Министерство науки и высшего образования
Российской федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный технический университет»

Кафедра графики, конструирования и информационных технологий в
промышленном дизайне

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «ГРАФИЧЕСКИЕ ТЕХНОЛОГИИ И ФОРМАТЫ ГРАФИЧЕСКИХ ДАННЫХ»

для обучающихся по направлению
09.03.02 Информационные системы и технологии
всех форм обучения

Воронеж 2021

Составители:
А.В. Кузовкин,
А.П. Суворов,
Ю.С. Золототрубова

Методические рекомендации по выполнению лабораторных работ по дисциплине «Графические технологии и форматы графических данных» для обучающихся по направлению 09.03.02 Информационные системы и технологии всех форм обучения / ФГБО ВО «Воронежский государственный технический университет»; сост.: А.В. Кузовкин, А.П. Суворов, Ю.С. Золототрубова. – Воронеж: Изд-во ВГТУ, 2021.

Приводится описание выполнения лабораторных работ по дисциплине «Графические технологии и форматы графических данных» для обучающихся по направлению 09.03.02 Информационные системы и технологии всех форм обучения для студентов обучающихся по направлению 09.03.02 Информационные системы и технологии всех форм обучения.

Методические указания подготовлены в электронном виде.

Издается по решению редакционно-издательского совета
Воронежского государственного технического университета

Введение

Если заглянуть в историю, то можно проследить, как с момента появления первых ЭВМ люди стремятся разнообразить способы общения человека и машины, приблизившись к уровню общения человека с человеком. Это общение было бы гораздо более ограниченным, если бы не использовало один из наиболее простых способов — язык изображений, образов. Сегодня графические изображения на экране монитора современного персонального компьютера стали для нас нормой, совершенно неотъемлемым атрибутом интерфейса. Спектр применения компьютерной графики, помимо средства интерфейса «человек-машина», чрезвычайно широк: от создания рекламных роликов, компьютерных мультфильмов и игр, кроя одежды, малых и монументальных форм дизайна, компьютерной живописи до визуализации результатов научных изысканий [10]. Можно с уверенностью сказать, что популярность Internet, и в частности WWW, во многом объясняется широким применением графики.

Рынок программного и аппаратного обеспечения компьютерной графики – один из самых динамичных. Об этом можно судить по объему литературы и числу сервисов Internet, посвященных так или иначе компьютерной графике.

Предметом данной работы является обширная область компьютерных наук, посвященная представлению данных в памяти ЭВМ в графической форме. Это самое общее определение, так как под данными можно понимать как непосредственно хранящееся в виде файла изображение в одном из графических форматов, так и протокол обмена командами между пользователем и ЭВМ (то, что мы называем графическим интерфейсом), и битовую последовательность, сформированную для вывода на экран или печатающее устройство. Методы и способы представления и манипуляции этим видом данных относятся к компетенции компьютерной графики.

В работе рассматриваются различные способы представления изображений в памяти ЭВМ, методы и алгоритмы растеризации и обработки растровых изображений, матричные преобразования на плоскости и в пространстве, методы и алгоритмы удаления скрытых линий и поверхностей. Кроме того, приводятся основы использования графической библиотеки OpenGL, а также описываются базовые аппаратные средства, используемые при работе с изображениями.

Программный код, приведенный в пособии, создан в MS Visual Studio 2010 на языке C#.

1. Способы представления изображений в ЭВМ

Компьютерная (машинная) графика – область деятельности, изучающая создание, способы хранения и обработки изображений с помощью ЭВМ. Под ***интерактивной*** компьютерной графикой понимают раздел компьютерной графики, изучающий вопросы динамического управления со стороны пользователя содержанием изображения, его формой, размерами и цветом на экране с помощью интерактивных устройств взаимодействия. Кроме интерактивной в компьютерной графике выделяют разделы, изучающие методы работы с изображением на плоскости, так называемую ***2D графику***, и ***трехмерную (3D) графику***.

Трехмерное изображение отличается от двухмерного, тем, что строится исходя из математического описания некоторой трехмерной сцены. Математическое описание сцены чаще всего является моделью физических объектов в трехмерном пространстве. Таким образом, для получения трехмерного изображения требуется построить математическую модель сцены и объектов на ней, а далее визуализировать путем получения проекции с учетом освещения материалов и пр. В результате визуализации мы получим изображение на плоскости экрана или на выходе принтера.

Вопросы 2D, 3D графики, общего геометрического моделирования, связанные с визуализацией геометрических моделей входят в компетенцию ***компьютерной геометрии***.

