \begin{matrix} \cos\chi & \sin\chi & 0 & 0 \\ -\sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Полезно обратить внимание на место знака « - » в каждой из трех приведенных матриц.

Б. Матрица растяжения-сжатия:

$$[\,D\,] = \begin{matrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

где

$\alpha > 0$ – коэффицент растяжения (сжатия) вдоль оси абсцисс;
$\beta > 0$ – коэффицент растяжения (сжатия) вдоль оси ординат;
$\gamma > 0$ – коэффицент растяжения (сжатия) вдоль оси аппликат.

В. Матрицы отражения

Матрица отражения относительно плоскости xy:

$$[\,Mz\,] = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \qquad (4.5)$$

Матрица отражения относительно плоскости yz:

$$[\,Mx\,] = \begin{matrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \qquad (4.6)$$

Матрица отражения относительно плоскости zx:

$$[My] = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \qquad (4.7)$$

Г. Матрица переноса (здесь (λ, μ, ν) – вектор переноса):

$$[T] = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & \mu & \nu & 1 \end{matrix} \qquad (4.8)$$

Как и в двумерном случае, все выписанные матрицы невырождены.

Приведем важный пример построения матрицы сложного преобразования по его геометрическому описанию.

Пример 3. Построить матрицу вращения на угол φ вокруг прямой L, проходящей через точку A (a, b, c) и имеющую направляющий вектор (l, m, n). Можно считать, что направляющий вектор прямой является единичным:

$l^2 + m^2 + n^2 = 1$

На рис. 10 схематично показано, матрицу какого преобразования требуется найти.

Рис. 10

Решение сформулированной задачи разбивается на несколько шагов. Опишем последовательно каждый из них.

1-й шаг. Перенос на вектор –A (-a, -b, -c) при помощи матрицы

$$[T] = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{matrix} \qquad (4.9)$$

В результате этого переноса мы добиваемся того, чтобы прямая L проходила через начало координат.

2-й шаг. Совмещение оси аппликатс прямой L двумя поворотами вокруг оси абсцисс и оси ординат.

1-й поворот – вокруг оси абсцисс на угол ψ (подлежащий определению). Чтобы найти этот угол, рассмотрим ортогональную проекцию L’ исходной прямой L на плоскость X = 0 (рис. 11).

Рис. 11

Направляющий вектор прямой L’ определяется просто – он равен (0, m, n).

Отсюда сразу же вытекает, что

$\cos \psi = n / d, \sin \psi = m / d,$ (4.10)

где $d = \sqrt{m^2 + n^2}$ (4.11)

Соответствующая матрица вращения имеет следующий вид:

$$[Rx] = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & n/d & m/d & 0 \\ 0 & -m/d & n/d & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \quad (4.12)$$

Под действием преобразования, описываемого этой матрицей, координаты вектора (l, m, n) изменятся. Подсчитав их, в результате получим

$(l, m, n, 1)[R_x] = (l, 0, d, 1).$

2-й поворот вокруг оси оси ординат на угол θ, определяемый соотношениями

$\cos \theta = l, \sin \theta = -d$ (4.14)

Соответствующая матрица вращения записывается в следующем виде:

$$[Ry] = \begin{matrix} l & 0 & d & 0 \\ 0 & 1 & 0 & 0 \\ -d & 0 & l & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

3-й шаг. Вращение вокруг прямой L на заданный уголφ.

Так ка теперь прямая L совпадает с осью аппликат, то соответствующая матрица имеет следующий вид:

$$[Rz] = \begin{matrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

4-й шаг. Поворот вокруг оси ординат на угол -θ.

5-й шаг. Поворот вокруг оси абсцисс на угол -ψ.

Однако вращение в пространстве некоммутативно. Поэтому порядок, в котором проводятся вращения, является весьма существенным.

6-й шаг. Перенос на вектор A (a, b, c).

Перемножив найденные матрицы в порядке их построения, получим следующую матрицу:

[T][Rx][Ry][Rz][Ry]-1[Rx]-1 [T]-1.

Выпишем окончательный результат, считая для простоты, что ось вращения ходит через начальную точку.

$l^2 + \cos\varphi(1 - l^2)$	$l(1 - \cos\varphi)m + n\sin\varphi$	$l(1 - \cos\varphi)n - m\sin\varphi$	0
$l(1 - \cos\varphi)m - n\sin\varphi$	$m^2 + \cos\varphi(1 - m^2)$	$m(1 - \cos\varphi)n + l\sin\varphi$	0
$l(1 - \cos\varphi)n + m\sin\varphi$	$m(1 - \cos\varphi)n - l\sin\varphi$	$n^2 + \cos\varphi(1 - n^2)$	0
0	0	0	1

Рассматривая примеры подобного рода, мы будем получать в результате невырожденные матрицы вида

$$[A] = \begin{matrix} \alpha_1 & \alpha_2 & \alpha_3 & 0 \\ \beta_1 & \beta_2 & \beta_3 & 0 \\ \gamma_1 & \gamma_2 & \gamma_3 & 0 \\ \lambda & \mu & \nu & 1 \end{matrix}$$

При помощи таких матриц можно преобразовать любые плоские и пространственные фигуры.

Пример 4. Требуется подвергнуть заданному аффинному преобразованию выпуклый многогранник.

Для этого сначала по геометрическому описанию отображения находим его матрицу [A]. Замечая далее, что произвольный выпуклый многогранник однозначно задается набором всех своих вершин

$V_i (x_i, y_i, z_i)$, i = 1,…,n,

Строим матрицу

$$V = \begin{matrix} x1 & y1 & z1 & 1 \\ . & . & . & . \\ xn & yn & zn & 1 \end{matrix} \qquad (4.18)$$

Подвергая этот набор преобразованию, описываемому найденной невырожденной матрицей четвертого порядка, [V][A], мы получаем набор вершин нового выпуклого многогранника – образа исходного (рис. 12).

Рис. 11

3.3 Проектирование

Ортографическая проекция - картинная плоскость совпадает с одной из координатных плоскостей или параллельна ей. Матрица проектирования вдоль оси X на плоскость YOZ имеет вид

$$[P_x] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

В случае, если плоскость проектирования параллельна координатной плоскости, необходимо умножить матрицу [Px] на матрицу сдвига . Имеем

$$[P_x]*\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\p&0&0&0\end{bmatrix}=\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

Аксонометрическая проекция - проектирующие прямые перпендикулярны картинной плоскости .

Различают три вида проекций в зависимости от взаимного расположения плоскости проектирования и координатных осей:

- триметрия-нормальный вектор картинной плоскости образует с ортами координатных осей попарно различные углы(рис.15);
- диметрия-два угла между нормалью картинной плоскости и координатными осями равны (рис. 16).

- изометрия-все три угла между нормалью картинной плоскости и координатными осями равны (рис. 17).

Каждый из трех видов указанных проекций получается комбинацией поворотов, за которой следует параллельное проектирование.

Косоугольные проекции – при проектировании используется пучок проямых не перпендикулярных плоскости экрана.

При косоугольном проектировании орта оси Z на плоскость XY имеем

(0 0 1 1) -> (□ □ 0 1)

Матрица соответствующего преобразования имеет вид

1 0 0 0

0 1 0 0

□ □ 0 1

0 0 0 1

Выделяют 2 вида косоугольных проекций: свободную (угол наклона проетирующих прямых равен половине прямого) и кабинетную (частный случай свободной – масштаб по третьей оси вдвое меньше)

В случае свободной проекции

□ □ □= cos □/4

а в случае кабинетной

□ □ □= 0.5 cos □/4

Перспективные (центральные) проекции строятся более сложно . Предположим что центр проектирования лежит на оси Z - C (0,0,c) а плоскость проектирования совпадает с координатной плоскостью XOY (рис. 19) . Возьмем в пространстве произвольную точку M(x,y,z), проведем через нее и точку C прямую и запишем ее параметрические уравнения . Имеем:

X`= xt , Y`=yt, Z`= c+(z-c)t

Найдем координаты точки пересечения этой прямой с плоскостью XOY. Из того , что z`=0, получаем

$$t'=\frac{1}{1-z/c}\qquad X'=\frac{1}{1-z/c},\quad Y'=\frac{1}{1-z/c}$$

Тот же самый результат мы получим, привлекая матрицу

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

В самом деле,

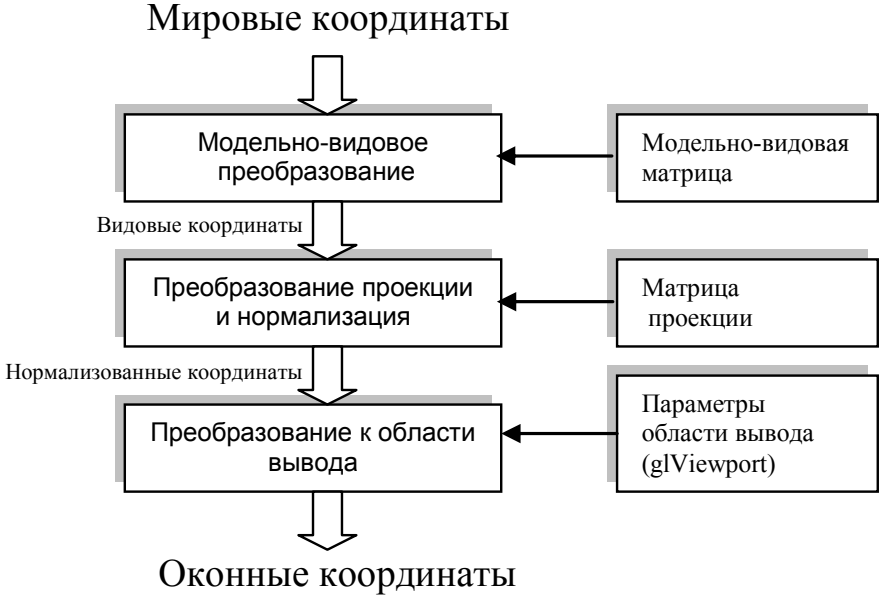
$$(x\,y\,z\,1)\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/c \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left(x\ y\ 0\ 1-\frac{z}{c}\right) \quad \left(\frac{x}{1-z/c}\ \frac{y}{1-z/c}\ 0\ 1\right)$$

Матрица проектирования, конечно, вырождена ; матрица же соответствующего перспективного преобразования(без проектирования) имеет следующий вид

$$[Q] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

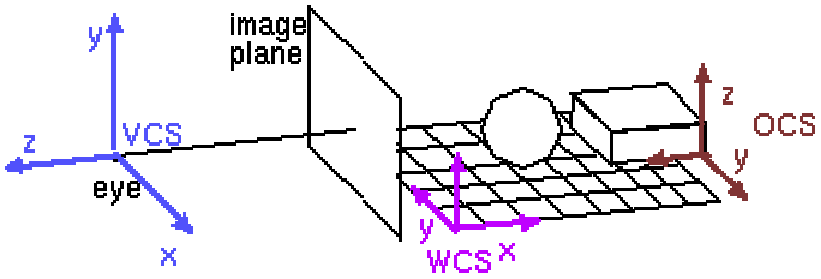
3.4 Этапы создания графического объекта. Графический конвейер.

Для получения двумерного изображения трехмерного объекта необходимо осуществить преобразование системы мировых координат в систему видовых координат.
В целом, для отображения трехмерных объектов сцены в окно приложения используется последовательность, показанная на рисунке:



Модельно-видовые преобразования

К модельно-видовым преобразованиям относят перенос, поворот и изменение масштаба. Для проведения этих операций достаточно умножить на соответствующую матрицу каждую вершину объекта и получить измененные координаты этой вершины:
$(x', y', z', 1)^T = M * (x, y, z, 1)^T$,
где M – матрица модельно-видового преобразования. Перспективное преобразование и проектирование производится аналогично. Сама матрица может быть создана с помощью следующих команд



Модельное преобразование

Переводит модель, заданную в локальных (собственных) координатах, в глобальное (мировое пространство)

Модель «собирается» из частей, с помощью модельных преобразований (обычно композиция переносов, поворотов, масштабирования),

На выходе получаем модель в единых мировых координатах.
Виртуальная камера Определяет положение наблюдателя в пространстве. Ее параметры: положение, направление взгляда, направление «вверх», параметры проекции. Они задаются матрицей видового преобразования

Видовое преобразование

Для чего нужно еще одно преобразование?

Проективные преобразования описывают «стандартные» проекции, т.е. проецируют фиксированную часть пространства.

Если мы хотим переместить наблюдателя, то нужно либо изменить матрицу проекции чтобы включить в нее информации о камере, либо применить дополнительное преобразование, «подгоняющее» объекты под стандартную камеру.

Стандартная камера в OpenGL:
Наблюдатель в (0, 0, 0)
Смотрит по направлению (0, 0, -1)
Верх (0, 1, 0)

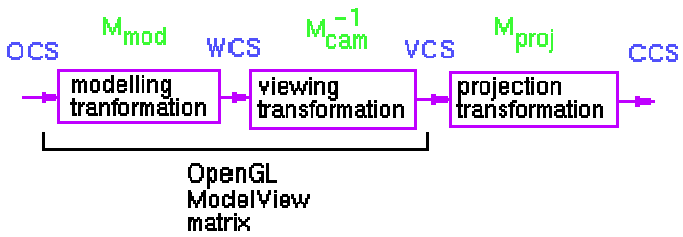
Итак, видовое - «подгоняет» мир под стандартную камеру, преобразует мировую систему координат в видовые координаты (которые подходят для «стандартной» камеры На выходе – модель, готовая к проекции на экран.

Проективное преобразование - Выполняет 3D преобразование, подготавливая модель к переходу на 2D

После перспективного преобразования необходимо отбросить координату z и получить значения в оконных координатах (обычно от –1 до 1)

Модельно-видовое преобразование. OpenGL не имеет раздельных матриц для видового и модельного преобразования, поэтому нужно задавать сразу произведение:

$$M = M_{view} \cdot M_{mdl}$$



4 Удаление невидимых граней

Алгоритмы удаления невидимых граней могут быть условно поделены на два класса в зависимости от принципов, заложенных для их реализации. Первый класс – это алгоритмы работающие в пространстве объекта. Это означает, что для определения видимости данной грани сравнивается ее взаимное расположение со всеми остальными гранями в трехмерной сцене. Пусть N – количество граней в трехмерной сцене. Для построения трехмерной сцены в этом случае необходимо сравнить положение каждой грани с оставшимися, что требует порядка N^2 операций. Например, пусть количество граней в трехмерной сцене $N = 1000$, тогда время работы алгоритмов этого класса порядка 1,000,000 операций.

Другой класс алгоритмов - работающих в пространстве изображения, основан на нахождении точки ближайшей грани которую пересекает луч зрения, проходящий через заданную точку на растре. Поскольку число точек на растровом экране фиксировано, то алгоритмы этого класса менее чувствительны к увеличению количества объектов в трехмерной сцене. Пусть n - число точек на растровом экране. Тогда количество операций, необходимых для построения трехмерной сцены будет порядка $n \cdot N$. Например, для экранного разрешения 320 $\times$ 200 точек, $n = 64000$, тогда количество операций для $N = 1000$ граней будет порядка 64,000,000. Выбор класса алгоритма может зависеть от особенностей конкретной задачи, а также от способов реализации алгоритма.

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.

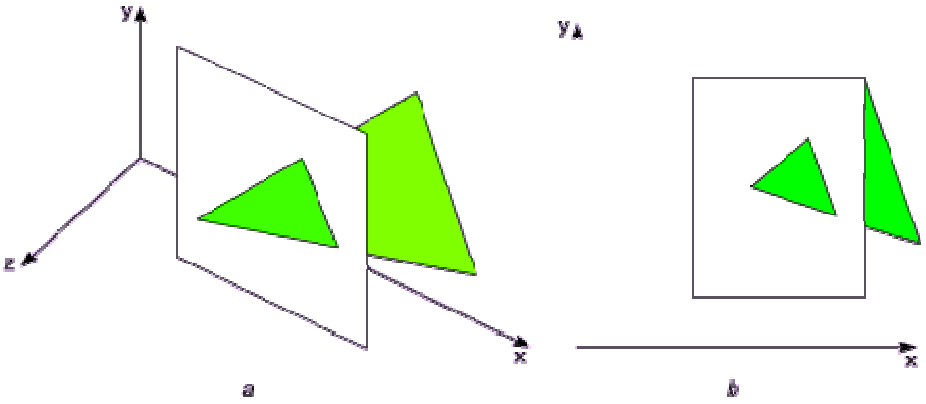


рис. 21.1

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.

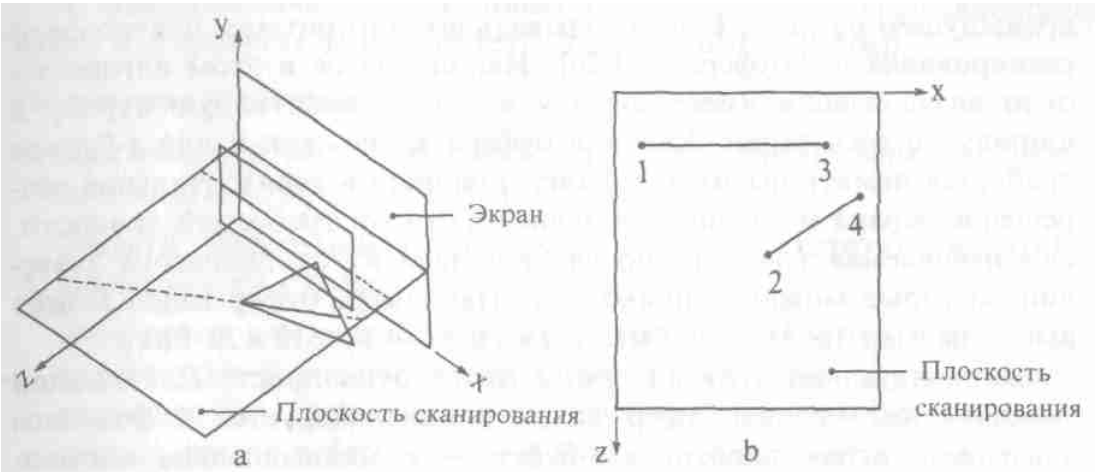


Рис. 4.57. Сканирующая плоскость.

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.

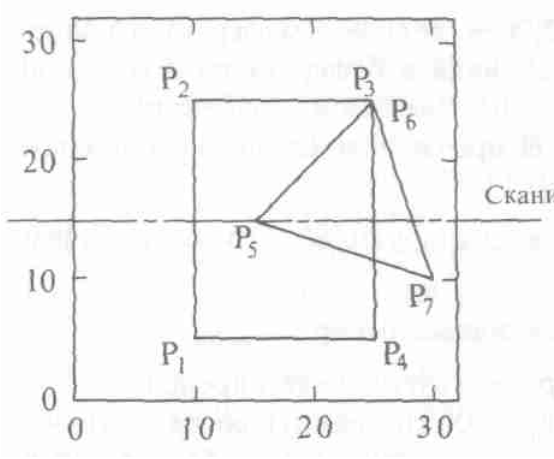


Рис. 4.58. Многоугольники из примера 4.22.

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:

Буфер кадра для строки																																			
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
z-буфер для строки																																			
0	0	0	0	0	0	0	0	0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0	0	0

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:

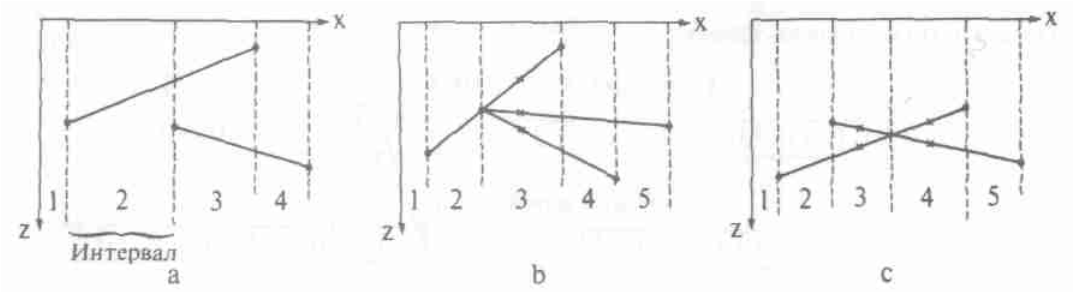


Рис. 4.59. Интервалы для построчного сканирования.

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.

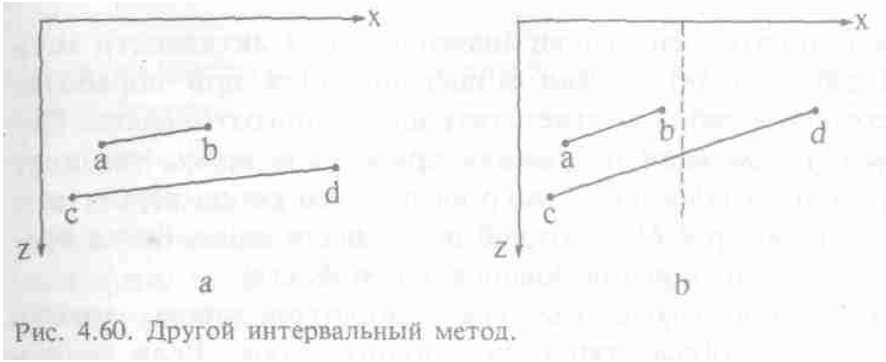


Рис. 4.60. Другой интервальный метод.

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, c). 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.