На специализацию в отдельных областях указывают названия некоторых разделов: инженерная графика, научная графика, Webграфика, компьютерная полиграфия и прочие.

Кроме этого, по способу представления изображения в памяти ЭВМ, компьютерную графику разделяют на ***векторную, растровую*** и ***фрактальную***. Рассмотрим подробнее эти способы представления изображений, выделим их основные параметры и определим их достоинства и недостатки.

1.1. Растровое представление изображений

Что такое растровое изображение?

Возьмѐм фотографию (например, см. рис. 1.1). Конечно, она тоже состоит из маленьких элементов, но будем считать, что отдельные элементы мы рассмотреть не можем. Она представляется для нас, как реальная картина природы.

Теперь наложим на изображение прямоугольную сетку. Таким образом, разобьем изображение на прямоугольные элементы. Каждый прямоугольник закрасим цветом, преобладающим в нѐм (на самом деле программы при

оцифровке генерируют некий «средний» цвет, т. е. если у нас была одна чѐрная точка и одна белая, то прямоугольник будет иметь серый цвет).

Как мы видим, изображение стало состоять из конечного числа прямоугольников определѐнного цвета. Эти прямоугольники называют *pixel* (от *PIX ELemen*t) – ***пиксел*** или ***пиксель***.

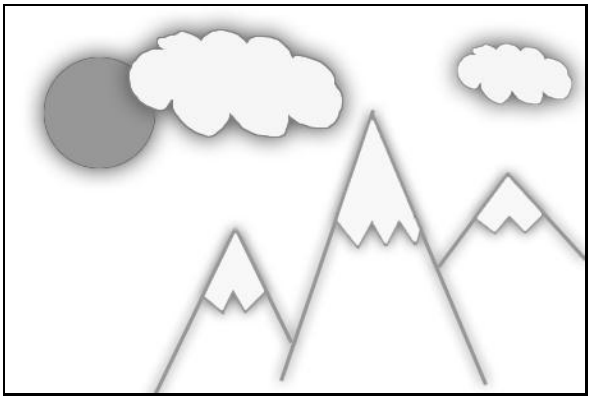


Рис. 1.1. Исходное изображение

Теперь каким-либо методом занумеруем цвета. Конкретная реализация этих методов нас пока не интересует. Для нас сейчас важно то, что каждый пиксель на рисунке стал иметь определѐнный цвет, обозначенный числом (рис. 1.2).

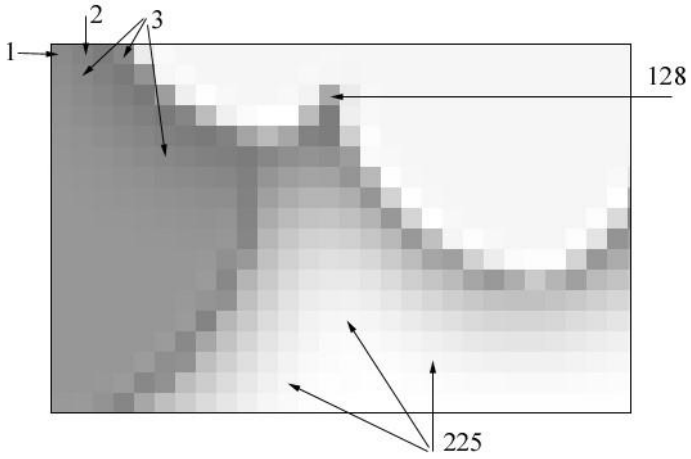


Рис. 1.2. Фрагмент оцифрованного изображения и номера цветов

Теперь пойдѐм по порядку (слева направо и сверху вниз) и будем в строчку выписывать номера цветов встречающихся пикселей. Получится строка примерно следующего вида:

1 2 8 3 212 45 67 45 127 4 78 225 34 ...

Вот эта строка и есть наши оцифрованные данные. Теперь мы можем сжать их (так как несжатые графические данные обычно имеют достаточно большой размер) и сохранить в файл.

Итак, под ***растровым*** (bitmap, raster) понимают способ представления изображения в виде совокупности отдельных точек (пикселей) различных цветов или оттенков. Это наиболее простой способ представления изображения, ибо таким образом видит наш глаз.

Достоинством такого способа является возможность получения фотореалистичного изображения высокого качества в различном цветовом диапазоне. Высокая точность и широкий цветовой диапазон требуют увеличения объема файла для хранения изображения и оперативной памяти для его обработки, что можно отнести к недостаткам. Вторым существенным недостатком является потеря качества изображения при его масштабировании.

1.1.1. Параметры растровых изображений

Как уже говорилось ранее, растровое изображение представляется в памяти ЭВМ в виде матрицы отдельных пикселей. В этой связи возникает вопрос о том, каково должно быть число этих пикселей и какое число бит отводится для хранения одного пикселя, т. е. каковы основные параметры растрового изображения – разрешение и глубина цвета.

Разрешение (resolution) — это степень детализации изображения, число пикселей (точек), отводимых на единицу площади. Поэтому имеет смысл говорить о разрешении изображения только применительно к какому-либо устройству ввода или вывода изображения. Например, пока имеется обычная фотография на твердом носителе, нельзя сказать о ее разрешении. Но как только мы попытаемся ввести эту фотографию в компьютер через сканер, нам необходимо будет определить разрешение оригинала, т. е. указать количество точек, считываемых сканером с одного квадратного дюйма.

Поскольку изображение можно рассматривать применительно к различным устройствам, то следует различать:

- разрешение оригинала;
- разрешение экранного изображения; ▪ разрешение печатного изображения.

Разрешение оригинала. Разрешение оригинала определяется при вводе изображения в компьютер и измеряется в точках на дюйм (*dots per inch - dpi*). При этом количество dpi определяет не число точек в квадратном дюйме, а количество точек на одной его стороне. Например, 300 dpi означает, что в квадратный дюйм изображения покрывается растровой сеткой 300x300 и после

сканирования, изображение соответствующее, квадратному дюйму будет состоять из 90 000 пикселей.

В дальнейшем разрешение оригинала влияет на разрешение изображений выводимых на разных устройствах (принтерах, экранах мониторов).

Установка разрешения оригинала зависит от требований, предъявляемых к качеству изображения и размеру файла. В общем случае действует правило: чем выше требования к качеству, тем выше должно быть разрешение оригинала.

Для получения на экране изображения близкого к размеру оригинала обычно использует разрешения 72-75 dpi. Для вывода изображения в дальнейшем на печать и распознавания текста рекомендуется устанавливать разрешении 300-600 dpi. Если исходное изображение небольшого размера и его планируется увеличить и вывести на печать, то в этом случае разрешение оригинала лучше устанавливать 600-1200 dpi. Сканирование слайдов, негативных фотопленок и качественных материалов для полиграфии требует установки величины разрешения 1200 и более dpi.

Разрешение экранного изображения. Для экранных копий изображения элементарную точку растра принято называть пикселем (*pixel*). Для измерения разрешения экранного изображения, кроме dpi, используют единица измерения *ppi* (*pixel per inch*). Размер пикселя, а значит и разрешение экранного изображения, варьируется в зависимости от выбранного разрешения экрана (из диапазона стандартных значений), разрешения оригинала и масштаба отображения. **Разрешение печатного изображения и понятие линиатуры.** Большинство находящихся в обращении печатающих устройств, от офсетных печатающих машин до простейших струйных принтеров, используют принципы полутонового растрирования.

Полутоновое растрирование (halftoning) – это способ имитации оттенков отдельными точками краски или тонера. Этот процесс основан на том, что печатающее устройство наносит на бумагу точки краски или тонера и располагает их в узлах регулярной прямоугольной сетки, которую иногда называют ***физическим растром***. Будем называть такие точки ***печатными точками***. Соседние точки физической сетки печатающего устройства объединяются в прямоугольники, которые называются ***полутоновыми ячейками*** (halftone cells). Из полутоновых ячеек образуется еще одна сетка, именуемая ***линейным растром*** (line screen). Линейный растр – это просто способ логической организации физического растра (рис. 1.3).

Частота линейного растра или количество полутоновых ячеек на единицу длины называется ***линиатурой*** и измеряется в линиях на дюйм

(line per inch, lpi).