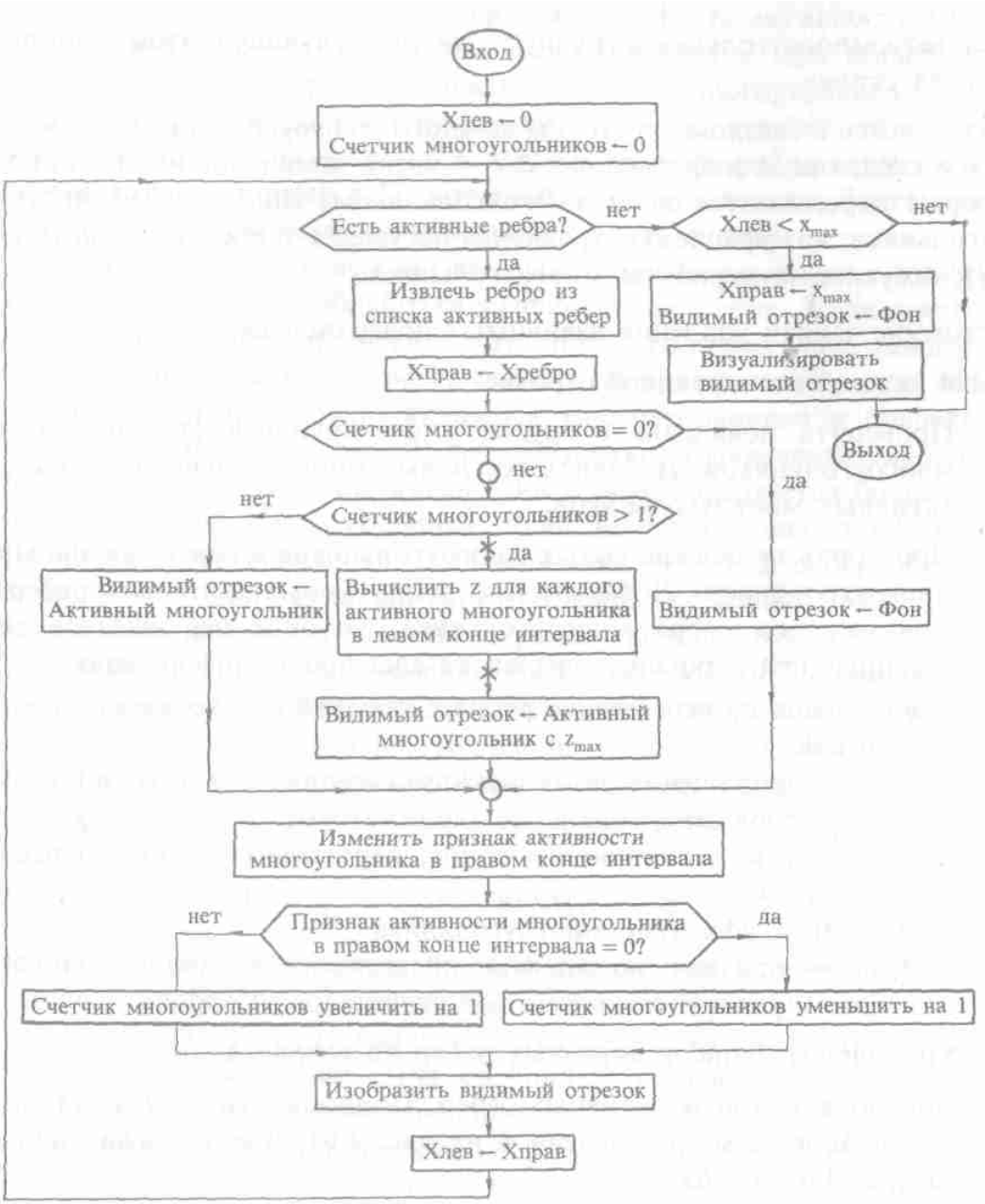


Рис. 4.61. Блок-схема интервального алгоритма для непротыкающих многоугольников.

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.

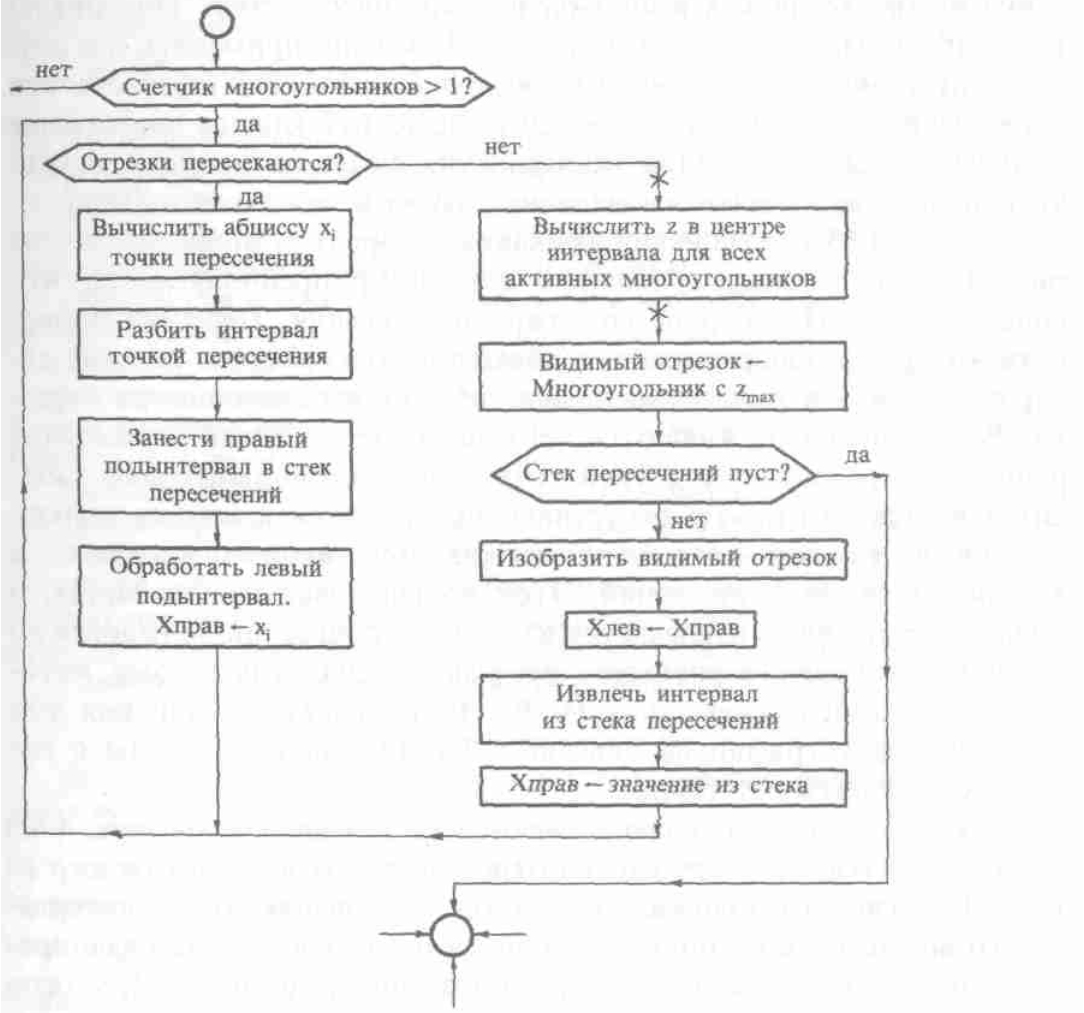


Рис. 4.63. Модификация блок-схемы с рис. 4.61 для протыкающих многоугольников.

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:

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

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:

$z = (120 - 15.5 - 3x)/4 = 10$ $x_i = 21.5$

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.

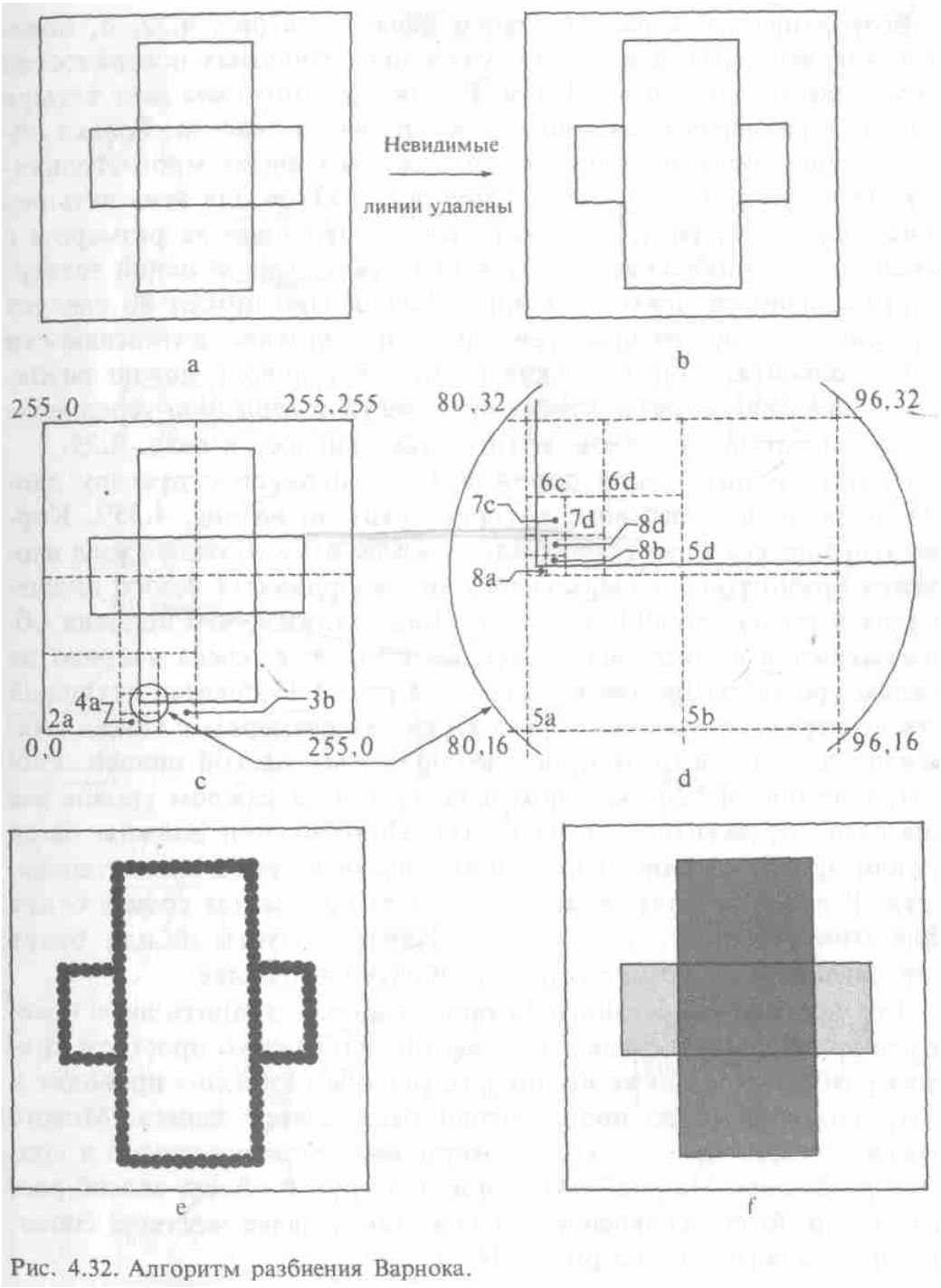


Рис. 4.32. Алгоритм разбиения Варнока.

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.

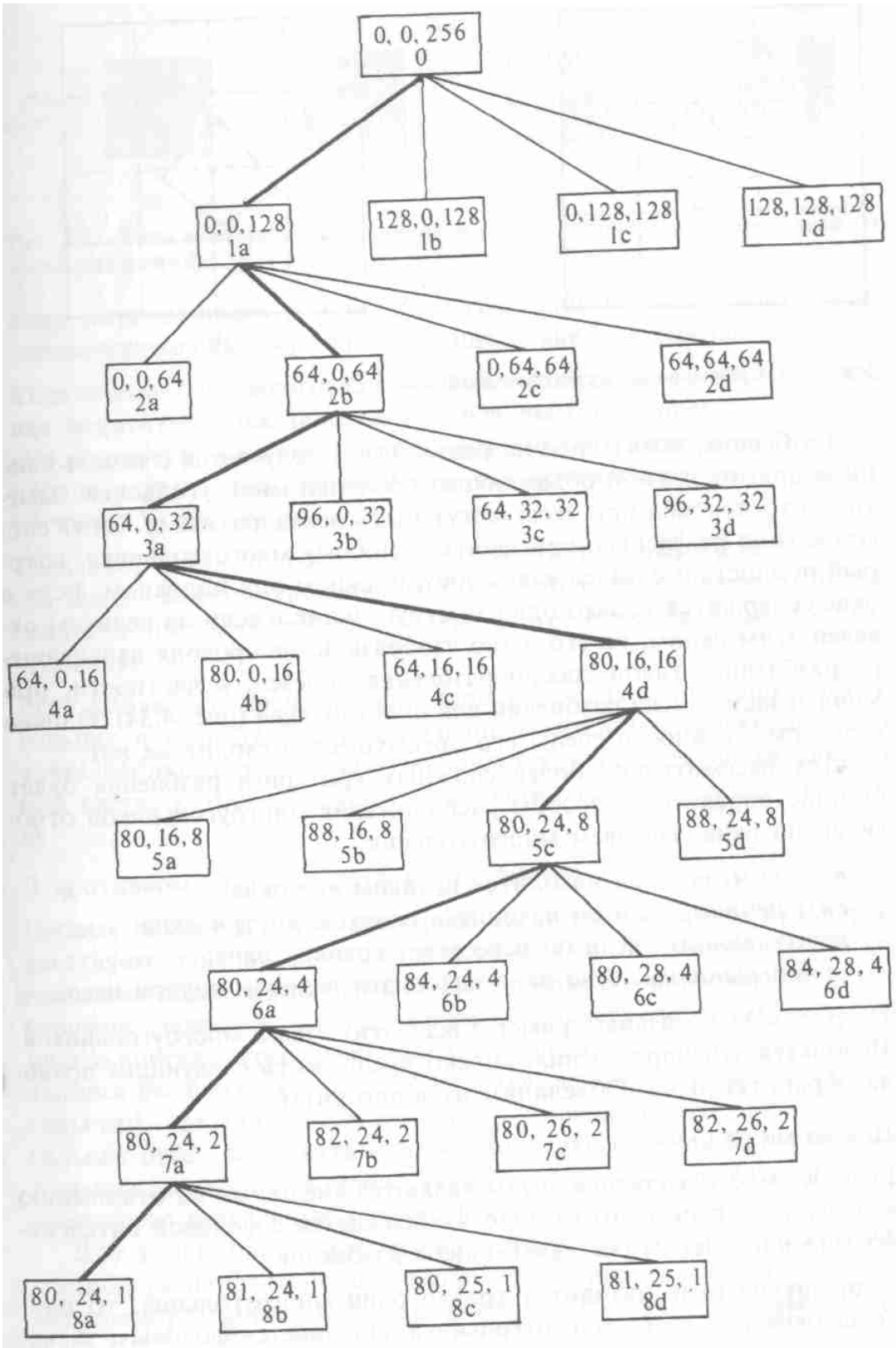


Рис. 4.33. Дерево структуры подокон.

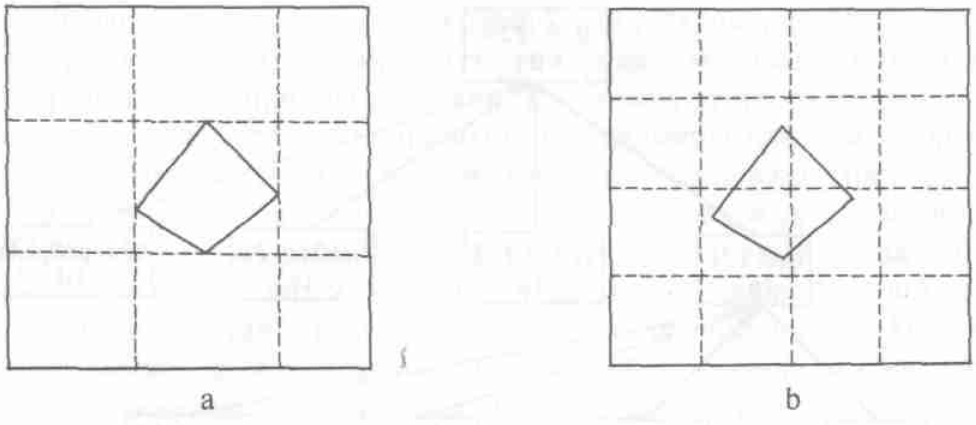


Рис. 4.34. Сравнение двух способов разбиения.

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

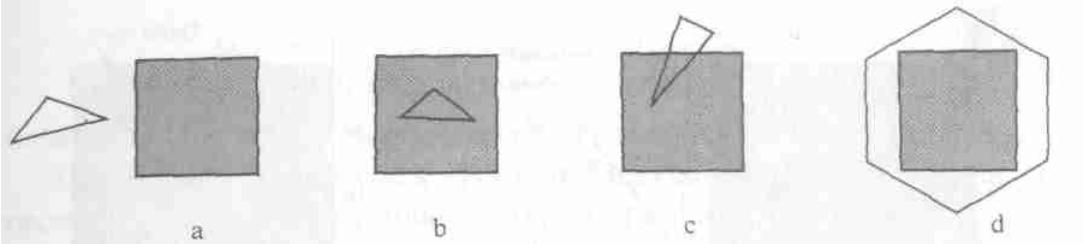


Рис. 4.35. Типы многоугольников: внешний (a), внутренний (b), пересекающий (c), охватывающий (d).

.

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

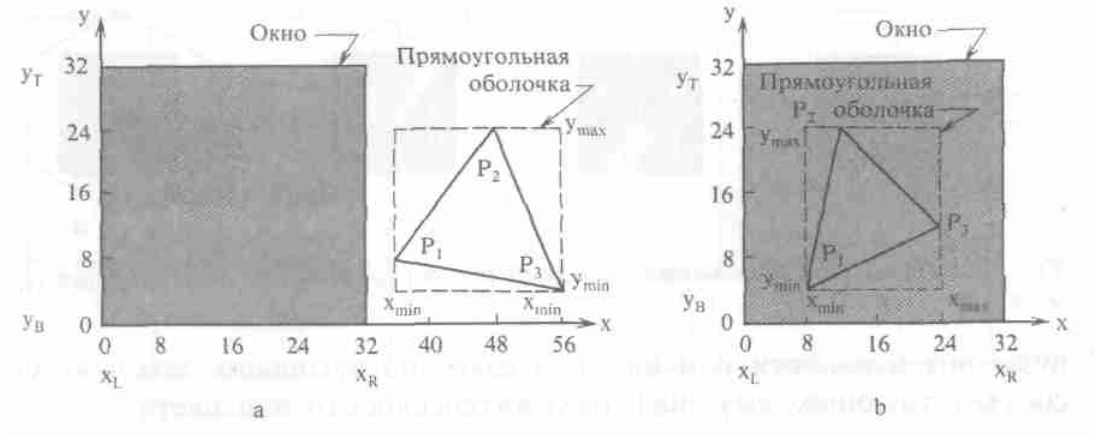


Рис. 4.36. Проверка с помощью прямоугольной оболочки для внешнего и внутреннего многоугольников.

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:

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

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

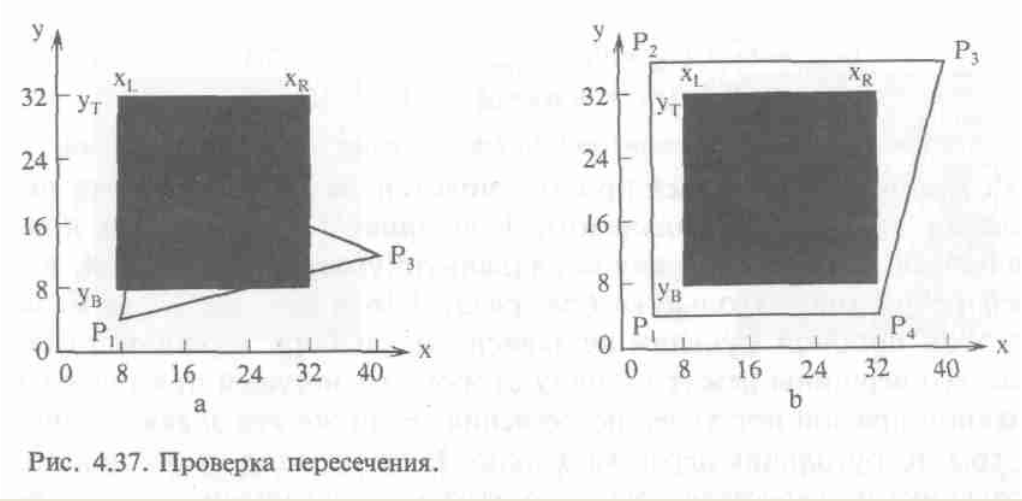


Рис. 4.37. Проверка пересечения.

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:

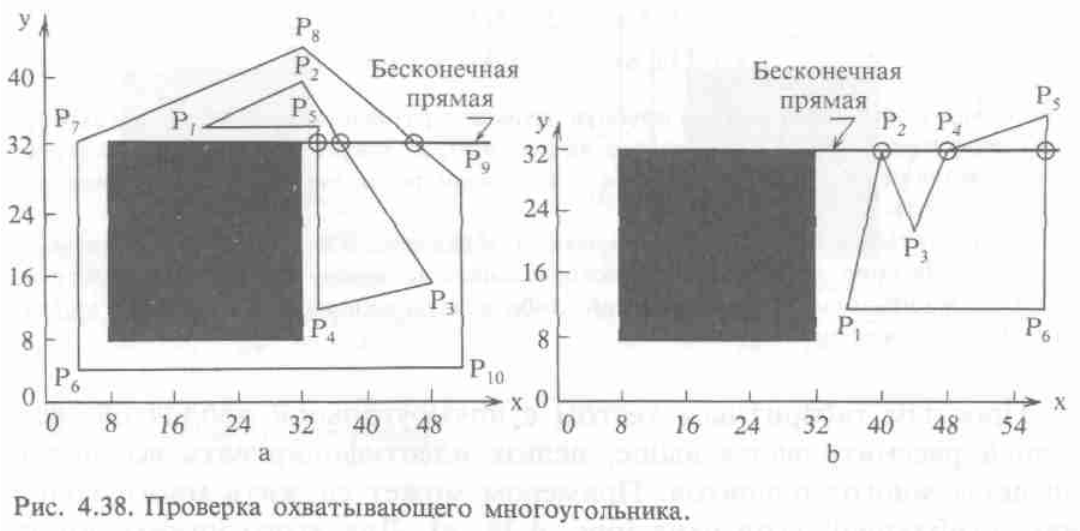


Рис. 4.38. Проверка охватывающего многоугольника.

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

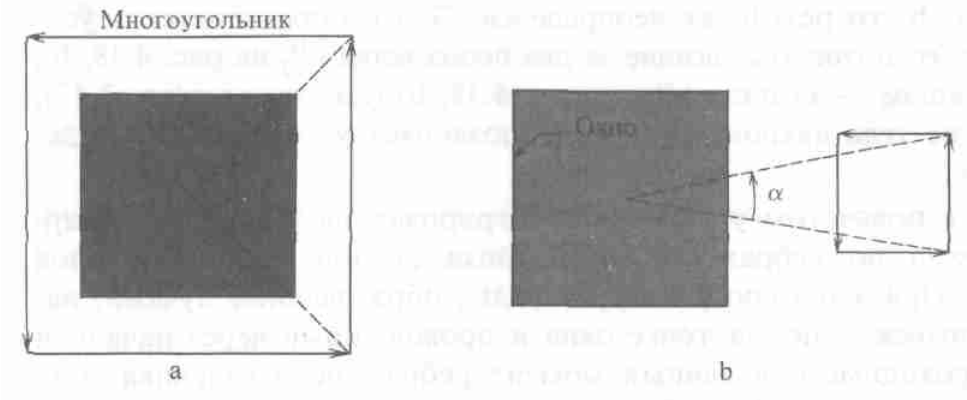


Рис. 4.39. Проверка с подсчетом угла.

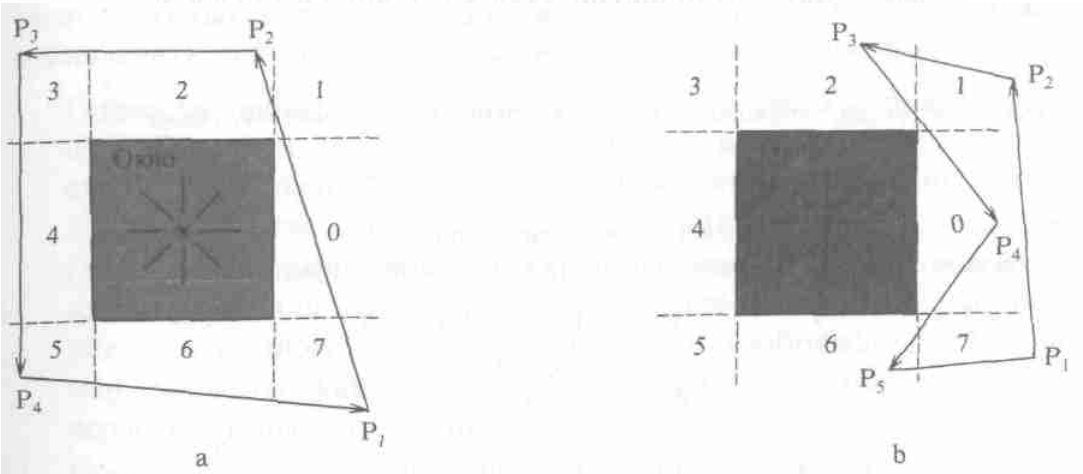


Рис. 4.40. Угловой тест для охватывающего и внешнего многоугольников.

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).

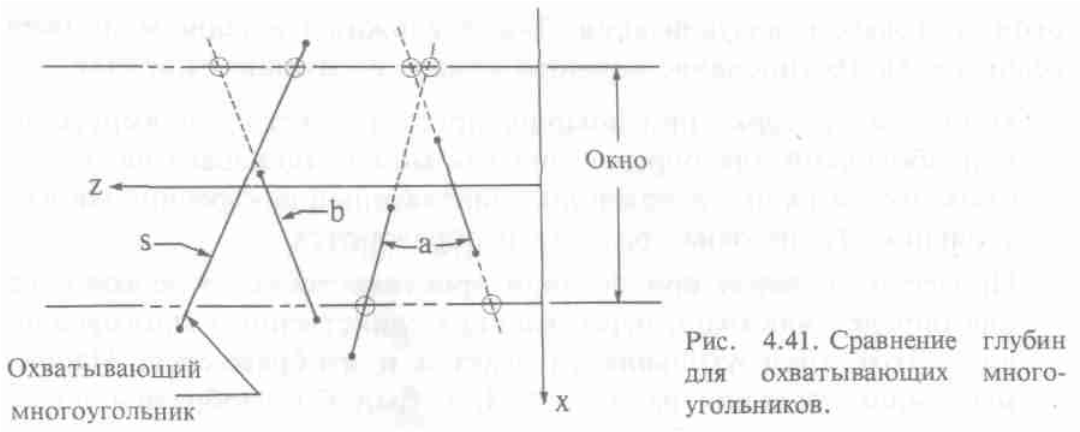


Рис. 4.41. Сравнение глубин для охватывающих многоугольников.

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.

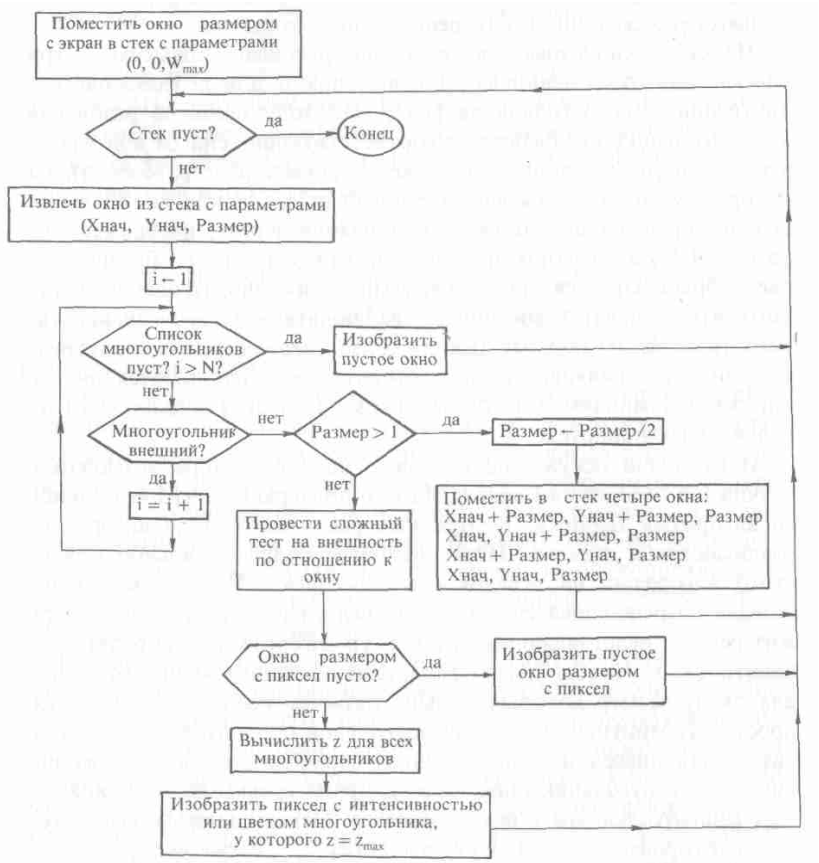


Рис. 4.42. Блок-схема простого алгоритма Варнока.

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.

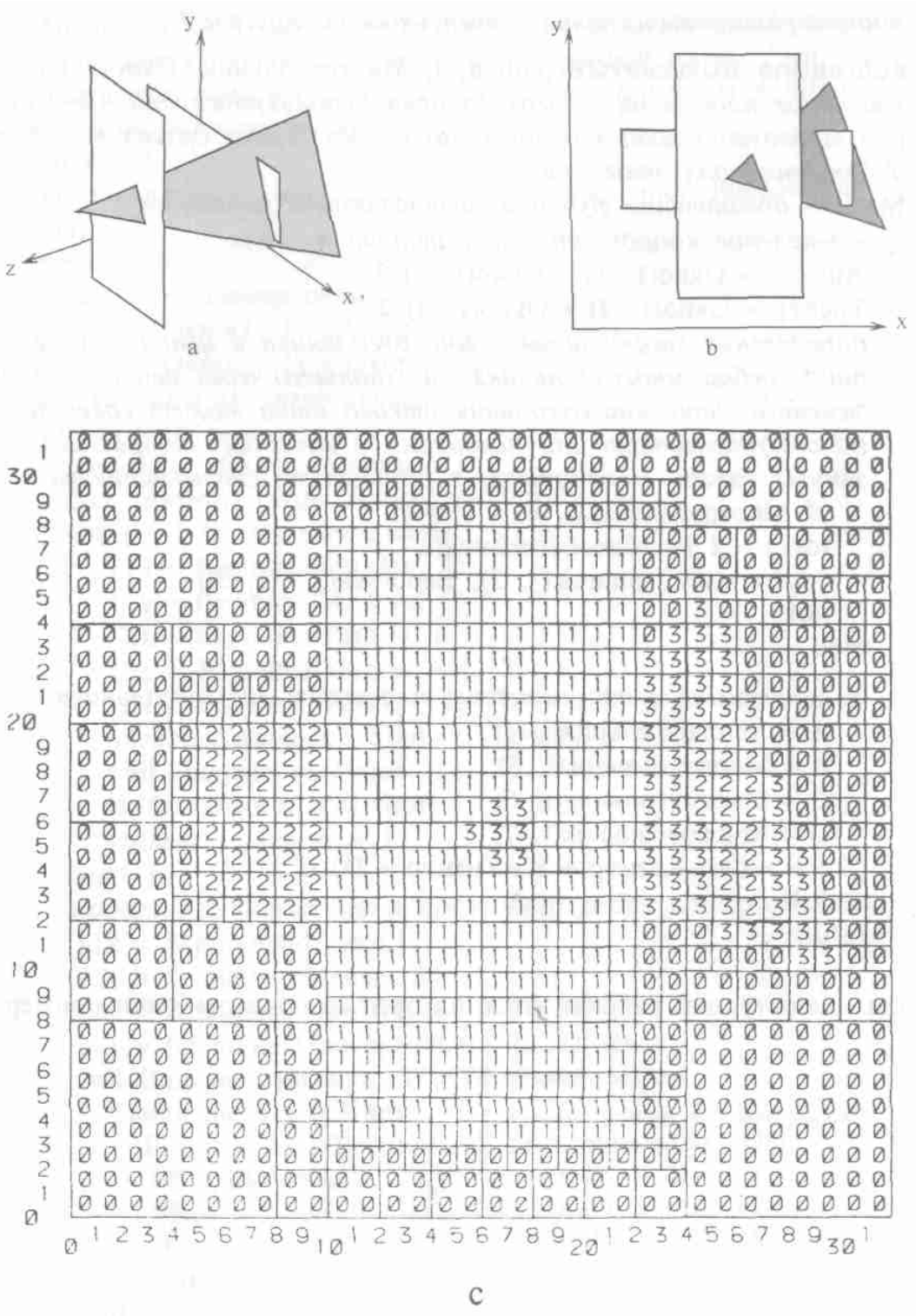


Рис. 4.43. Пример многоугольников для простого алгоритма Варнока.

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

5 Создание реалистических изображений

С развитием вычислительной техники и расширением возможностей компьютеров все больший интерес вызывает задача создания изображения по заданной модели сцены. Построенное таким образом изображение в идеале должно восприниматься человеком как максимально приближенного к реальному. Подобная задача возникает в целом ряде самых разных областей - от рекламы и компьютерных игр до архитектурных сооружений и разработки новых моделей машин. Как правило, чем выше желаемая степень реалистичности, тем больше вычислительных ресурсов (времени процессора, памяти) требуется для синтеза изображения.

Как известно, световая энергия, падающая на поверхность от источника света, может быть отражена, поглощена или пропущена. То есть, объект можно увидеть, только если он отражает или пропускает свет; если же объект поглощает весь падающий свет, то он невидим и называется абсолютно черным телом.

Распределение падающей энергии на поверхности зависит от длины световой волны и свойств поверхности. Цвет поверхности определяется поглощаемыми длинами волн, однако для закрашивания объекта используется оценка отражённого света. Чем интенсивность отражённого света выше, тем выше должна быть интенсивность цвета пикселя при закраске.

5.1 Простейшая модель освещенности.

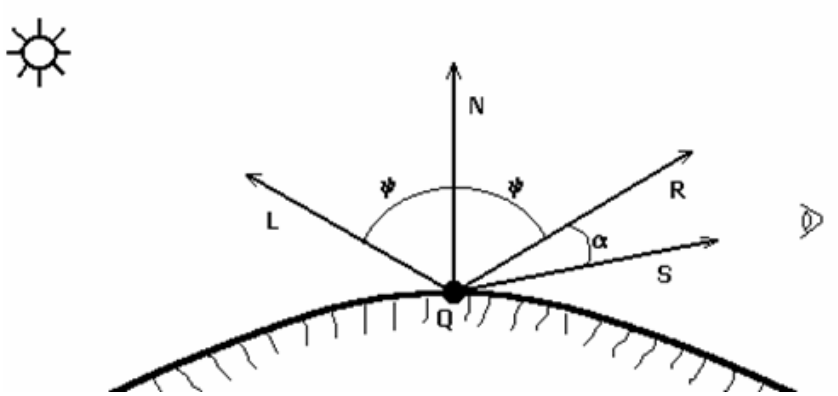


Рис. 5.1 Модель отражения

Простейшая модель освещенности предназначена для отрисовки простейших тел, не обладающих свойствами зеркального отражения (reflection) и прозрачности. От таких тел обычно рассматривают диффузное и зеркальное отражение.

Свет может быть отражен от таких тел Отраженный объекта свет может также быть диффузным или зеркальным. (Рис. 5.1)

Зеркальное отражение происходит от внешней поверхности объекта. Поверхность считается *идеально зеркальной,* если на ней отсутствуют какие-либо неровности, шероховатости. Собственный цвет у такой поверхности наблюдается. Световая энергия падающего луча отражается только по линии отраженного луча. Какое-либо рассеяние в стороны от этой линии отсутствует.

В природе, вероятно, нет идеально гладких поверхностей, поэтому полагают, что если глубина шероховатостей существенно меньше длины волн излучения, то рассеивания не

наблюдается. Для видимого спектра можно принять, что глубина шероховатостей поверхности зеркала должна быть существенно меньше 0.5 мкм.

Падающий луч, попадая на слегка шероховатую поверхность реального зеркала, порождает не один отраженный луч, а несколько лучей, рассеиваемых по различным направлениям. Зона рассеивания зависит от качества полировки и может быть описана некоторым законом распределения. Как правил форма зоны рассеивания симметрична относительно линии идеального зеркально отраженного луча. К числу простейших, но достаточно часто используемых, относится эмпирическая модель распределения Фонга, согласно которой интенсивность зеркально отраженного излучения пропорциональна $\cos\alpha$, где α — угол направлением на наблюдателя и линией идеально отраженного луча.

$$I_s = I_l K_s \cos^p \alpha$$

где, $\boldsymbol{I_l}$— интенсивность ***l***-того источника света, $\boldsymbol{K_s}$— коэффициент зеркального отражения и зависит от свойств материала поверхности. Значение K_s находится в диапазоне от 0 до 1. Показатель ***p*** – коэффициент Фонга, находится в диапазоне от 1 до 200 и зависит от качества полировки.
Заметим, что интенсивность отраженного света зависит от расположения наблюдателя и в точке зеркального отражения наблюдатель будет наблюдать блик.

Диффузное отражение. Этот вид отражения присущ *матовым* поверхностям. Матовой можно считать такую поверхность, размер шероховатости которой уже настолько велик, что падающий луч рассеивается равномерно все стороны. Такой тип отражения характерен, например, для гипса или бумаги. Диффузное отражение описывается законом Ламберта, согласно которому интенсивность отраженного света пропорциональна косинусу угла между нормалью и падающим лучом.

$$I_d = I_l K_d \cos\psi$$

где K_d— коэффициент диффузного отражения и зависит от свойств материала поверхности. Значение K_d находится в диапазоне от 0 до 1. Интенсивность отраженного света не зависит от расположения наблюдателя. При этом положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям.

При создании реалистичных изображений следует учитывать то, что в природе, вероятно, не существует идеально зеркальных или полностью матовых поверхностей. При изображении объектов средствами компьютерной графики обычно моделируют сочетание зеркальности и диффузного рассеивания в пропорции, характерной для конкретного материала. В этом случае модель отражения записывают в виде суммы диффузной и зеркальной компонент. Кроме того, в реальных сценах обычно нет полностью затемненных объектов, следует учитывать освещение рассеянным светом, отраженным от других объектов – фоновое освещение.

Запишем формулу для вычисления интенсивности света в точке **Q**.

$$I = I_\alpha * K_\alpha + \frac{I_l}{k + d}\left(k_d(\vec{\mathbf{N}} * \vec{\mathbf{L}}) + k_s(\vec{\mathbf{R}} * \vec{\mathbf{S}})^n\right), \qquad (5.1)$$

где **N**, **L**, **R**, **S** – единичные векторы (рис. 5.1):
L – вектор направления на источник света;
N – нормаль к поверхности в точке **Q**;

R – отраженный луч;
S – вектор наблюдения.
I_a – интенсивность фонового света. K_a – коэффициент фонового света.
κ – произвольная постоянная, d– определяется положением ближайшего объекта к точке наблюдения, т.е. ближайший объект освещается с полной интенсивностью, а более далекие – с уменьшенной, n – коэффициент Фонга.

5.2. Простейшая закраска граней многогранника

Этот способ закраски заключается в том, что на каждой из граней выбирается по одной точке, для неё определяется освещённость, а затем вся грань закрашивается с найденной освещённостью.

Предложенный алгоритм является самым быстрым, однако, обладает одним большим недостатком – если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник. Казалось бы, этот дефект можно замаскировать за счет увеличения количества граней при аппроксимации поверхности. Но зрение человека имеет способность подчеркивать перепады яркости на границах смежных граней — такой эффект называется эффектом *полос Маха*. Поэтому для создания иллюзий гладкости нужно намного увеличить количество граней, что приводит к существенному замедлению визуализации— чем больше граней, тем меньше скорость рисования объектов.

Для вуалирования переходов освещенности между этими поверхностями используют методы Гуро и Фонга, в которых выполняют аппроксимацию (то есть сглаживание).

5.3 Закраска методом Гуро

Метод Гуро является достаточно простым методом закраски поверхностей и основывается на определении освещенности грани в ее вершинах с последующей билинейной интерполяцией получившихся величин на всю грань.
Закрашивание граней по методу Гуро осуществляется в четыре этапа.

1. Вычисляются нормали к каждой грани.
2. Определяются нормали в вершинах. Нормаль в вершине определяется усреднением нормалей примыкающих граней.
3. На основе нормалей в вершинах вычисляются значения интенсивностей в вершинах согласно выбранной модели отражения света.
4. Закрашиваются полигоны граней цветом, соответствующим линейной интерполяции значений интенсивности в вершинах.

Обратимся к рисунку, на котором изображена выпуклая четырехугольная грань.

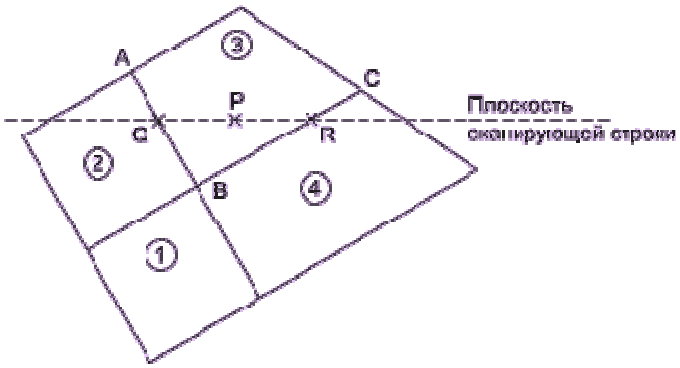


Рис. 5.2. Аппроксимация при использовании методов Гуро и Фонга.

Предположим, что интенсивности в ее вершинах **A, B,C** известны и равны I_A, I_B, I_C.
Пусть **P** – произвольная точка грани. Для определения интенсивности (освещенности) в этой точке проведем через нее горизонтальную прямую. Обозначим через **Q** и **R** точки пересечения проведенной прямой с границей грани.
Будем считать, что интенсивность на отрезке **QR** изменяется линейно, то есть

$$I_P = (1 - t)I_Q - tI_R,$$

где $t = |QP| / |QR|, 0 \le t \le 1$
Для определения интенсивности в точках **Q** и **R** вновь воспользуемся линейной интерполяцией, также считая, что вдоль каждого из ребер границы интенсивность изменяется линейно.
Тогда интенсивность в точках **Q** и **R** вычисляется по формулам:

$$I_Q = (1 - u)I_A - uI_B,$$
$$I_R = (1 - v)I_C - vI_B,$$

где $u = |AQ|/|AB|, |, 0 \le u \le 1$ $v = |CR|/|CB|, 0 \le v \le 1.$

Метод Гуро обеспечивает непрерывное изменение интенсивности при переходе от одной грани к другой без разрывов и скачков.
Еще одним преимуществом этого метода является его инкрементальный характер: грань рисуется в виде набора горизонтальных отрезков, причем так, что интенсивность последующего пиксела отрезка отличается от интенсивности предыдущего на величину постоянную для данного отрезка. Кроме того, при переходе от отрезка к отрезку значения интенсивности в его концах также изменяются линейно.

Таким образом, процесс рисования грани слагается из следующих шагов:
1. Проектирование вершин грани на экран;
2. Отыскание интенсивностей в вершинах по формуле учитывающий фоновое освещение и диффузное отражение;
3. Определение координат концов очередного отрезка и значений интенсивности в них линейной интерполяцией;
4. Рисование отрезка с линейным изменением интенсивности между его концами.

Замечания:

1. При определении освещенности в вершине, естественно, встает вопрос о выборе нормали. В качестве нормали в вершине выбирается нормированная сумма нормалей прилегающих граней
$\mathbf{N} = (a_1N_1 + \ldots + a_kN_k)/|a_1n_1 + \ldots + a_kn_k|$, где $a_1 \ldots a_k$– произвольные весовые коэффициенты.
2. Дефекты изображения, возникающие при закраске Гуро, частично объясняются тем, что этот метод не обеспечивает гладкости изменения интенсивности.

5.4 Закраска методом Фонга

Закраска Фонга требует больших вычислительных затрат, однако она позволяет разрешить многие проблемы метода Гуро. При закраске Фонга вдоль сканирующей строки интерполируются не интенсивности отраженного света, а вектор нормали, который затем используется в модели освещения для вычисления интенсивности пикселя. Включает следующие этапы:

1 Определяются нормали к граням;
2 По нормалям к граням определяются нормали в вершинах.
3 В каждой точке закрашиваемой грани определяется интерполированный вектор нормали;
4 По направлению векторов нормали определяется цвет точек грани в соответствии с выбранной моделью отражения света.

При этом достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, получается более реалистичное изображение. В частности, правдоподобнее выглядят зеркальные блики. Это объясняется тем, что при расчете освещенности

используется формула (5.1), учитывающая не только рассеянный свет и диффузное отражение (как в методе Гуро), но и отраженный свет благодаря которому появляются блики.
При закраске Фонга аппроксимация кривизны поверхности производится сначала в вершинах многоугольников путем аппроксимации нормали в вершине. После этого билинейной интерполяцией вычисляется нормаль в каждом пикселе.
Например:
Ищем нормаль в точке **Q** линейной интерполяцией между **A** и **B**, в **R**– между **B**и **C**, и , наконец, в **P** – между **Q** и **R**. Таким образом:

$$N_q = u * N_a + (1-u) * N_b$$
$$N_r = w * N_b + (1-w) * N_c$$
$$N_p = t * N_q + (1-t) * N_r$$

где **u = |AQ|/|AB|, w = |BR|/|BC|, t = |QP|/|QR|, 0≤t≤1, 0≤u≤1, 0≤w≤1.**
Хотя метод Фонга устраняет большинство недостатков метода Гуро, он тоже основывается на линейной интерполяции. Поэтому в местах разрыва первой производной интенсивности заметен эффект полос Маха, хотя и не такой сильной, как при закраске Гуро. Кроме того, закраска может значительно изменяться от кадра к кадру. Это происходит из-за того, что правило закраски зависит от поворотов, а обработка ведется в пространстве изображения. Поэтому, когда от кадра к кадру меняется ориентация объекта, его закраска тоже изменяется, причем достаточно заметно.

Метод Фонга дает значительно лучшие результаты, в особенности при имитации зеркальных поверхностей. Общие черты и отличия методов Гуро и Фонга можно показать на примере цилиндрической поверхности, аппроксимированной многогранником (рис. 4.40). Пусть источник света находится позади нас. Проанализируем закрашивания боковых граней цилиндра.

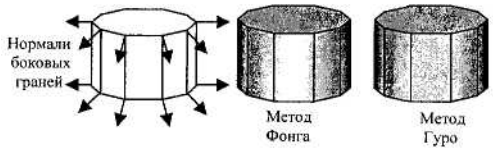


Рис. 5.3. Отличия закраски Гуро и Фонга.

На рис. 5.3 на закрашенной поверхности показаны черным цветом ребра граней — это сделано для иллюстрации особенностей закрашивания, на самом деле после закрашивания никакого черного каркаса не будет, и поверхность выглядит гладкой.
Основные отличия можно заметить для закрашивания передней грани. Он перпендикулярна направлению лучей света. Поэтому нормали в вершина этой грани располагаются симметрично — они образовывают попарно равные по абсолютной величине углы с лучами света. Для метода Гуро это обуславливает одинаковые интенсивности в вершинах передней грани. А раз интенсивности одинаковые, то и для любой точки внутри этой грани интенсивность одинакова (для линейной интерполяции). Это обуславливает единый цвет закрашивания. Все точки передней грани имеют одинаковый цвет что, очевидно, неправильно.
Метод Фонга дает правильное закрашивание. Если интерполировать вектор нормалей передней грани, то по центру будут интерполированные нормали параллельные лучам света (рис. 5.4).
По методу Фонга центр передней грани будет светлее, чем края.

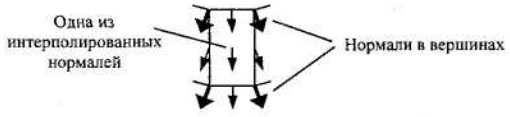


Рис. 5.4. Интерполяция нормалей в методе Фонга.

5.5 Прямая и обратная трассировка лучей

Мы опишем один из самых простых методов создания реалистических изображений, позволяющий тем не менее достичь достаточно высокого качества (разумеется, при учете ряда сложных эффектов, таких, как отражение и преломление).

Методы трассировки лучей на сегодняшний день считаются наиболее мощными и универсальными методами получения реалистичных изображений. Известно много примеров реализации алгоритмов трассировки для качественного отображения самых сложных трехмерных сцен. Можно отметить, что универсальность методов трассировки в значительной степени обусловлено тем, что в их основе лежат простые и ясные понятия, отражающие наше восприятие окружающего мира.
Предположим, окружающие нас объекты обладают по отношению к свету следующими свойствами:

- излучают;
- отражают и поглощают;
- пропускают сквозь себя.

Каждое из этих свойств можно описать некоторым набором характеристик. Например, излучение можно охарактеризовать интенсивностью, направленностью, спектром. Излучение может исходить от условно точечного источника (далекая звезда) или протяженного (скажем, от извергающейся из кратера вулкана расплавленной лавы). Распространение излучения может осуществляться вдоль достаточно узкого луча (сфокусированный луч лазера), конусом (прожектор), равномерно во все стороны (Солнце), либо еще как-нибудь. Свойство отражения (поглощения) можно описать характеристиками диффузного рассеивания и зеркального отражения. Прозрачность можно описать ослаблением интенсивности и преломлением.
Распределение световой энергии по возможным направлениям световых лучей можно отобразить с помощью векторных диаграмм, в которых длина векторов соответствует интенсивности (рис. 5.5—5.6)

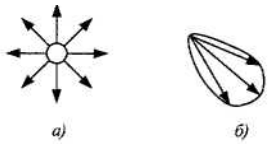


Рис.5.5. Излучение: а—равномерно во все стороны; б—направленно.

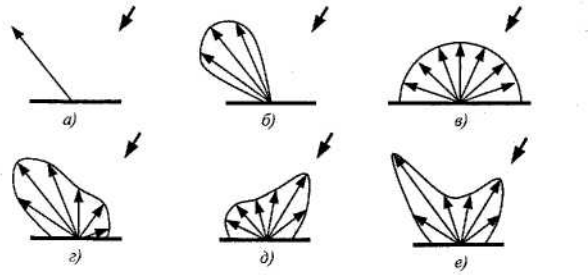


Рис 5.6. Отражение: а — идеальное зеркало, б — неидеально зеркало, в — диффузное, г — сумма диффузного и зеркального, д — обратное, е — сумма диффузного зеркального и обратного.

В предыдущих разделах были рассмотрены наиболее часто упоминаемые виды отражения — зеркальное и диффузное. Два крайних, идеализированных случая преломления изображены на рис. 5.7.

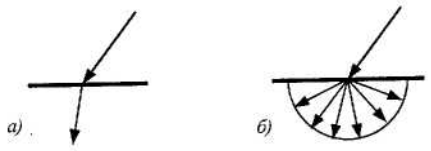


Рис 5.7. Преломление: а —идеальное, б — диффузное.

Некоторые реальные объекты преломляют лучи гораздо более сложным образом, например, обледеневшее стекло. Один и тот же объект реальной действительности может восприниматься виде источника света, а при ином рассмотрении может считаться предмет только отражающим и пропускающим свет. Например, купол облачного неба в некоторой трехмерной сцене может моделироваться в виде протяженного (распределенного) источника света, а в других моделях это же небо выступает как полупрозрачная среда, освещенная со стороны Солнца. В общем случае **каждый объект описывается некоторым сочетанием перечисленных выше трех свойств.**

Теперь рассмотрим то, как формируется изображение сцены включающей в себя несколько пространственных объектов. Будем полагать, что из точек поверхности (объема) излучающих объектов исходят лучи света. Можно назвать такие лучи первичными — они освещают все остальное. Важным моментом является предположение, что световой луч в свободном пространстве распространяется *вдоль прямой линии,* пока не встретится отражающая поверхность или граница среды преломления.

От источников излучения исходит по различным направлениям бесчисленное множество первичных лучей (даже луч лазера невозможно идеально сфокусировать — все равно свет будет распространяться не одной идеально тонкой линией, а конусом, пучком лучей). Некоторые лучи уходят в свободное пространство, а некоторые (их также бесчисленное множество) попадают на другие объекты. Если луч попадает в прозрачный объект, то, преломляясь, он идет дальше, при этом некоторая часть световой энергии поглощается. Подобно этому, если на пути луча встречается зеркально отражающая поверхность, то он также изменяет направление, а часть световой энергии поглощается. Если объект зеркальный и одновременно прозрачный (например, обычное стекло), то будет уже два луча— в этом случае говорят, что луч расщепляется.

Можно сказать, что в результате действия на объекты первичных лучей возникают вторичные лучи. Бесчисленное множество вторичных лучей уходит в свободное пространство, но некоторые из них попадают на другие объекты. Так, многократно отражаясь и преломляясь, отдельные световые лучи приходят в точку наблюдения — глаз человека или оптическую систему камеры. Очевидно, что в точку наблюдения может попасть и часть первичных лучей непосредственно от источников излучения. Таким образом, изображение сцены формируется некоторым множеством световых лучей.

Цвет отдельных точек изображения определяется спектром и интенсивностью первичных лучей источников излучения, а также поглощением световой энергии в объектах, встретившихся на пути соответствующих лучей. Непосредственная реализация данной лучевой модели формирования изображения представляется затруднительной. Можно попробовать построить алгоритм построения изображения указанным способом. В таком алгоритме необходимо предусмотреть перебор всех первичных лучей и определить те из них, которые попадают в объекты и в камеру. Затем выполнить перебор всех вторичных лучей, и также учесть только те, которые попадают в объекты и в камеру. И так далее. В результате в камере формируется изображение сцены.

Такой метод называют *прямой* трассировкой лучей. Практическая ценность такого метода вызывает сомнения. Несмотря на физическую корректность, учитывать бесконечное множество лучей, идущих во все стороны в принципе невозможно. Даже если каким-то образом свести это к конечному числу операций (например, разделить всю сферу направлений на угловые секторы и оперировать уже не бесконечно тонкими линиями, а секторами), все

равно остается главный недостаток метода— много лишних операций, связанных с расчетом лучей которые затем не используются.

Метод ***обратной трассировки*** лучей позволяет значительно сократить перебор световых лучей. Метод разработан в 80-х годах, основополагающими считаются работы *Уиттеда* и *Кэя*. Согласно этому методу отслеживание лучей производится не от источников света, а в обратном направлении — точки наблюдения. Так учитываются только те лучи, которые вносят вклад; формирование изображения. Рассмотрим, как можно получить растровое изображение некоторой трёх мерной сцены методом обратной трассировки. Предположим, что плоскость проецирования разбита на множество квадратиков — пикселов. Выберем центральную проекцию с центром схода на некотором расстоянии от плоскости проецирования. Проведем прямую линию из центра схода через пиксел плоскости проецирования.

Это будет первичный луч обратной трассировки. Если прямая линия этого луча попадает один или несколько объектов сцены, то выбираем ближайшую точку пересечения. Для определения цвета пикселя изображения нужно учитывать свойства объекта, а также то, какое световое излучение приходится на соответствующую точку объекта.

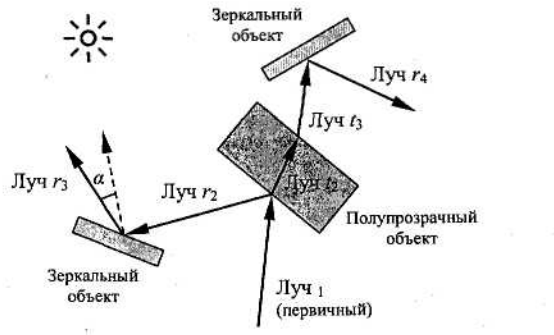


Рис. 5.8. Обратная трассировка для объектов, обладающих свойствами зеркального отражения и преломления.

Если объект зеркальный (хотя бы частично), то строим вторичный луч — луч падения, считая лучом отражения предыдущий, первичный трассируемый луч. Выше мы рассматривали зеркальное отражение и получили формулы для вектора отраженного луча по заданным векторам нормали и луча падения. Но здесь нам известен вектор отраженного луча, а как найти вектор падающего луча? Для этого можно использовать ту же самую формулу зеркального отражения, но определяя необходимый вектор луча падения как отраженный луч. То есть отражение наоборот.

Для идеального зеркала достаточно затем проследить лишь очередную точку пересечения вторичного луча с некоторым объектом. Что означает термин "идеальное зеркало"? Будем полагать, что у такого зеркала идеально ровная отполированная поверхность, поэтому одному отраженному лучу соответствует только один падающий луч. Зеркало может быть затемненным, то есть поглощать часть световой энергии, но все равно остается правило: один луч падает — один отражается. Можно рассматривать также "неидеальное зеркало". Это будет означать, что поверхность неровная. Один падающий луч порождает несколько отраженных лучей, образующих некоторый конус, возможно, несимметричный, с осью вдоль линии луча, отраженного от идеального зеркала. Конус соответствует некоторому закону распределения интенсивностей, простейший из которых описывается моделью Фонга — косинус угла, возведенный в некоторую степень (ее называют коэффициентом Фонга). Неидеальное зеркало резко усложняет трассировку— нужно проследить не один, а множество падающих лучей, учитывать вклад излучения от других видимых из данной точки объектов.

Если объект прозрачный, то необходимо построить новый луч, такой, который при преломлении давал бы предыдущий трассируемый луч. Здесь также можно воспользоваться обратимостью, которая справедлива и для преломления. Для расчета вектора искомого луча можно применить рассмотренные выше формулы для вектора луча преломления, считая, что преломление происходит в обратном направлении.

Если объект обладает свойствами диффузного отражения и преломления, то, в общем случае, как и для неидеального зеркала, необходимо трассировать лучи, приходящие от всех имеющихся объектов. Для диффузного отражения интенсивность отраженного света, как известно, пропорциональна косинусу угла между вектором луча от источника света и нормалью. Здесь источником света может выступать любой видимый из данной точки объект, способный передавать световую энергию.
Когда выясняется, что текущий луч обратной трассировки не пересекает какой-либо объект, а уходит в свободное пространство, то трассировка этого луча заканчивается.
Обратная трассировка лучей в том виде, в каком мы ее здесь рассмотрели, хоть и сокращает перебор, но не позволяет избавиться от бесконечного числа анализируемых лучей. В самом деле, данный метод позволяет сразу получить для каждой точки изображения единственный первичный луч обратной трассировки. Однако вторичных лучей отражения уже может быть бесконечное количество. Так, например, если объект может отражать свет от любого другого объекта, и если эти другие объекты имеют достаточно большие размеры, то для построения соответствующих лучей, например, при диффузном отражении, нужно учитывать все точки.
При практической реализации метода обратной трассировки вводят ограничения. Некоторые их них необходимы, чтобы можно было в принципе решить задачу синтеза изображения, а некоторые ограничения позволяют значительно повысить быстродействие трассировки.

Рассмотрим примеры таких ограничений:

1. Среди всех типов объектов выделим те, которые назовем источниками света. Источники света могут только излучать свет, но не могут его отражать или преломлять. Будем рассматривать только *точечные* источники света.
2. Свойства отражающих поверхностей описываются суммой двух компонент — диффузной и зеркальной.
3. В свою очередь, зеркальность также описывается двумя составляющими.Первая *(reflection)* учитывает отражение от других объектов, не являющихся источниками света. Строится только один зеркально отраженный луч ***r*** для дальнейшей трассировки. Вторая компонента *(specular)* означает световые блики от источников света. Для этого направляются лучи на все источники света и определяются углы, образуемые этими лучами с зеркально отраженным лучом обратной трассировки ***(r)***. При зеркальном отражении цвет точки поверхности определяется цветом того, что отражается. В простейшем случае зеркало не имеет собственного цвета поверхности.
4. При диффузном отражении учитываются только лучи от источников света. Лучи от зеркально отражающих поверхностей игнорируются. Если луч, направленный на данный источник света, закрывается другим объектом, значит, данная точка объекта находится в тени. При диффузном отражении цвет освещенной точки поверхности определяется собственным цветом поверхности и цветом источников света.
5. Для прозрачных *(transparent)* объектов обычно не учитывается зависимость коэффициента преломления от длины волны. Иногда прозрачность вообще моделируют без преломления, то есть направление преломленного луча *t* совпадает с направлением падающего луча.
6. Для учета освещенности объектов светом, рассеиваемым другими объектами, вводится фоновая составляющая *(ambient)*.
7. Для завершения трассировки вводят некоторое пороговое значение освещенности, которое уже не должно вносить вклад в результирующий цвет, либо ограничивают количество итераций. Возможны следующие случаи:
   - Луч попал в источник света. В этом случае цвет точки полностью определяется источником света.

- Луч попал в точку, лежащую на одном из объектов сцены. В такой ситуации пытаемся определить, откуда мог прийти свет, чтобы, попав в эту точку, отразиться (преломиться) в направлении исходного луча. Определив все возможные направления, выпустим вдоль каждого из них луч для определения фотонов, которые могли оттуда прийти.
- Луч не попал ни в источник света, ни в один из объектов. Тогда световая энергия, приносимая этим лучом, определяется окружающей сцену средой и в простейшем случае является постоянной величиной.

В первом и третьем случаях световая энергия, попадающая в данный пиксел, определяется сразу. Второй случай требует выпускания новых лучей и определения приносимой ими световой энергии. Это может, в свою очередь, потребовать выпускания новых лучей и т.д. Таким образом, процесс трассировки лучей является **рекурсивным**.

5.6 Модель освещенности Уиттеда

Для определения цвета (освещенности) произвольного пиксела экрана проведем луч Q из положения наблюдателя через этот пиксел до ближайшей точки пересечения **P** с одним из объектов сцены (попадание луча в точечный источник света можно игнорировать). Всю световую энергию, покидающую точку **P** в направлении, определяемом этим лучом, разобьем на две части – непосредственную (первичную) освещенность (световая энергия, падающая в эту точку непосредственно от источников света) и вторичную освещенность (световая энергия, отраженная и преломленная другими объектами).

Рисовать!

Обычно непосредственная освещенность вносит заметно больший вклад, нежели вторичная, поэтому и рассматриваются они по-разному.

Для определения непосредственной освещенности из точки **P** выпускаются лучи ко всем источникам света ($\mathbf{L}_i$), с целью проверки их видимости.

Вторичная освещенность, связанная с диффузным отражением, обычно просто игнорируется, поскольку для нее невозможно однозначно определить направление, откуда энергия могла прийти, чтобы затем отразиться в заданном направлении. Чтобы компенсировать это допущение, вводится так называемая **фоновая освещенность** - равномерное, не зависящее ни от чего освещение.

Для определения вторичной освещенности, получаемой зеркальным отражением и преломлением, в соответствующих направлениях выпускаются лучи (**R** и **T**) и определяется приносимая ими энергия $\boldsymbol{I_r}$ и $\boldsymbol{I_t}$ соответственно.

Одной из самых простых моделей освещенности является модель Уиттеда. Согласно этой модели световая энергия, покидающая точку в направлении вектора ***r*,** определяется формулой

$$I(\lambda)=K_a I_a(\lambda)+K_d C(\lambda)\Sigma_j I_j(\lambda)(n,l_j)+K_r I_r(\lambda)e^{-\beta_r d_r}+$$
$$+K_s\Sigma_j I_j(\lambda)(n,h_j)^p+K_t I_t(\lambda)e^{-\beta_t d_t}$$

где

$\boldsymbol{K_a, K_d, K_s}$ - веса фонового, диффузного и зеркального освещения;

$\boldsymbol{K_r, K_t}$ - веса отраженного и преломленного лучей;

$\boldsymbol{C(\lambda)}$ - цвет объекта в точке *P;*

$\boldsymbol{n}$ - вектор нормали в точке *P;*

$\boldsymbol{l_j}$ - направление на j-й источник света;

$\boldsymbol{I_j(\lambda)}$- интенсивность j-го источника света;

$\boldsymbol{\beta_r, \beta_t}$ - коэффициенты ослабления для сред, через которые проходит отраженный и преломленный лучи;

d_r, d_t - расстояния от точки P до ближайших точек пересечения отраженного и преломленного лучей с объектами сцены;

p - коэффициент Фонга.

Чтобы учесть ослабление интенсивностей излучений, приходящих по отраженному лучу I_r, а также по преломленному лучу I_t, умножают на коэффициент, учитывающий ослабление интенсивности в зависимости от расстояния, пройденного лучом. Такой коэффициент записывается в виде $e^{\beta d}$, где d — пройденное расстояние, β -параметр ослабления, учитывающий свойства среды, в которой распространяется луч.

6 Основы растровой графики

6.1 Основные характеристики растровой графики

Под *графической информацией* мы понимаем всю совокупность информации, которая нанесена на самые различные носители — бумагу, пленку, кальку, картон, холст и, с помощью кодирования, на цифровые носители.
Графическая информация бывает двух видов.

- *Растровая графика* (bitmaped images, scanned images, raster images) представляет собой совокупность точек (или пикселей), которые различаются только цветом (тоном) и взаимным расположением.
- *Векторная графика* (vector drawing, vector illustration) представляет собой линейно-контурное изображение,которое состоит из геометрических примитивов, таких как точки, линии, сплайны и полигоны, и их заполнения ("заливок").

Основными характеристиками растровой графики являются разрешение и глубина цвета
Разрешение — это количество точек (пикселей) на единицу длины. Понятие разрешения, однозначно связывает размер элемента дискретизации (пикселя) со стандартными единицами измерения, принятыми в науке и технике.
Разрешение представляет собой достаточно универсальное понятие, которое применяется в разных областях, имеющих дело с изображениями (например, в телевидении, полиграфии и компьютерной графике), оно хотя и имеет разные названия и разные формы единиц измерения, сохраняет единый смысл: количество дискретных элементов, приходящихся на стандартную единицу длины (фактически — на единицу площади).

> Оно представляет собой просто сокращение английских слов picture element, что означает "элемент изображения", "элемент картинки". Эти два слова сократили до "pic" и "el" и соединили по принципу "колхоз", получилось слово "pixel" (при этом буква "c" заменена на "x" по причине чередования, например как в русском языке происходит чередование согласных в словах "печка" и "пекарь").

Таким образом, пиксельная изображение, или растровая графика, т. е. матрица, которая представляет собой совокупность пикселей.
Отличительными особенностями пиксела являются его однородность (все пикселы по размеру одинаковы) и неделимость (пиксел не содержит более мелких пикселов).
Единица измерения разрешения ppi — это количество пикселов в каждом дюйме изображения. В настоящее время более популярной единицей разрешения является **dpi**, пришедшее из полиграфии и ставшее повсеместно используемым в компьютерной графике.
Большинство редакторов растровой графики отображают пикселы изображения при помощи пикселов экрана (без учета физических размеров): если, предположим, разрешение документа совпадает с разрешением экрана (так стараются делать, если изображение предназначено только для экрана, например компьютерная презентация, компьютерная заставка, изображение для Web-страницы и т. д.), то изображение на экране будет отображаться "пиксел в пиксел", а это означает масштаб 100%.Если у документа разрешение превышает экранное, то при масштабе 100% документ будет отображаться в несколько раз увеличенным. Например, изображение с разрешением 144 ppi на экране монитора будет в два раза больше, чем изображение с разрешением 72 ppi, даже если их физический размер (например, 1x1 дюйм) будет одинаковым.
Изображение с большим разрешением содержит больше пикселей, которые имеют меньший размер, чем у изображения с меньшим разрешением, у которого пикселей имеют больший размер.
С практической точки зрения это означает, что изображение с высоким разрешением может быть распечатано в большем формате без потери качества.
Например, в одном квадратном дюйме изображение, предназначенное для вывода на экран монитора с разрешением 96 ppi, содержит 9216 пикселов. В том же квадратном дюйме изображение, предназначенное для вывода на лазерный принтер с разрешением 600 dpi, содержит 360 000 пикселов. Очевидно, что во втором случае физический размер точки будет в 70 раз меньше.

Однако следует очень четко уяснить, что **уровень качества изображения** закладывается в процессе фотографирования или сканирования в зависимости от устанавливаемого разрешения. Последующее увеличение разрешения цифрового изображения в любом графическом редакторе, не способствует улучшению качества изображения. Это связано с тем, что ПО, конечно, не способно добавить новую изобразительную информацию (добавить новые более мелкие детали), а только перераспределить уже имеющиеся данные на большее число пикселей. В этом случае, как правило, даже происходит ухудшение некоторых параметров изображения, например резкости.
Правильный выбор величины разрешения зависит от многих факторов: назначения изображения и способа его использования. Необходимо найти разумный баланс между качеством, размером файла и временем его обработки, а также учесть возможности системы обработки.

Основной *цветовой* характеристикой растрового изображения является **глубина цвета.**
Цветные изображения составляют в настоящий период подавляющее большинство изображений. Однако цвет представляет массу проблем с точки зрения технологии его использования.
Для того чтобы оцифровать и сохранить цветовую информацию, все технические системы используют цветную фильтрацию: пропускают цветовой поток через три цветных фильтра (красный, зеленый и синий). В результате получается 3 числа, характеризующих интенсивность красной, зеленой и синей составляющей. Совмещение тоновых градаций всех составляющих (каналов) и определяет цвет изображения.
Т.о. каждый пиксель описывается восемью двоичными разрядами, в сумме это составит 24 бита, т. е. полноцветные изображения называются "24-bit image". Это дает возможность закодировать 16,7 млн. оттенков, что достаточно много. Поэтому данную модель иногда называют TrueColor (Истинный цвет). Такое изображение называется по имени цветовой модели — "RGB-image" (изображение в цветовой модели RGB).
Цветовые составляющие в программе организуются в виде так называемых каналов, каждый из которых представляет собой изображение в градациях серого со значениями яркости от 0 до 255.

Из этих определений следует, что хотя разрешение и глубина цвета друг с другом существуют неразрывно (не бывает изображений с разрешением, но без глубины цвета и наоборот), фактически они никак не связаны.

6.2 Виды изображений

Изображения бывают *векторными* и *растровыми*. Векторным называется изображение, описанное в виде набора графических примитивов. Растровые же изображения представляют собой двумерный массив, элементы которого (пикселы) содержат информацию о цвете. В цифровой обработке используются растровые изображения. Они в свою очередь делятся на типы - *бинарные*, *полутоновые*, *палитровые*, *полноцветные*.

Элементы *бинарного* изображения могут принимать только два значения: 0 или 1. Природа происхождения таких изображений может быть самой разнообразной. Но в большинстве случаев, они получаются в результате обработки полутоновых, палитровых или полноцветных изображений методами бинаризации с фиксированным или адаптивным порогом. Бинарные изображения имеют то преимущество, что они очень удобны при передаче данных.

Полутоновое изображение состоит из элементов, которые могут принимать одно из значений интенсивности какого-либо одного цвета. Это один из наиболее распространенных типов изображений, который применяется при различного рода исследованиях. В большинстве случаев используется глубина цвета 8 бит на элемент изображения.

В палитровых изображениях значение пикселов является ссылкой на ячейку карты цветов (палитру). Палитра представляет собой двумерный массив, в столбцах которого расположены интенсивности цветовых составляющих одного цвета.

> Первые цветные мониторы работали с ограниченным цветовым диапазоном: сначала 16, затем 256 цветов. Они кодировались, соответственно, 4 битами (16 цветов) и 8 битами (256 цветов). Такие цвета называются *индексированными* (indexed colors).
> Индексированные цвета кодируются в виде так называемых *цветовых таблиц* (color lookup table, LUT), т. е. серий таблиц цветовых ссылок (индексов, откуда произошло название индексированных цветов). В этой таблице цвета уже предопределены как мелки в коробке пастели.
> Несмотря на ограниченность палитры индексированных цветов с ними продолжают активно работать, например они используются в изображениях для Web-страниц. В этом режиме обеспечивается удачное сочетание параметров: небольшой размер графического файла, что предпочтительно для времени передачи, и относительно большой выбор цветов, что предпочтительно для пользователя.
> При преобразовании полноцветного изображения в индексированное (редуцирование цветовой палитры) возможны различные режимы, которые позволят выбрать, какие потери наиболее допустимы.

В отличии от палитровых, элементы *полноцветных* изображений непосредственно хранят информацию о яркостях цветовых составляющих.

Выбор типа изображения зависит от решаемой задачи, от того, насколько полно и без потерь нужная информация может быть представлена с заданной глубиной цвета. Также следует учесть, что использование полноцветных изображений требует больших вычислительных затрат.

6.3Форматы графических файлов

Рассмотрим несколько распространенных графических форматов и кратко охарактеризуем их возможности. Все эти сведения сведены в нижеследующую таблицу:

Формат	Макс. глуб. цвета	Макс. число цветов	Макс. размер изображения, пиксел	Методы сжатия	Кодирование нескольких изображений
BMP	24	16'777'216	65535x65535	RLE	-
GIF	8	256	65535x65535	LZW	+
JPEG	24	16'777'216	65535x65535	JPEG	-
PCX	24	16'777'216	65535x65535	RLE	-
PNG	48	281'474'976'710'656	2147483647x 2147483647	Deflation (LZ77)	-
TIFF	24	16'777'216	всего 4'294'967'295	LZW, RLE и другие*	+

Заметим, что глубокое представление цвета (например 32 бит/пиксел) реально оказывается практически неотличимым от данного при просмотре на современных мониторах и при распечатке на большинстве доступных принтеров. Такая глубина цвета может оказаться полезной разве только в издательской деятельности.

Кроме этого следует отметить, что форматы JPEG, GIF, PNG, являются платформо-независимыми. TIFF является частично платформенно-независимым, однако слишком объемен для использования в сети и, что еще хуже, слишком сложен для интерпретации.

6.3.1 Растровые графические форматы

Самый простой растровый **формат BMP** является родным форматом Windows, он поддерживается всеми графическими редакторами, работающими под ее управлением. В BMP данные о цвете хранятся только в модели RGB, поддерживаются как индексированные цвета (до 256 цветов), так и полноцветные изображения, причем в режиме индексированных цветов возможна простейшая компрессия RLE (Run Length Encoding - кодирование с

переменной длиной строки). Без компрессии размер файла оказывается близок к максимально возможному. Применяется для хранения растровых изображений, предназначенных для использования в Windows и, по сути, больше ни на что не пригоден. Использование BMP не для нужд Windows является распространенной ошибкой новичков: использовать BMP нельзя ни в web, ни для печати (особенно), ни для простого переноса и хранения информации.

Примерно такими же возможностями, как BMP, обладает и **формат PCX**, разработанный еще на заре компьютерной эпохи фирмой Z-Soft специально для своего графического редактора PC PaintBrush под операционную систему MS-DOS, отсутствует только поддержка операционной системы OS/2. Зато изображения в формате PCX можно посмотреть большинством программ под DOS, в том числе и внутренним просмотрщиком Norton Commander. Цветовые возможности: 1, 2, 4, 8 или 24- битовый цвет, поддерживается только схема RGB, причем полностью отсутствуют возможности сохранения монохромного изображения в оттенках серого. Всегда применяется сжатие ROB. Как и BMP, этот формат в значительной мере устарел и поддерживается современными графическими программами исключительно для совместимости с антикварным софтом.

Формат GIF, разработанный компанией CompuServe, и изначально предлагавшийся именно как формат для обмена изображениями в сети является форматом с достаточно высокой степенью сжатия изображения. Небольшие размеры файлов изображений обусловлены применением алгоритма сжатия без потерь качества LZW, благодаря чему изображения в этом формате наиболее удобны для пересылки по все еще узким каналам связи глобальной сети. Кроме того, GIF обладает дополнительными возможностями, которые делают его использование в сети привлекательным.

- использование режима индексированных цветов (не более 256), что ограничивает область применения формата изображениями, имеющими резкие цветовые переходы;
- возможность изменения порядка вывода на экран строк изображения, с заполнением промежутков между ними временной информацией. Визуально это выглядит так, что по мере загрузки из сети (что происходит нередко с катастрофически низкой скоростью) изображение на экране появляется как бы "в низком качестве", а затем, по мере подгрузки дополнительной информации, восстанавливает пропущенные строки изображения;
- возможность хранении в одном файле более, чем одного изображения, что делает возможной элементарную покадровую анимацию;
- возможность, один из цветов объявить "прозрачным", и тогда при выводе изображения те его части, которые выкрашены этим цветом не будут выводиться на экран и под ними будет виден фон, на который изображение накладывается;

Основное ограничение формата GIF состоит в том, что цветное изображение может быть записано только в режиме 256 цветов, что в последнее время становиться все менее и менее приемлемым. Для полиграфии этого явно недостаточно.

Формат PNG, являющийся плодом трудов сообщества независимых программистов, появился на свет как ответная реакция на переход популярнейшего формата GIF в разряд коммерческих продуктов. Этот формат, сжимающий графическую информацию без потерь качества, используя алгоритм Deflate, в отличие от GIF или TIFF сжимает растровые изображения не только по горизонтали, но и по вертикали, что обеспечивает более высокую степень сжатия и поддерживает цветные фотографические изображения вплоть до 48-битных включительно. Как недостаток формата часто упоминается то, что он не дает возможности создавать анимационные ролики, хотя сейчас, при повальном переходе практически всей анимации на технологию Flash, это уже совсем не актуально. Зато формат PNG позволяет создавать изображения с 256 уровнями прозрачности за счет применения дополнительного альфа-канала с 256 градациями серого что, безусловно, выделяет его на фоне всех

существующих в данный момент форматов. В числе других отличительных особенностей этого формата можно отметить двумерную чересстрочную развертку (т.е. изображение проявляется постепенно не только по строкам, но и по столбцам) и встроенную гамма-коррекцию, позволяющую сохранять изображения, яркость которых будет неизменна не только на любых машинах PC, но и на таких альтернативных платформах, как Mac, Sun или Silicon Graphics. Так как формат создавался для интернета, в его заголовке не предназначено место для дополнительных параметров типа разрешения, поэтому для хранения изображений, подлежащих печати, PNG плохо подходит, для этих целей лучше подойдет PSD или TIFF. Зато он хорош для публикации высококачественной растровой графики в интернете.

Но широкое распространение этого, поистине передового формата сдерживают и некоторые его недостатки. Так, формат PNG значительно уступает своему предшественнику, GIF-у, в тех случаях, когда речь идет о мелких элементах оформления веб-страниц, таких, как кнопки, рамки и т.п. Проблема заключается в том, что в файле изображения около 1 Кб занимает описание палитры цветов, что порой бывает сопоставимо с размером самого изображения. Существуют 2 основных типа формата:

- PNG8 - для рисунков и фотографий в моделях Grayscale и Indexed (недеструктивное сжатие);
- PNG24 - для рисунков и фотографий в модели RGB (недеструктивное сжатие).

Самый популярный формат для хранения фотографических изображений **JPEG (или JPG)** является общепризнанным стандартом в интернете. JPEG может хранить только 24-битовые полноцветные изображения. Одноименный с форматом, достаточно сложный алгоритм сжатия основан на особенностях человеческого зрения (используется представление блока пикселей 8x8 одним цветом с сохранением информации о яркости плюс метод Хаффмана и, в зависимости от степени компрессии, некоторые другие ухищрения).

Хотя JPEG отлично сжимает фотографии, но это сжатие происходит с потерями и портит качество, тем не менее, он может быть легко настроен на минимальные, практически незаметные для человеческого глаза, потери. Однако не стоит использовать формат JPEG для хранения изображений, подлежащих последующей обработке, так как при каждом сохранении документа в этом формате процесс ухудшения качества изображения носит лавинообразный характер. Наиболее целесообразно будет корректировать изображение в каком-нибудь другом подходящем формате, например TIFF, и лишь по завершению всех работ окончательная версия может быть сохранена в JPEG. Таким образом, можно сохранить вполне приемлемое качество изображения при минимальном размере итогового файла.

Формат JPEG пригоден в подавляющем большинстве случаев только для публикации полноцветных изображений, типа фотографических, в интернете.

Новый открытый стандарт **JPEG 2000**, в разработке которого приняли участие Международная организация по стандартизации (International Organization for Standardization), Международный союз телекоммуникаций (International Telecommunications Union), компании Agfa, Canon, Fujifilm, Hewlett-Packard, Kodak, LuraTech, Motorola, Ricoh, Sony и др., обещает разрубить этот гордиев узел и с лихвой удовлетворить требованиям, предъявляемым к изображениям в различных сферах современного производства (издательском деле, цифровой фотографии, Internet-технологиях и проч.).

JPEG2000 позволяет сжимать изображения в 200 раз без заметной для глаза человека потери качества. Основным отличием JPEG2000 от предыдущей версии этого формата является сжатие с использованием алгоритма волнового преобразования (изображение описывается с помощью математических выражений как непрерывный поток) вместо преобразования Фурье, что и позволяет предотвратить появление характерных блоков. Допустимо также без

ущерба модифицировать (масштабировать, редактировать) рисунок, сохраненный в этом формате.

Алгоритм волнового преобразования позволяет просматривать и распечатывать одно и то же изображение при различных (заданных пользователем) значениях разрешения и с требуемой степенью детализации. Благодаря этой особенности JPEG2000, очевидно, быстро найдет свое место в сети Internet, поскольку обеспечит возможность загружать картинку с разными значениями разрешения в зависимости от пропускной способности конкретного канала связи. Да и тот факт, что пользователи Internet смогут получать изображения высокого качества, немаловажен.

Еще одно значимое преимущество JPEG2000 — возможность управлять 256 цветовыми каналами, что позволит получать качественные цветные изображения.

Формат TIFF был разработан компанией Aldus для своего графического редактора PhotoStyler, впрочем, уже почившего в бозе, однако самому формату была уготована гораздо более долгая жизнь. Как универсальный формат для хранения растровых изображений, TIFF достаточно широко используется, в первую очередь, в издательских системах, требующих изображения наилучшего качества. Кстати, возможность записи изображений в формате TIFF является одним из признаков высокого класса современных цифровых фотокамер.

Благодаря своей совместимости с большинством профессионального ПО для обработки изображений, формат TIFF очень удобен при переносе изображений между компьютерами различных типов (например, с PC на Mac и обратно).

Формат PSD является стандартным форматом пакета Adobe Photoshop и отличается от большинства обычных растровых форматов возможностью хранения слоев (layers). Он содержит много дополнительных переменных (не уступает TIFF по их количеству) и сжимает изображения, используя алгоритм сжатия без потерь RLE Packbits, иногда даже сильнее, чем PNG (только в тех случаях, когда размеры файла измеряются не в килобайтах, а в десятках или даже сотнях мегабайт). Формат поддерживает глубины цвета, вплоть до 16 бит на канал (48-битные цветные и 16-битные черно-белые), а также альфа-каналы, слои, контуры, прозрачность, векторные надписи и т. п. Прекрасно подойдет для переноса или хранения изображений, содержащих специфические, свойственные только Adobe Photoshop, элементы. Файлы PSD свободно читаются большинством популярных просмотрщиков, но не стоит забывать, что, открыв эти файлы в некоторых графических редакторах третьих фирм, даже декларирующих поддержку формата PSD, можно потерять значительную часть их специфических возможностей (особенно в части работы со слоями, см. серию наших статей "Adobe Photoshop, первые шаги").

6.3.2 Векторные графические форматы

Среди векторных форматов, в отличие от растровых, идея хоть какой-то разумной стандартизации проявляется значительно слабее. Разработчики практически всех векторных графических программ предпочитают иметь дело только со своими собственными форматами, что связано, скорее всего, со спецификой алгоритмов формирования векторного изображения. Но, так как возможность переноса файлов между различными приложениями в векторной графике не менее актуальна, чем в растровой, то своего рода стандартом стали файловые форматы двух наиболее популярных профессиональных графических пакетов - Adobe Illustrator и CorelDRAW.

Первый из них, **AI (Adobe Illustrator),** поддерживают практически все программы, так или иначе связанные с векторной графикой. Этот формат является наилучшим посредником при передаче изображений из одной программы в другую, с PC на Macintosh и наоборот. В целом, несколько уступая CorelDRAW по иллюстративным возможностям, (может содержать

в одном файле только одну страницу, имеет маленькое рабочее поле - этот параметр очень важен для наружной рекламы - всего 3x3 метра) тем не менее, он отличается наибольшей стабильностью и совместимостью с языком PostScript, на который ориентируются практически все издательско-полиграфические приложения.

Довольно противоречивым является **формат CDR**, основной рабочий формат популярного пакета CorelDRAW, являющимся неоспоримым лидером в классе векторных графических редакторов на платформе PC. Имея сравнительно невысокую устойчивость и проблемы с совместимостью файлов разных версий формата, тем не менее формат CDR, особенно последних, 7-й и 8-й версий, можно без натяжек назвать профессиональным. В файлах этих версий применяется раздельная компрессия для векторных и растровых изображений, могут внедряться шрифты, файлы CDR имеют огромное рабочее поле 45x45 метров, поддерживается многостраничность.

WMF - еще один родной формат Windows, на сей раз векторный. Понимается практически всеми программами Windows, так или иначе связанными с векторной графикой. Однако, несмотря на кажущуюся простоту и универсальность, пользоваться форматом WMF стоит только в крайних случаях, поскольку он не может сохранять некоторые параметры, которые могут быть присвоены объектам в различных векторных редакторах, не воспринимается Macintosh-ами, и, самое главное, способен исказить цветовую схему изображения.

6.4 Простейшая обработка фотоизображения.

При работе с изображениями часто возникает необходимость каким-то особым образом изменить его некоторые характеристики. Например, можно сократить выдержку фотоизображения, уменьшив цветовые значения пикселов. При необходимости, красную, зеленую и синюю компоненты можно изменять раздельно, чтобы получить наилучший цветовой баланс. Задача цветокоррекции применяется и в других областях: создание текстур при моделировании объектов, выравнивание гистограмм изображений и многое другое. Все это порождает множество постановок задачи цветокорекции. В то же время заметим, что не все из перечисленных преобразваний легко выполнимы алгоритмически.
Ниже будут рассмотрены простейшие задачи по преобразованию изображений фотографического качества.

Яркость точки и гистограммы изображения. Яркость точки находится по формуле, коэффициенты которой определяются свойствами человеческого зрения:

$$Y=0,299R + 0,5876G + 0,114B$$

Гистограммой в данном случае называется так или иначе представленная (например, в виде столбчатой диаграммы) зависимость числа повторений того/иного значения яркости на всём изображении от этого самого значения (то есть сколько раз встречается абсолютно чёрная точка, абсолютно белая и др.); при этом можно рассматривать 4 гистограммы: по 3 каналам и по вычисленной яркости.

Изменения яркости и контрастности. Эти изменения можно обобщить выражением «изменение баланса изображения», так как оба понятия – и яркость, и контрастность, – схожи и относятся к сфере восприятия изображения человеком.
Повышение/снижение яркости – это, соответственно, сложение/вычитание значения каждого канала с некоторым фиксированным значением (также в пределах от 0 до 255); при этом обязательно необходимо контролировать выход нового значения канала за пределы диапазона 0..255.
Повышение/снижение контрастности – это, соответственно, умножение/деление значения каждого канала на некоторое фиксированное значение (в том числе действительное), что

приводит к изменению соотношений между цветами и, соответственно, к более чётким цветовым границам. На практике же существует такой принцип: изменение контрастности не должно приводить к изменению средней яркости по изображению, поэтому пользуются следующей формулой:
$Y=K*(Y_{old}-Y_{av})+ Y_{av}$,
где Y – новое значение одного из каналов, K – коэффициент контрастности (K<1 – снижение, K>1 – повышение контрастности), OldY – текущее значение того же канала, AveY – среднее значение того же канала по изображению (таким образом, алгоритм фактически является двухпроходовым). Обязательна всё та же коррекция нового значения при выходе его за границы 0..255.

Изменение цветности. Под изменением цветности здесь понимается изменение спектра цветов, используемых в изображении. Минимальные преобразования – бинаризация, оттенки серого и получение негатива.

Бинаризация – это преобразование изображения, в общем случае, к одноцветному (чаще всего к черно-белому). В терминах Photoshop это ещё называется «по уровню 50%», так как при этом выбирается некий порог (например, посередине), все значения ниже которого превращаются в цвет фона, а выше – в основной цвет. Само преобразование можно осуществлять по каналам, но в этом случае результирующее изображение не будет в прямом смысле бинарным (чёрно-белым), а будет содержать 8 чистых цветов, представляющих собой комбинации чистых красного, зелёного и голубого цветов, то есть будет бинарным по каналам. Поэтому лучше проводить преобразование над «полным» цветом точки.

Преобразование к оттенкам серого заключается в получении яркости каждой точки по известной формуле и последующем копировании полученного значения по все три канала (R=G=B=Y).

И, наконец, негатив получается простой заменой значения каждого канала на его дополнение до 255 (например, R=255-R).

7 Цвет в компьютерной графике

Точное описание цвета имеет большое коммерческое значение. Многие виды продукции тесно связаны со специальными цветами (например, цвет различных популярных продуктов или цвет коробок), и производители готовы многое отдать, чтобы убедиться, что две различные партии товара имеют один и тот же цвет. Здесь нужна стандартная система цветов, поскольку простых названий цветов недостаточно: относительно немногие люди знают разнообразные названия цветов, кроме того, большинство людей по-разному трактуют названия цветов.

Существует естественный механизм передачи цвета: нужно договориться о стандартном наборе основных цветов, а затем описывать любое цветное освещение весовыми коэффициентами.

Стандартизацией цветовых пространств занимается МКО — Международная комиссия по освещению (*CIE - Commission International d'Eclairage)*.

7.1 Линейные цветовые пространства

Цветовое пространство представляет собой модель представления цвета, основанную на использовании цветовых координат. Цветовое пространство строится таким образом, чтобы любой цвет был представим точкой, имеющей определённые координаты, причём так, чтобы одному набору координат соответствовал один цвет.

Очевидным решением было бы использование в качестве координат коэффициентов основных цветов в эксперименте подбора цвета. Если для каждой длины волны λ подобрать коэффициенты основных цветов A, B и C, мы получим некоторые зависимости a(λ), b(λ), c(λ), которые называются функциями подбора цвета. Любой чистый (монохромный) цвет теперь можно будет найти как Цвет(λ) = a(λ) + b(λ) + c(λ).

Если в качестве основных цветов выбрать красный, зеленый и синий с длинами волн 645.16 нм, 526.32 нм и 444.44 нм соответственно, функции подбора цвета для среднего наблюдателя примут следующий вид:

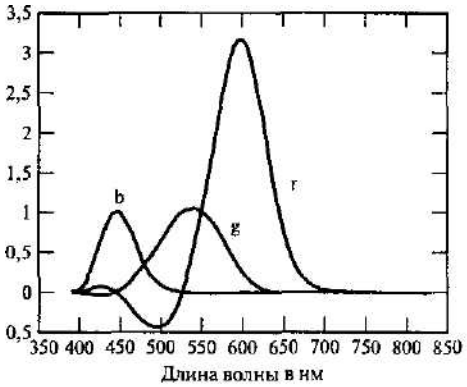


Рис.7.1. Функции (коэффициенты) подбора цвета для основных цветов RGB. Отрицательные значения указывают на то, что для подбора цвета на этой длине волны с основными цветами красный-зеленый-синий необходимо пользоваться разностным соотношением.

Неприятный факт: если основные цвета — это реальное освещение, то на определенной длине волны по крайней мере одна из функций подбора цвета будет отрицательной (рис. 7.1). Это не противоречит законам природы, а просто говорит о том, что для подбора некоторых видов освещения необходимо пользоваться разностным соответствием независимо от принятого набора основных цветов.

Этой проблемы можно избежать, если, например, определить функции подбора цвета, которые будут всегда положительными (при этом основные цвета будут нереальными, поскольку на определенной длине волны их спектральное излучение будет отрицательным). Хотя это и кажется проблематичным — как можно составить настоящий цвет из нереальных основных цветов? — на самом деле это не так, поскольку в системах именования цветов эта схема не применяется.

Цветовое пространство XYZ согласно стандартам МКО

Самым распространенным стандартом является *цветовое пространство XYZ, определенное МКО.* Оно построено на основе зрительных возможностей так называемого *Стандартного Наблюдателя*, то есть гипотетического зрителя, возможности которого были тщательно изучены и зафиксированы в ходе проведенных комитетом МКО длительных исследований человеческого зрения.

Функции подбора цвета выбраны так, чтобы они были всегда положительными (рис.7.2); поэтому координаты любого реального цвета в этом пространстве всегда положительны.

Хотя, сами X, Y и Z не являются реальными цветами, подобраны они так, что любой реальный цвет представим в виде их линейной комбинации с неотрицательными коэффициентами. Теперь любой реальный цвет мы можем представить вектором в пространстве XYZ, но далеко не любой вектор в этом пространстве представляет реальный цвет.

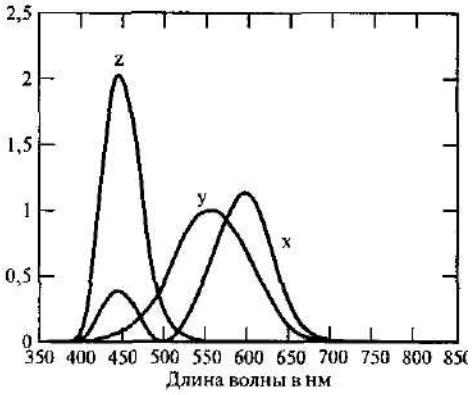


Рис 7.2. Функции подбора цветов для системы МКО с основными цветами X, Y и Z; эти функции всегда положительны, но основные цвета не являются реальными.

Эксперимент МКО по подбору цветовой пары

Для выработки модели Стандартного Наблюдателя МКО проводила серии следующих тестов.

Три источника света были направлены на левую часть белого экрана таким образом, что их цвет смешивался. Экспериментаторы сидели перед экраном, им давался пульт управления, состоящий из трех ручек, позволяющих управлять яркостью каждого из трех источников света. Затем на правой стороне экрана показывалась точка некоторого заданного цвета. Задача состояла в том, чтобы настроить ручки таким образом, чтобы цвета двух точек совпали. Лампы были настроены таким образом, чтобы интенсивность каждой контролировалась числом от –1 до 1. При 1 лампа включалась на полную мощность. Если ручка стояла на нуле, лампа была выключена. В положении меньше нуля свет лампы «вычитался» из результирующего. Это достигалось путем увеличения соответствующей компоненты яркости правой точки. Хотя результаты разных людей были разными, усредненный результат при заданных длинах волн для каждой из трех монохроматических ламп (эти длины волн примерно соответствовали пикам чувствительности разных типов колбочек) был принят за гипотетического стандартного наблюдателя.

Т.е. по спектру можно получить весовые коэффициенты XYZ, лежащие в пределах [-1, 1], и наоборот. В то время, когда разрабатывалась модель XYZ, использовать отрицательные числа было неудобно (в основном использовались расчеты вручную), поэтому была разработана модифицированная система с гипотетическими источниками света, в которой все коэффициенты X,Y и Z положительны. Эта система и

стала стандартом для CIE XYZ 1931 года. Недостатком этой системы является то, что не всем точкам в пространстве XYZ соответствуют реальные цвета в силу неортогональности функций соответствия цветов.

В прямоугольной системе координат XYZ область видимых человеческим глазом цветов представляет собой конус со сложным основанием. Его вершина находится в начале координат. Как правило, для упрощения подбора цвета удобно снизить его яркость (это можно делать, поскольку восприятие цвета почти линейно) — изображенный конус разрезается плоскостью X + Y + Z = 1, в результате чего получается *xy*-пространство МКО (рис 7.3), которое принято изображать в координатах (x,y).

$$(x, y) = \left(\frac{X}{X+Y+Z}, \frac{Y}{X+Y+Z} \right)$$

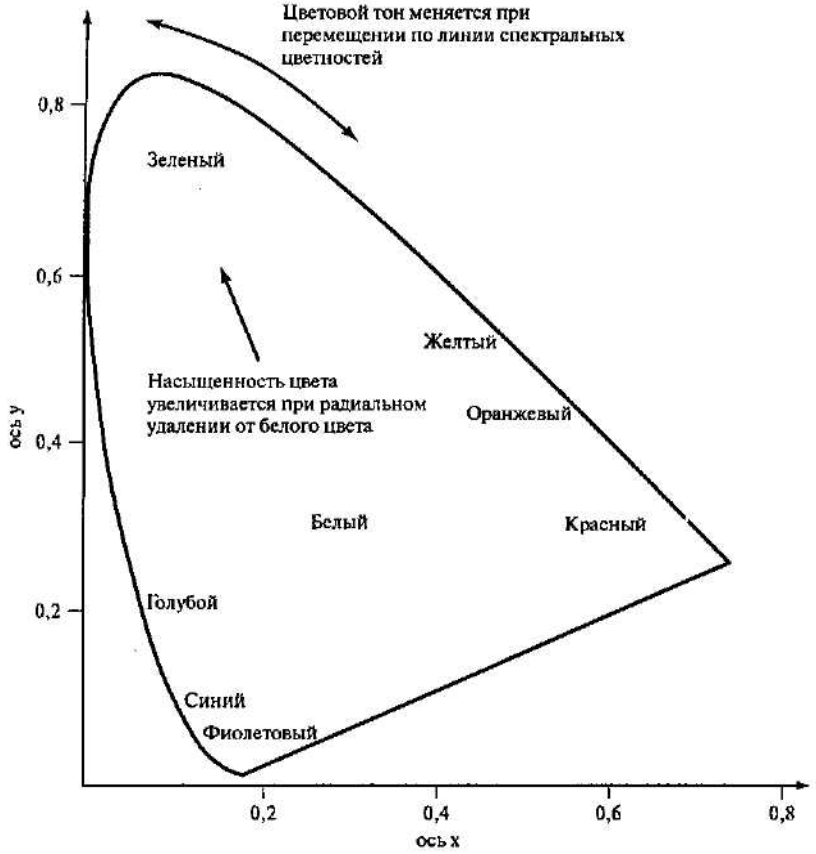


Рис. 7.3. На рисунке показано сечение постоянной яркости стандарта МКО 1931 цветового пространства *(xy)*. Обозначены названия цветов. Как правило, цвета, которые находятся дальше от нейтральной точки, насыщеннее (так темно-красный отличается от бледно-розового) и при перемещении вокруг нейтральной точки меняется их оттенок (от красного до зеленого)

Пространство МКО *(xy)* широко используется в учебниках по компьютерному зрению и компьютерной графике, а также в некоторых прикладных областях, но профессионалы, работающие с цветом, обычно называют этот подход несовременным.

Цветовое пространство RGB

Цветовые пространства обычно предназначены для практических целей, поэтому придумано их много. Одним из линейных цветовых пространств является *пространство RGB,* в котором для каждого основного цвета формально определена длина волны (645,16 нм для красного, 526,32 нм для зеленого и 444,44 нм для синего; см. рис. 1.1). Неформально в системе RGB используются все цвета, какие могут появиться на экране в качестве основных. Имеющиеся цвета обычно можно представить единичным *RGB-кубом,* грани

которого соответствуют весовым коэффициентам красного, зеленого и синего цвета. Этот куб показан на рис. 1.4

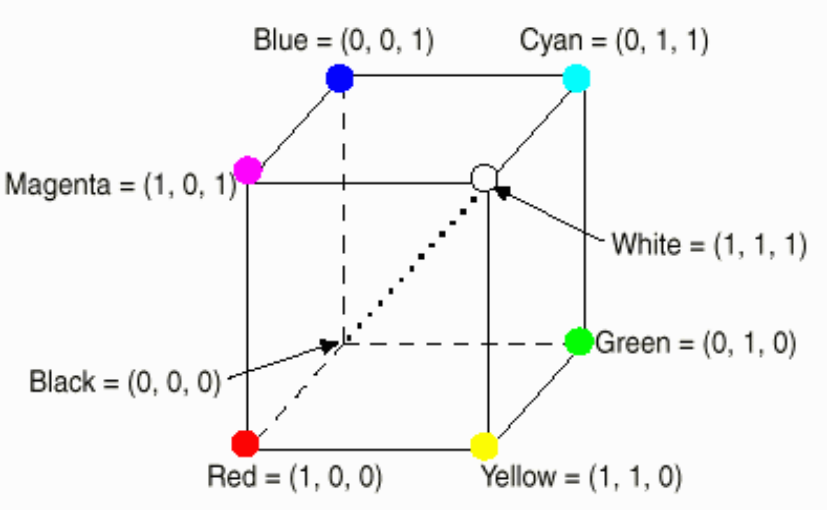


Рис. 7.4. RGB-куб — пространство всех цветов, которые получаются из комбинаций трех основных цветов (красного, зеленого и синего — это обычно определяется цветной реакцией монитора) с весовыми коэффициентами от нуля до единицы. Это аддитивная цветовая модель: для получения искомого цвета базовые цвета в ней складываются. Главная диагональ куба, характеризуемая равным вкладом трех базовых цветов, представляет серые цвета: от черного (0, 0, 0) до белого – (1, 1, 1)

На основе *R*, *G* и *B* часто вводится величина:

$Y=0,299R + 0,5876G + 0,114B$,

называемая в компьютерной графике *яркостью*.

CMY и черный цвет

Как только мы начинаем рисовать, интуиция подсказывает, что основными цветами должны быть красный, желтый и синий, и что желтый можно получить, смешав красный и зеленый. Это интуитивное представление неприменимо к мониторам, поскольку здесь речь идет о пигментах, которые смешиваются по разностному принципу, а не об освещении. Пигмент частично поглощает падающий свет, забирая некоторые составляющие, в результате чего отраженный свет приобретает определенный цвет. Таким образом, красные чернила — это на самом деле краситель, который поглощает синий и зеленый цвет, а падающий красный свет проходит сквозь краситель и отражается от бумаги.

Цветовые пространства с таким типом разностного соответствия могут быть очень сложными. В простейшем случае смешивание цветов линейно (или достаточно близко к линейному), и тогда можно пользоваться *пространством CMY*. В этом пространстве есть три основных цвета: *cyan* (оттенок голубого), *magenta* (оттенок пурпурного) и *yellow* (желтый). Эти основные цвета можно считать результатом отнимания базового цвета от белого; голубой (*C*) — это *W* — *R* (белый - красный); пурпурный (*M*) — это *W* — *G* (белый — зеленый), а желтый (*Y*) — это *W*—*B* (белый —синий):

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

где интенсивность (R , G , B) меняется в диапазоне от 0 до 1.

Теперь то, как выглядят комбинации цветов, можно оценивать относительно цветового пространства RGB. Например, синий – это комбинация голубого и пурпурного цветов:

$$(W\text{-}R) + (W - G) = R + G + B - R - G = B,$$

Обратите внимание, что $W+W=W$, поскольку считается, что чернила не могут повлиять на бумагу, чтобы она отражала больше света, чем чистая бумага.

По целому ряду причин (большой расход дорогостоящих цветных чернил, высокая влажность бумаги, получаемая при печати на струйных принтерах, нежелательные визуальные эффекты, возникающие за счет того, что при выводе точки трех базовых цветов ложатся с небольшими отклонениями) использование трех красителей для получения черного цвета оказывается неудобным. Поэтому его просто добавляют к трем базовым цветам. Так получается модель CMYK - Cyan, Magenta, Yellow, blacK.

Для перехода от модели CMY к модели CMYK используют следующие соотношения:

$$K = \min(C, M, Y)$$

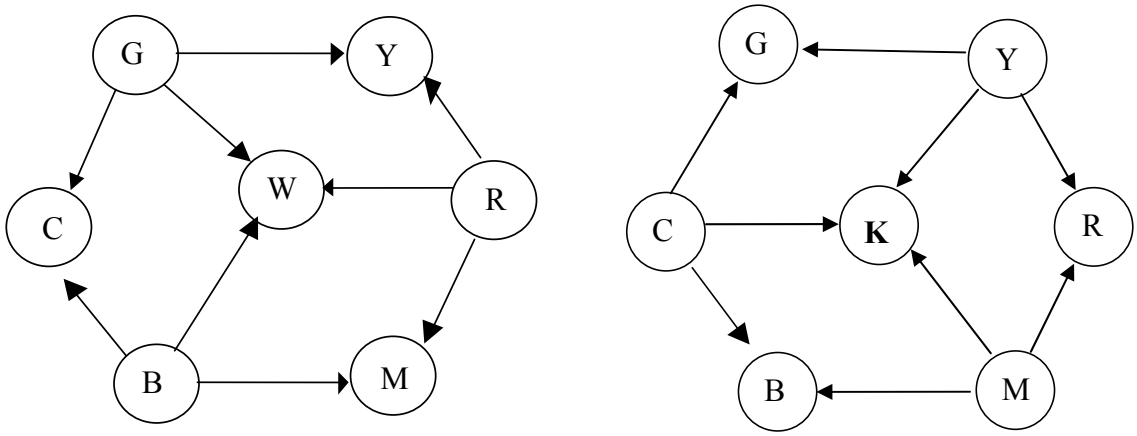
$$C = C - K$$

$$M = M - K$$

$$Y = Y - K$$

Схематически, линейные системы цветов можно представить в виде следующих схем:

Аддитивная система(RGB) **Субтрактивная система (CMYK)**



Если на поверхность бумаги наносится голубой (**C**) цвет, то красный цвет, падающий на бумагу, полностью поглощается. Таким образом, голубой краситель вычитает красный цвет из падающего белого (являющегося суммой красного, зеленого и синего цветов). Аналогично малиновый краситель (**M**) поглощает зеленый, а желтый краситель – синий цвет. Поверхность, покрытая голубым и желтым красителями, поглощает красный и синий, оставляя только зеленую компоненту. Голубой, желтый и малиновый красители поглощают красный, зеленый и синий цвета, оставляя в результате черный.

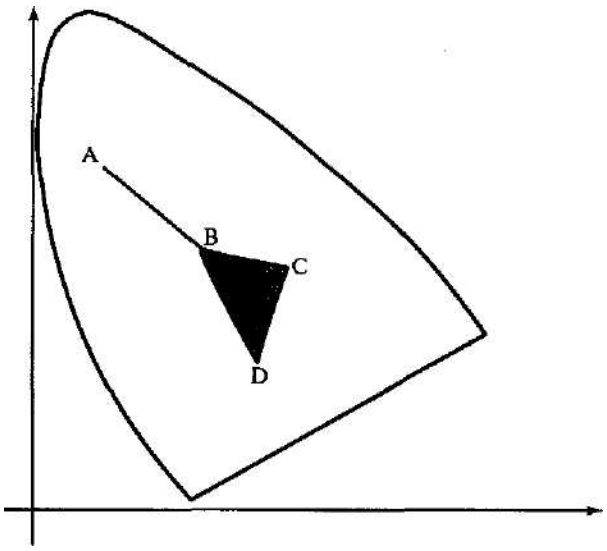


Рис. 7.5 Линейная модель системы цветов дает ряд полезных конструкций. Если есть два цвета с координатами МКО A и B, то все цвета, которые можно получить из неотрицательных комбинаций этих цветов, можно представить на отрезке прямой, соединяющем *A и B.* Подобным образом, при заданных *B, C* и *D* цвета, которые получаются из их комбинаций, лежат в треугольнике, образованном тремя этими точками.

Линейные модели (рис. 7.5) важны при разработке мониторов — у каждого монитора есть только три цвета, и чем насыщеннее цвет каждого из них, тем больше цветов можно изобразить. Именно поэтому одни и те же цвета выглядят по разному на различных мониторах. Кривизна спектральной траектории — вот причина того, почему ни один набор из трех реальных основных цветов не может передать все цвета без разностного сопоставления

7.2 Нелинейные цветовые пространства

Координаты цвета в линейном пространстве не обязательно кодируют свойства, принятые в языке, или важные для применения. В число полезных понятий, касающихся света, входят: *оттенок* — свойство цвета, которое меняется от красного до зеленого, *насыщенность* — свойство цвета, которое изменяется от красного до розового и *яркость* (иногда его называют *светлостью,* или *значением)* — свойство, которое изменяется от черного до белого. Например, если нужно проверить, попадает ли цвет в особенную область красного цвета, следует закодировать непосредственно оттенок цвета.

Еще одна неприятность, связанная с линейными пространствами, — это то, что координаты цвета не соответствуют человеческому пониманию топологии цвета. Принято считать, что оттенки образуют круг, в том смысле, что оттенок изменяется от красного через оранжевый до желтого, а затем до зеленого и голубого, синего, фиолетового и опять красного. Другой подход — это понятие о локальной связи оттенков: красный соседствует с фиолетовым и оранжевым, оранжевый — с красным и желтым, желтый — с оранжевым и зеленым, зеленый — с желтым и голубым, голубой — с зеленым и синим, синий — с голубым и фиолетовым, а фиолетовый — с синим и красным. Каждая из этих локальных связей работает, и глобально их можно смоделировать, расположив оттенки по кругу. Это значит, что оттенок нельзя смоделировать с помощью отдельных координат линейного цветового пространства, так как у координат есть максимальное значение, которое сильно отличается от минимального.

Пространство HSV

Обычный способ решить эту проблему — создать цветовое пространство, которое будет отражать эти связи с помощью нелинейных преобразований в пространстве RGB. Существует множество таких пространств. Одно, называемое *пространством HSV* (Hue-Saturation-Value — оттенок-насыщенность-значение), можно получить, направив луч зрения от начала координат вдоль (1,1,1)-оси RGB-куба. Поскольку RGB — линейное пространство, то яркость, которую в пространстве HSV называют *значением,* изменяется при удалении от начала координат. Можно спроектировать RGB-куб и получить двумерное пространство с постоянным значением, а для повышения точности преобразовать его в шестиугольник. Это

даст структуру, показанную на рис. 7.6, где оттенок определяется углом, который изменяется при повороте вокруг нейтральной точки, а при удалении от нее изменяется насыщенность.

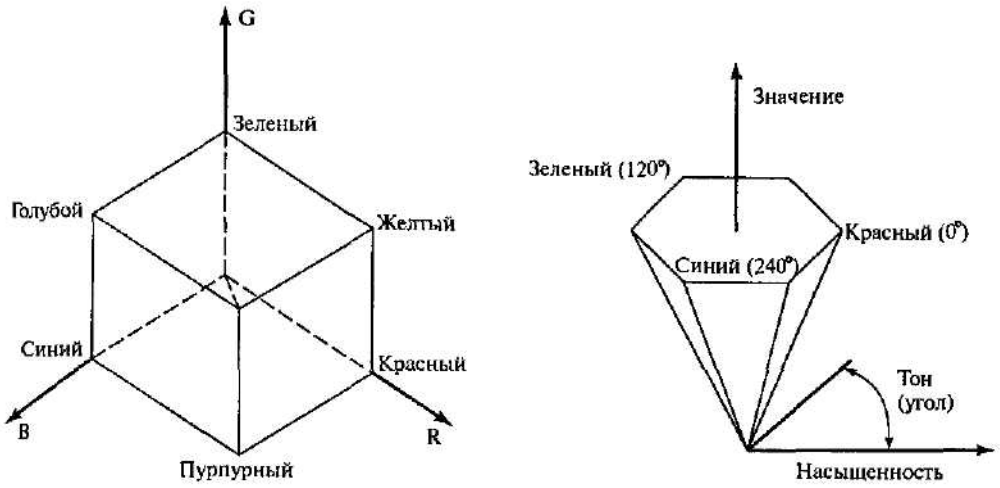


Рис. 7.6 Слева мы видим RGB-куб — пространство всех цветов, которые получаются из комбинаций трех основных цветов. Представленный куб принято рассматривать вдоль его нейтральной оси — от начала координат до точки (1, 1, 1) — в этом случае виден шестиугольник, изображенный справа. Шестиугольник кодирует оттенок (свойство, которое изменяется при изменении цвета от зеленого до красного) как угол, который подбирается интуитивно. Справа мы видим конус, образованный этим сечением, где длина оси конуса равна значению (или яркости) цвета, угол наклона дает оттенок, а расстояние до начала координат — насыщенность

Основание конуса представляет яркие цвета и соответствует $V = 1$. Однако цвета основания $V = 1$ не имеют одинаковой воспринимаемой интенсивности (люминантности). Тон (*H)* измеряется углом, отсчитываемым вокруг вертикальной оси *Значение.* При этом красному цвету соответствует угол 0°, зеленому – угол 120° и т.д. Цвета, взаимно дополняющие друг друга до белого, находятся напротив один другого, т.е. их тона отличаются на 180°. Величина *S* изменяется от 0 на оси *Значение до* 1 на гранях конуса.

Конус имеет единичную высоту ($V = 1$) и основание, расположенное в начале координат. В основании конуса величины *H* и *S* смысла не имеют. Белому цвету соответствует пара *S = 1, V= 1*. Ось *Значение (*$S = 0$) – серым тонам. При $S = 0$ значение *H* не имеет смысла (по соглашению принимает значение HUE_UNDEFINED).

Процесс добавления белого цвета к заданному можно представить как уменьшение насыщенности *S*, а процесс добавления черного цвета – как уменьшение яркости *V*.

Существует множество других возможных преобразований координат при переходе от одного линейного цветового пространства к другому или от линейного к нелинейному. Очевидного преимущества одних координат над другими нет (особенно если эти системы отличаются только одним преобразованием), если не рассматривать вопросы кодирования, скорости передачи информации и т.п. или однородности восприятия.

7.3 Равномерные цветовые пространства

Как правило, точно воспроизвести цвет нельзя. Это значит, что требуется знать, заметит ли человек разницу в цвете. В общем случае полезно сравнивать между собой небольшие различия в цвете, но опасно пытаться сравнивать большие различия, например искать ответ на вопрос: "Какая разница больше: между синим и желтым участком или между красным и зеленым?".

Можно определить *едва заметную разницу,* модифицируя цвет, который показывается наблюдателю, до тех пор, пока наблюдатель не заметит, что по сравнению с первоначальным вариантом цвет изменился. Если эти различия отобразить в цветовом пространстве, они сформируют границу области цветов, неотличимых от исходного цвета. Обычно едва

различимые отличия образуют эллипсы. Оказывается, что в пространстве МКО *(x, y)* эти эллипсы довольно сильно зависят от того, в какой части пространства они расположены.

Из сказанного следует, что величина отличий по координатам *(x,y),* которая задается как $((\Delta x)^2 + (\Delta y)^2)^{1/2}$, — это плохой показатель значения отличий в цвете (если бы это был хороший показатель, то эллипсы, представляющие неотличимые цвета, были бы кругами). В этой связи полезно ввести *однородное цветовое пространство* — пространство, в котором расстояние однозначно указывает на величину отличия двух цветов; если в таком пространстве расстояние не будет превышать некоторого порогового значения, человек не сможет отличить один цвет от другого.

На сегодняшний день почти универсальным и самым распространенным однородным цветовым пространством является *пространство МКО* ***LAB***. Координаты цвета в пространстве LAB можно найти как нелинейную проекцию координат XYZ:

$$L^* = 116\left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - 16, \frac{Y}{Y_n} > 0.00856$$

$$a^* = 500\left[\left(\frac{X}{X_n}\right)^{\frac{1}{3}} - \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}}\right]$$

$$b^* = 200\left[\left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - \left(\frac{Z}{Z_n}\right)^{\frac{1}{3}}\right]$$

Здесь X_n, Y_n и Z_n — это координаты X, Y и Z эталонного белого участка. Пространство LAB популярно потому, что оно однородно по своей природе. В некоторых задачах необходимо знать, насколько разными два цвета будут казаться *человеку,* а отличия координат пространства LAB позволяют получить однозначный ответ на этот вопрос.

7.4 Оптимизация палитр

Каждый графический файл состоит из двух основных частей - оглавления и непосредственно данных. В первой части содержится информация о структуре графического файла. В большинстве случаев после оглавления в файле палитрового изображения помещена палитра цветов. Для реальных изображений эта палитра имеет большие размеры, что существенно влияет на объем графического файла.

При передаче информации по каналам связи, а также при решении некоторых практических задач важным является минимизация размера файла. Этого можно достичь за счет оптимизации палитры изображения. Искажение визуального качества при этом должно быть минимально.

К решению этой актуальной задачи можно подходить по-разному. Известные подходы, которые базируются на так называемой "безопасной" палитре или выборе наиболее встречающихся цветов, не дают хорошего результата. Это объясняется тем, что при преобразованиях с фиксированной палитрой, не проводится анализ цветности обрабатываемого изображения. Поэтому для изображений, которые содержат лишь оттенки одного цвета, метод с использованием "безопасной" палитры неэффективен. Метод, основанный на использовании наиболее встречающихся цветов, хорошо обрабатывает изображения, которые содержат много оттенков, и неэффективен для обработки изображений с большим количеством основных цветов.

7.4.1 Поиск ближайшего цвета в палитре

В тех случаях, когда **целевая палитра уже известна**, нужно воспользоваться алгоритмом поиска ближайшего цвета в палитре. Для решения рассмотрим более простую задачу - нужно найти в некоторой палитре из n цветов, цвет максимально похожий на некоторый заданный цвет.

Введем обозначения:

(R_0,G_0,B_0) - цвет, аналог которого нужно найти в палитре.

(R_i,G_i,B_i) - i-тый цвет в палитре.

Различие цветов будем оценивать с помощью следующей функции:

$$f_i = 30*(R_i-R_0)^2+59*(G_i-G_0)^2+11*(B_i-B_0)^2.$$

Множители 30, 59 и 11 - отражают различную чувствительность человеческого глаза к красному, зеленому и синему цветам соответственно. Далее мы по очереди перебираем все цвета палитры и ищем цвет, для которого f_i принимает минимальное значение. Это и будет искомый цвет.

Таким образом, если нам известна целевая палитра задача оптимизации сводится к поиску для каждого точки изображения ближайшего цвета в палитре. Качество получаемого изображения в этом случае зависит от способа получения палитры. Рассмотрим некоторые способы получения целевой палитры из N цветов.

7.4.2 Частотный метод получения палитры

Самый простой подход заключается в том, чтобы, перебрав все пикселы в картинке, посчитать сколько раз встречается каждый цвет и составить палитру из тех цветов, которые встречаются чаще других. Если некоторый оттенок синего цвета встречается 100 раз, а оттенок красного только 20, то, очевидно, предпочтение отдается синему цвету. Но этот метод имеет несколько недостатков. Один из них заключается в том, что некоторые цвета будут исключены напрочь. Представьте себе картинку с загородной дорогой, где преобладают синие, коричневые, желтые, зеленые тона, и где-то в одном углу оказался маленький красный, дорожный знак. Если красный цвет более нигде на этой картинке не встречается, то он не попадет в палитру, и, следовательно, знак будет окрашен в какой-то другой цвет.

Заметим, что данный метод не накладывает никаких ограничений на количество N цветов в палитре.

7.4.3 Равномерный метод получения палитры

Другим способом выбрать комплект цветов является палитра с равномерно распределенными красной, зеленой и синей компонентами. Нужно разделить R,G и B компоненты на K отрезков и усреднить цвет в этих отрезках. Такой подход обеспечивает широкий выбор цветов, но при этом не учитывается тот факт, что в большинстве картинок нет равномерного цветового распределения. А дорожный знак хотелось бы отчетливо показать, не утратив ничего в тонких оттенках неба, деревьев и дороги.

Для такого способа выбора палитры N будет равно:

$$N=K^3,$$

где K – количество разбиений на канал.

7.4.4 Квантование цветов медианным сечением

Другое решение проблемы - это метод квантования цветов медианным сечением. Цветовое пространство рассматривается как трехмерный куб. Каждая ось куба соответствует

одному из трех основных цветов: красному, зеленому или синему. Каждая из трех сторон разбивается на 255 равных частей, деления на осях нумеруются от 0 до 255, причем большее значение соответствует большей интенсивности цвета. Вы можете отмечать точки внутри куба, соответствующие цветам так же, как точки трехмерного графика, если бы вам заданы x,y и z координаты точки. К примеру, если значения красной, зеленой и синей компонент есть 128, 64 и 192 соответственно, то вы можете отложить на оси К - 128, на оси З - 64, на оси С - 192 и получить точку внутри куба, соответствующую данному цвету. Черный цвет с компонентами 0, 0, 0, попадет на одну вершину куба, а белый - с компонентами 255, 255, 255, в другую, диагонально противоположную вершину. Если бы вы отметили точки внутри куба, соответствующие цветам пикселов в обычном полноцветном изображении, вы бы обнаружили, что точки отнюдь не равномерно разбросаны по всему кубу; очевидна тенденция группирования точек в отдельных регионах.

Метод медианного сечения делит куб на 256 параллелепипедов, каждый из которых содержит примерно одинаковое количество пикселов. При таком разбиении куба центральная точка каждого параллелепипеда представляет оптимальный выбор для цветовой палитры. И в самом деле, в той области куба, которая густо заполнена точками, будет больше параллелепипедов и, соответственно, в палитру попадет больше цветов. А там, где точек меньше, оттуда и цветов будет взято меньше. Ни один цвет не будет отброшен полностью. Тем же цветам, которые встречаются чаще, будет отдано предпочтение. Обратимся еще раз к примеру с загородной дорогой. Цвета в палитре, полученной медианным сечением, будут концентрироваться вокруг синего, коричневого, желтого и зеленого, но по крайней мере найдется один оттенок с достаточной красной компонентой, чтобы аппроксимировать цвет знака "стоп". Рассмотрим метод медианного сечения при квантовании цветов более подробно.

Алгоритм квантования цветов медианным сечением

1. Метод квантования цветов с помощью медианного сечения применяется при выборе 256 цветов, чтобы представить полноцветное изображение, содержащее несколько тысяч цветов. Чтобы понять как работает метод медианного сечения, мы представим цветовое пространство как куб. Каждая ось соответствует одному из трех основных цветов и деления на оси нумеруются от 0 до 255; большему номеру соответствует большая интенсивность цвета. Цвета в изображении отмечаются внутри куба так же, как точки на трехмерном графике.

2. Первый шаг состоит в отсечении "краёв" куба, которые не содержат пикселов. К примеру, если у всех пикселов значения красной компоненты не меньше, чем 8 и не больше, чем 250, то отбрасываются части куба от К=0 до К=7 и от К=251 до К=255.

3. Второй шаг заключается в разрезании полученного параллелепипеда на два в срединной точке (медиане) самой длинной стороны. Если самая длинная сторона параллельна оси С, то компьютер выбирает срединное синее значение из всех синих значений представленных в параллелепипеде (к примеру, 50.000-ое вхождение в отсортированном списке из 100.000 синих значений) и разрезает в этой точке. Теперь параллелепипед разделен на два параллелепипеда меньшего размера, содержащих одинаковое количество пикселов.

4. Весь предыдущий процесс - отсечение пустых "краев" и разрезание самой длинной стороны в срединной точке - повторяется для двух меньших параллелепипедов. Теперь исходный куб разделен на четыре параллелепипеда, содержащих приблизительно одинаковое количество пикселов.

5. Медианное сечение повторно применяется для того, чтобы разделить куб на 8, 16, 32, 64, 128 и 256 параллелепипедов. Они содержат примерно одно и то же количество пикселов и их объемы обратно пропорциональны плотностям пикселов.

6. Имея пространство, разделенное таким образом, легко выбрать палитру. Каждый из 256 параллелепипедов содержит пикселы приблизительно одинакового цвета и центр каждого параллелепипеда представляет оптимальное значение цвета для палитры. Имея координаты вершин, очень просто вычислить координаты центральной точки. (Некоторые графические программы вместо того, чтобы вычислять центральную точку, усредняют значения всех пикселов находящихся внутри параллелепипеда; на вычисления уйдет больше времени, но полученная палитра будет лучше.) Вычислив R, G и B координаты для всех 256 центральных точек в параллелепипедах, получим 256 цветов, которые и будут составлять палитру

Количество цветов целевой палитры будет равно:
$N=2^K$, где K – количество разбиений пространства.

7.4.5 Квантование цветов методом k-средних

Целью задачи кластеризации является разбиение множества объектов на классы (кластеры) на основе некоторой меры сходства объектов. При кластеризации цветов мерой сходства является близость яркостей пикселей (см. формулу в п. 7.4.1). Мы будем пытаться разбить палитру на $N=k$ кластеров и заменить цвета попавшие в какой-либо кластер центром этого кластера. Стандартным методом решения такой задачи является *метод k-средних*.

Алгоритм метода k-средних

Пусть имеем набор векторов x_i $(i=1 \div p)$ и k – число кластеров, на которые нужно разбить набор x_i. Суть алгоритма заключается в том, чтобы найти k средних m_j $(j=1 \div k)$ (центров кластеров) и отнести каждый из векторов x_i к одному из k кластеров.

1. Случайным образом выбрать k центров m_j $(j=1 \div k)$;
2. Для каждого x_i $(i=1 \div p$ подсчитать расстояние до каждого из m_j $(j=1 \div k)$;
   Отнести (приписать) x_i к кластеру j расстояние до центра которого m_j минимально.
   (Расстояние между цветами предлагается вычислять по формуле различия цветов, приведенной выше).
3. Пересчитать центры кластеров m_j $(j=1 \div k)$ по всем кластерам;
4. Повторять шаги 2, 3 пока кластеры не перестанут изменяться;

Пример

Пусть имеем 10 точек: 7 5 6 9 11 15 56 45 27 20. Необходимо разбить из на 3 кластера.
Выберем 3 центра кластера случайным образом: 6 9 15.
Вычисляем расстояния для каждой точки до кластеров и записываем эту точку в ближайший кластер.

Центр кластера	6	9	15
Точки кластера	5 6 7	9 11	15 20 27 45 56
Новые центры кластеров	6	10	33

Вычисляем расстояния для каждой точки до кластеров и записываем эту точку в ближайший кластер.

Центр кластера	6	10	33
Точки кластера	5 6 7	9 11 15 20	27 45 56
Новые центры кластеров	6	14	43

Вычисляем расстояния для каждой точки до кластеров и записываем эту точку в ближайший кластер.

Центр кластера	6	14	43
Точки кластера	5 6 7 9	11 15 20 27	45 56
Новые центры кластеров	7	18	51

Вычисляем расстояния для каждой точки до кластеров и записываем эту точку в ближайший кластер.

Центр кластера	7	18	51
Точки кластера	5 6 7 9 11	15 20 27	45 56
Новые центры кластеров	8	21	51

Вычисляем расстояния для каждой точки до кластеров и записываем эту точку в ближайший кластер.

Центр кластера	8	21	51
Точки кластера	5 6 7 9 11	15 20 27	45 56
Новые центры кластеров	8	21	51

Кластеры не изменились, значит можно прекратить итерации.

8 Фильтрация изображений

8.1 Общие принципы

Кроме простейших алгоритмов обработки фотоизображения, рассмотренных выше, существует великое множество полезных манипуляций с цифровыми фотоизображениями. А если изображение снято не в фокусе? В расплывчатых изображениях можно увеличить резкость, и, наоборот, четкие, контрастные изображения можно размыть, имитируя эффект смягчающих фотофильтров.

Фильтрация изображений является одной из самых фундаментальных операций технического зрения, распознавания образов и обработки изображений. Фактически, с той или иной фильтрации исходных изображений начинается работа подавляющего большинства методов. Рассматриваемые в данном модуле фильтры имеют, таким образом, чрезвычайную важность с точки зрения их применения в различных приложениях.

Основная стратегия, используемая при фильтрации, состоит в том, чтобы применять взвешенные суммы соседних пикселей, используя различные весовые коэффициенты. Несмотря на свою простоту такой процесс очень полезен. Он позволяет сглаживать шумы на изображении, определять края и получать интересные визуальные эффекты.

Для начала введём специальный термин: *маска фильтра (скользящее окно, апертура)* представляет собой матрицу размера $n \times m$ в общем случае. Она накладывается на изображение и осуществляется умножением элементов *маски фильтра* и соответствующих элементов изображения с последующей обработкой результата. В большинстве случаев апертура фильтра квадратная и имеет нечетные размеры. Обычно работают не с полными размерами апертуры, а с половинными размерами $r=[n/2],\ q=[m/2]$, где квадратные скобки это операция взятия целой части.

Заметим также, что если сумма коэффициентов ядра больше 1, то яркость изображения будет увеличиваться; если меньше 1 - яркость уменьшится.

Иногда операцию наложения фильтра называют операцией свертки, а маску фильтра – *ядром* свертки. Записывают операцию свертки функции *f* по ядру *h* следующим образом

$$g_{i,j} \equiv f_{i,j} \otimes h_{i,j} \quad \text{где } i,j \in Z$$

Или с точки зрения фильтрации изображения

$$\widetilde{f}_{i,j} == \sum_{k=-r}^{r} \sum_{l=-q}^{q} h_{k,l} f_{i-k,j-l}, \qquad r=[n/2],\ q=[m/2]$$

Когда маска передвигается к границе изображения, возникает так называемое *явление краевого эффекта.* Во избежание этого нежелательного эффекта необходимо, когда маска вышла за пределы исходного изображения, дополнить его ненулевыми элементами, например, элементами изображения, симметричными относительно его краев.

Апертура фильтра передвигается по изображению слева направо и сверху вниз на один пиксель (то есть на следующем шаге фильтр работает с окном, состоящим не только из элементов исходного изображения, но и из элементов, ранее подвергнувшихся преобразованию, – своего рода «принцип снежного кома»). Кроме того, заметим, что если речь идёт об апертуре размера $n \times 1$, то такое преобразование называется одномерным; соответственно, когда маска – матрица - это двумерное преобразование. Обычно используется апертуры с нечетными размерами, чтобы совместить центральный элемент матрицы с тем пикселем изображения, для которого и вычисляется свертка в данный момент (иногда говорят, что пиксель находится под центром ядра)

Речь пойдёт, в основном, о фильтрах шумоподавления, но для того, чтобы можно было быстрее и нагляднее оценивать их работу, требуется реализовать ещё и возможность

искусственного зашумления изображения. Одним из способов является получение изображения с равномерным шумом

$$I_{i,j} = g_{i,j} + Err(i,j),$$

где *Err(i,j)* – нормально распределенная случайная величина.

8.2 Фильтры шумоподавления

Одним из их возможных применений сглаживающих фильтров является шумоподавление, т.е. задача восстановления исходного изображения, к пикселям которого добавлен случайный шум. Шум меняется независимо от пикселя к пикселю и, при условии, что математическое ожидание значения шума равно нулю, шумы соседних пикселей будут компенсировать друг друга. Чем больше окно фильтрации, тем меньше будет усредненная интенсивность шума, однако при этом будет происходить и существенное размытие значащих деталей изображения.

При размывании перераспределяются цвета в изображении и смягчаются резкие границы, в то время как при увеличении резкости подчеркиваются различия между цветами смежных пикселей и выделяются незаметные детали. Такие фильтры называют сглаживающими, они позволяют подавить высокочастотные (мелкие) шумы, снижая локальную контрастность изображения.

Ядро **равномерного фильтра**, состоит из совокупности коэффициентов, каждый из которых меньше 1, а их сумма составляет 1. В простейшем случае каждый элемент ядра размером $n \times m$ равен $1/mn$. Это означает, что каждый пиксель поглотит что-то из цветов соседей, но полная яркость изображения останется неизменной.

Итоговое изображение будет более размытым по сравнению с оригиналом потому, что цвет каждого пикселя распространился среди соседей. Степень размывания можно увеличить либо используя ядро большего размера, чтобы распределить цвета среди большего числа соседей, либо, подбирая коэффициенты ядра и уменьшая влияние центрального коэффициента, либо фильтруя изображение еще раз с ядром размывания. Характерной чертой этого фильтра, отличающей его, к примеру, от эффекта расфокусировки линз в реальной жизни, является то, что образом белой точки на черном фоне будет равномерно серый квадрат.

Естественным предположением об исходном незашумленном изображении будет схожесть значений интенсивности пикселей, находящихся рядом. Причем чем меньше расстояние между пикселями, тем больше вероятность их похожести. Заметим, что в зашумленных изображениях схожесть пикселей никак не зависит от расстояния между ними. Исходя из вышесказанного можно предположить, что шумоподавление при помощи прямоугольного фильтра имеет существенный недостаток: пиксели, находящиеся на значительном расстоянии от обрабатываемого, оказывают на результат тот же эффект, что и ближние.

Более эффективное шумоподавление можно, таким образом, осуществить, если влияние пикселей на обрабатываемый будет уменьшаться с расстоянием. Этим свойством обладает **гауссовский фильтр**. Размытие с помощью Гауссова фильтра (Gaussian blur) это фактически свертка по функции:

$$\widetilde{I}(i,j) = \sum_{l=-r}^{r} \sum_{k=-q}^{q} I(i-l)(j-k) \cdot A \cdot e^{-\frac{(l^2+k^2)}{2\sigma^2}},$$

где *i,j* – обрабатываемая точка изображения; A –нормирующая величина, она выбирается так, чтобы сумма коэффициентов ядра фильтра была равно 1; σ - среднеквадратичное отклонение, задает степень размытия. При высоких σ влияние удаленных пикселей увеличивается и степень размытия повышается.

Значение A обычно выбирается следующим образом

$$h_{l,k} = e^{-\frac{(l^2+k^2)}{2\sigma^2}}$$, где $l=[-r \div r]$, $k=[-q \div q]$

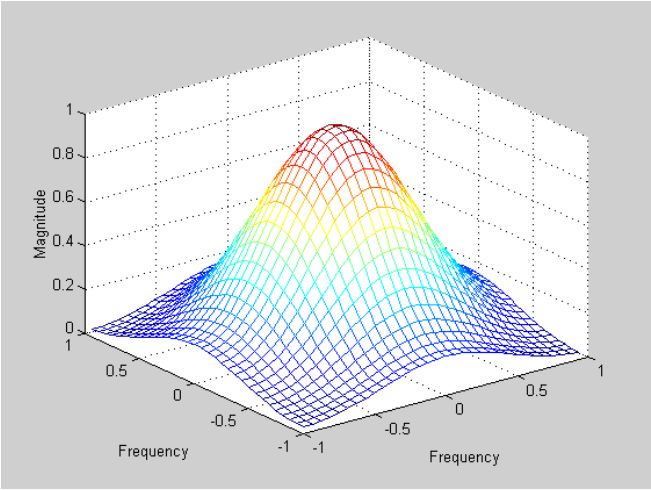
$$A = \frac{1}{\sum_{l=-r}^{r}\sum_{k=-q}^{q} h_{l,k}}$$

Пример маски для фильтра Гаусса размера 5×5 и с σ=0,5:

$$\begin{pmatrix} 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0002 & 0.0837 & 0.6187 & 0.0837 & 0.0002 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \end{pmatrix}$$

Пример маски для фильтра Гаусса размера 5×5 и с σ= 0,7:

$$\begin{pmatrix} 0.0001 & 0.0020 & 0.0055 & 0.0020 & 0.0001 \\ 0.0020 & 0.0422 & 0.1171 & 0.0422 & 0.0020 \\ 0.0055 & 0.1171 & 0.3248 & 0.1171 & 0.0055 \\ 0.0020 & 0.0422 & 0.1171 & 0.0422 & 0.0020 \\ 0.0001 & 0.0020 & 0.0055 & 0.0020 & 0.0001 \end{pmatrix}$$



Все линейные алгоритмы фильтрации приводят к сглаживанию резких перепадов яркости изображений, прошедших обработку. Этот недостаток, особенно существенный, если потребителем информации является человек, принципиально не может быть исключен в рамках линейной обработки.

Вторая особенность линейной фильтрации - ее оптимальность при гауссовском характере помех. Обычно этому условию отвечают шумовые помехи на изображениях, поэтому при их подавлении линейные алгоритмы имеют высокие показатели. Однако, часто приходится иметь дело с изображениями, искаженными помехами других типов. Одной из них является импульсная помеха. При ее воздействии на изображении наблюдаются белые или (и) черные точки, хаотически разбросанные по кадру. Применение линейной фильтрации в этом случае неэффективно - каждый из входных импульсов (по сути - дельта-функция)

дает отклик в виде импульсной характеристики фильтра, а их совокупность способствует распространению помехи на всю площадь кадра.

Удачным решением перечисленных проблем является применение **медианной фильтрации**, предложенной Дж. Тьюки в 1971 г.

При медианной фильтрации используется окно, обычно имеющее центральную симметрию. Наиболее часто применяемые варианты окон - крест или квадрата. Размеры апертуры принадлежат к числу параметров, оптимизируемых в процессе анализа эффективности алгоритма. Отсчеты изображения, оказавшиеся в пределах окна, образуют *рабочую выборку* текущего шага.

Двумерный характер окна позволяет выполнять, по существу, двумерную фильтрацию, поскольку для образования оценки привлекаются данные как из текущих строки и столбца, так и из соседних. Если упорядочить последовательность $\{y_i,\ i = \overline{1,n}\}$ по возрастанию, то ее *медианой* будет тот элемент выборки, который занимает центральное положение в этой упорядоченной последовательности. Полученное таким образом число и является продуктом фильтрации для текущей точки кадра.

Если импульсная помеха не является точечной, а покрывает некоторую локальную область, то она также может быть подавлена, если размер этой локальной области будет меньше, чем половина размера апертуры медианного фильтра. Поэтому для подавления импульсных помех, поражающих локальные участки изображения, следует увеличивать размеры апертуры МФ.

Импульсная помеха. При ее наложении использовался датчик случайных чисел с равномерным на интервале [0, 1] законом распределения, вырабатывающий во всех точках кадра независимые случайные числа. Интенсивность помехи задавалась вероятностью p ее возникновения в каждой точке. Если для случайного числа n_{i_1,i_2}, сформированного в точке (i_1,i_2), выполнялось условие $n_{i_1,i_2} < p$, то яркость изображения x_{i_1,i_2} в этой точке *замещалась* числом 255, соответствующим максимальной яркости (уровню белого).

Вместе с тем, как говорилось выше, медианная фильтрация в меньшей степени сглаживает границы изображения, чем любая линейная фильтрация. Механизм этого явления очень прост и заключается в следующем. Предположим, что апертура фильтра находится вблизи границы, разделяющей светлый и темный участки изображения, при этом ее центр располагается в области темного участка. Тогда, вероятнее всего, рабочая выборка будет содержать большее количество элементов с малыми значениями яркости, и, следовательно, медиана будет находиться среди тех элементов рабочей выборки, которые соответствуют этой области изображения. Ситуация меняется на противоположную, если центр апертуры смещен в область более высокой яркости. Но это и означает наличие чувствительности у МФ к перепадам яркости.

8.3 Фильтры увеличения резкости

В ядре резкости центральный коэффициент больше 1, а окружен он отрицательными числами, сумма которых на единицу меньше центрального коэффициента. Таким образом, увеличивается любой существующий контраст между цветом пикселя и цветами его соседей. Увеличение резкости достигается точно так же, как и размывание, за исключением того, что используются другое ядро с иной целью увеличить, а не уменьшить четкость изображения.

Общий вид ядра 3×3 для повышения резкости выглядит следующим образом:

$$h = \begin{pmatrix} -k/8 & -k/8 & -k/8 \\ -k/8 & k+1 & -k/8 \\ -k/8 & -k/8 & -k/8 \end{pmatrix},$$

где параметр *k* определяет степень увеличения резкости. Обычно используется *k=2*.

Эффект повышения резкости достигается за счет того, что фильтр подчеркивает разницу между интенсивностями соседних пикселей, удаляя эти интенсивности друг от друга. Этот эффект будет тем сильней, чем больше значение центрального члена ядра.

Характерным артефактом линейной контрастоповышающей фильтрации являются заметные светлые и менее заметные темные ореолы вокруг границ.

8.4 Оконтуривание (нахождение границ)

Все методы выделения границ работают с *яркостью* точки, то есть со значением, полученным из значений трёх цветовых каналов по известной формуле (см. п. 7.X). Однако для этого вовсе необязательно предварительно преобразовывать всё изображение к оттенкам серого, достаточно лишь получать значение яркости в тот момент и для той точки, с которой идёт работа, а полученное в результате преобразований значение повторять по трём каналам.

8.4.1 Поиск границ на основе градиента

Существует несколько подходов к проблеме выделения границ [Форсайт Понс]. Первое семейство методов основано на приближенном вычисление градиента, анализе его направления и абсолютной величины.

В точках большого перепада яркости градиент имеет большее значение. Это неудивительно, поскольку (частные) производные по определению соответствуют скоростям изменения функции (яркости) по вертикали и горизонтали. Взяв пиксели с соответствующей длиной градиента, большей порога α, мы получим некий алгоритм нахождения границ.

Недостаток алгоритма: Если же мы имеем дело с зашумленным изображением, то карту граничных точек будут загрязнять не только реально существующие, но и несущественные детали и просто шум. Так произойдет, поскольку в нашем алгоритме мы не учли, что граничные точки соответствуют не просто перепадам яркости, а перепадам яркости между относительно монотонными областями.

Как отделить подобные перепады яркости от перепадов яркости, вызванных шумами и несущественными деталями? Для этого изображение подвергают сглаживающей гауссовской фильтрации. Такое решение проблемы, на первый взгляд, парадоксально - для нахождения границ мы их с начала размываем. Данный прием основывается на том, что при сглаживающей фильтрации мелкие несущественные детали будут размываться существенно быстрее перепадов между областями.

Другой недостаток алгоритма: четко выраженные границы проявятся как жирные линии в несколько пикселей толщиной. Чтобы преодолеть данный недостаток необходимо подбтрать значения α вручную, и, очевидно, такое значение будет оптимально не для всех частей изображения

Существует множество способов определить диффенециальный оператор для дискретных изображений при помощи линейного фильтра [Иванов и др.]. В частности, распространенными вариантами являются фильтры Превита (Prewitt) и Собеля (Sobel).

Маска фильтра Собеля для выделения горизонтальных границ:

$$h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Маска фильтра Превита для выделения горизонтальных границ:

$$h = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}.$$

Фильтры, приближающие оператор производной по *x*-координате, получаются путем транспонирования матриц.

В силу того, что данные фильтры не меняют среднюю интенсивность изображения (сумма элементов ядра равна единице), в результате их применения получается, как правило, изображение со средним значением пикселя близким к нулю (сумма элементов ядра равна нулю). Вертикальным перепадам (границам) исходного изображения соответствуют пиксели с большими по модулю значениями на результирующем изображении.

Неплохие результаты дает фильтрация с ядром Робертса:

$$h = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$h = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

8.4.2 Поиск границ на основе лапласиана

Как известно из математического анализа, необходимым и достаточным условием экстремального значения первой производной функции в некой точке является равенство нулю второй производной в этой точке, причем по разные стороны от точки вторая производная должна иметь разные знаки. Про такую точку говорят, что вторая производная в ней пересекает ноль.

В двумерном случае, аналогом второй производной является скалярный оператор, называемый лапласианом:

$$\Delta f = \left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \right)$$

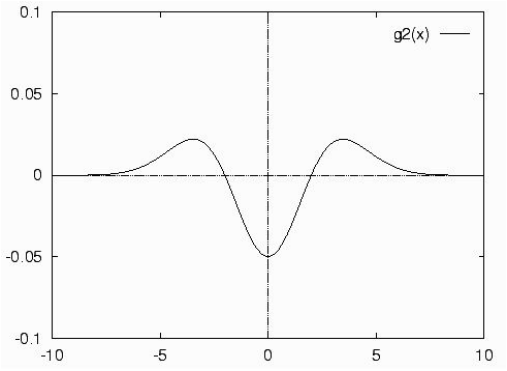
Нахождение границ на изображении может, таким образом, производиться по аналогии с одномерным случаем: граничными признаются точки, в которых лапласиан равен нулю и вокруг которых он имеет разные знаки. Кроме того, оценка лапласиана при помощи линейной фильтрации предваряется гауссовской сглаживающей фильтрацией, чтобы снизить чувствительность алгоритма к шуму (аналогично тому, как это описано выше).

Композиция линейных фильтров есть линейный фильтр. Поэтому гауссовское сглаживание и поиск лапласиана можно осуществить единовременно при помощи фильтра, который называется **лапласиан-гауссиана**. Поиск пересечений нуля, так же, как и линейная фильтрация, является сравнительно быстрой операцией, поэтому, описанный выше алгоритм применяется в системах, где принципиально и качество результата и быстродействие.

$$\Delta f(x,y) = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y}; \quad g(x,y) = Ae^{-\frac{x^2+y^2}{\sigma}};$$

$$\Delta g(x,y) = A'\left(2 - \frac{x^2+y^2}{\sigma}\right)e^{-\frac{x^2+y^2}{\sigma}}$$

График второй производной одномерной функции Гаусса с $\sigma = 8$



Простейший вариант дискретного приближения ядра лапласиана:

$$h = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Пример матрицы размера 11×11

$$h = \begin{bmatrix} 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & 0 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & 0 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -2 & -9 & -23 & -1 & 103 & 178 & 103 & -1 & -23 & -9 & -2 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & 0 \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & 0 \end{bmatrix}$$

В результате применения дискретного лапласиана большие по модулю значения соответствуют как вертикальным, так и горизонтальным перепадам яркости. h_{Δ} является, таким образом, фильтром, находящим границы любой ориентации. Нахождение границ на изображении может производиться путем применения этого фильтра и взятия всех пикселей, модуль значения которых превосходит некоторый порог. Однако такой алгоритм имеет существенные недостатки. Главный из них - неопределенность в выборе величины порога. Для разных частей изображения приемлемый результат обычно получается при существенно разных пороговых значениях. Кроме того, разностные фильтры очень чувствительны к шумам изображения.

Чтобы еще уменьшить чувствительность алгоритма к несущественным деталям, из числа граничных точек можно исключить те, длина градиента в которых меньше порога. Для одномерного случая это будет соответствовать разумному требованию того, чтобы в точке перепада величина первой производной была не слишком маленькой.

8.5 Примеры декоративных фильтров и спецэффектов

Акварелизация. Первый шаг в применении акварельного фильтра - сглаживание цветов в изображении. Одним из способов сглаживания является процесс медианнного осреднения цвета в каждой точке – так называемый *медианный фильтр*.

Медиана – средний элемент последовательности в результате её упорядочения по возрастанию/убыванию и присваиванию найденного значения только среднему элементу (речь снова о нечётной апертуре). Например, для той же апертуры 3 и двумерного фильтра

мы должны упорядочить 9 точек (например, по возрастанию), после чего значение 5-ой точки упорядоченной последовательности отправить в центр окна

После сглаживания цветов медианным фильтром (иногда это делают несколько раз), необходимо обработать изображение ядром резкости, чтобы подчеркнуть границы переходов цветов.

Результирующее изображение напоминает акварельную живопись. Это лишь один пример, который показывает, как можно объединять различные методы обработки изображений и добиваться необычных визуальных эффектов.

Тиснение преобразует изображение так, что фигуры внутри изображения смотрятся так, как будто они выдавлены на поверхности.

Предварительно изображение обрабатывается каким либо сглаживающим фильтром, после чего применяется ядро тиснения.

В отличие от ядер размывания и резкости, в которых сумма коэффициентов равна 1, сумма весов в ядре тиснения равна 0. Это означает, что "фоновым" пикселам (пикселам, которые не находятся на границах перехода от одного цвета к другому) присваиваются нулевые значения, а "не фоновым" пикселям - значения, отличные от нуля.

Примеры ядер тиснения

$$h = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}, \quad h = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Есть и другие способы получить ядро для тиснения. Один из общих случаев выглядит так:

$$h = \begin{pmatrix} c & c & c \\ c & c & c \\ -2c & -2c & -2c \end{pmatrix},$$

где c - некоторый коэффициент, обычно от 0.5 до 1.

После того, как значение пиксела обработано ядром тиснения, выполняется нормировка - к полученному значению прибавляется 128. Таким образом, значением фоновых пикселов станет средний серый цвет. Суммы, превышающие 255, можно округлить до 255 или взять остаток по модулю 255, чтобы значение оказалось между 0 и 255.

Заметим, что направление подсветки изображения можно изменять, меняя позиции 1 и -1 в ядре.

Ангелина Юрьевна Андреева,

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебное пособие
Издано в авторской редакции

Макет: А.Ю. Андреева