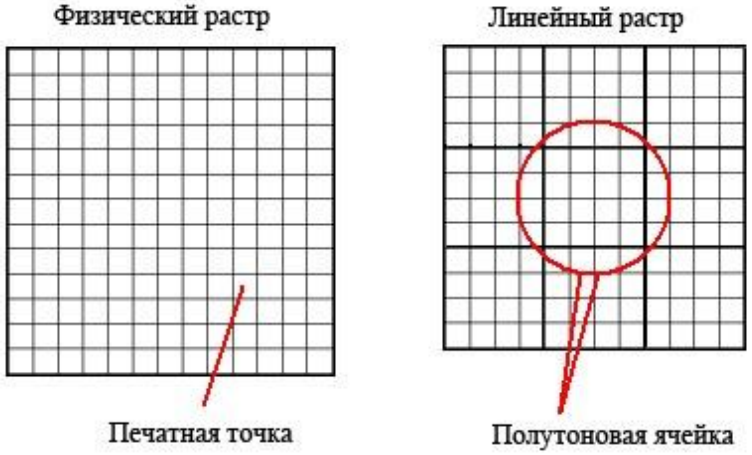


Рис. 1.3. Физический и линейный растры

При выводе на печать пиксели изображения представляются полутоновыми ячейками, а не точками физического растра печатающего устройства. Меняя заполнение полутоновых ячеек печатными точками, можно имитировать градации яркости пикселей изображения.

Рассмотрим простейшие методы растрирования оригинала в градациях серого цвета. Первым методом является метод ***растрированием с амплитудной модуляцией*** *(АМ)*, при котором иллюзия тона создается за счет формирования в центрах полутоновых ячеек, из печатных точек каких либо фигур (кругов, эллипсов, ромбов или квадратов) различного размера (рис. 1.4). Иллюзия более темного тона создается за счет увеличения радиальных размеров этих фигур и, как следствие, сокращения пробельного поля между ними при одинаковом расстоянии между центрами полутоновых ячеек.

Существует и метод ***растрирования с частотной модуляцией*** *(ЧМ)*, когда интенсивность тона регулируется изменением расстояния между соседними печатными точками одинакового размера. Таким образом, при частотно-модулированном растрировании в полутоновых ячейках с разной интенсивностью тона находится разное число печатных точек. Изображения, растрированные ЧМ-методом, выглядят более качественно, так как размер точек минимален и, во всяком случае, существенно меньше, чем средний размер фигуры при АМрастрировании. Еще более повышает качество изображения разновидность ЧМ-метода, называемая ***стохастическим растрированием***. В этом случае рассчитывается число точек, необходимое для отображения требуемой интенсивности тона в ячейке растра. Затем эти точки располагаются внутри ячейки на расстояниях, вычисленных квазислучайным методом (на самом деле используется специальный математический алгоритм),

т. е. регулярная структура растра внутри ячейки, как и на изображении в целом, вообще отсутствует. Поэтому при стохастическом ЧМ-растрировании теряет смысл понятие линиатуры растра, имеет значение лишь разрешающая способность устройства вывода. Такой способ требует больших затрат вычислительных ресурсов и высокой точности.

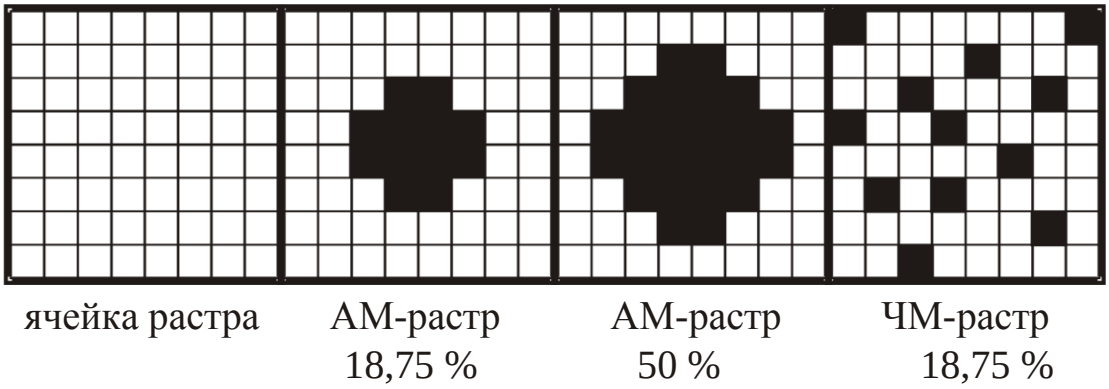


Рис. 1.4. Примеры амплитудной и частотной модуляции растра

Следующим параметром растрового изображения, который следует рассмотреть, является глубина цвета.

Глубина цвета (color depth) — это число бит, используемых для представления каждого пикселя изображения. С развитием вычислительных средств глубина цвета, хранимых в компьютере, изображений все время возрастала. Одним из первых распространенных стандартов мониторов являлся VGA, который поддерживал глубину цвета 8 бит для цветных изображений. Следующим шагом стало введение в компьютерах системы Macintosh стандарта HighColor, который кодировал цвет с глубиной 16 бит, что позволяло получить 65536 цветов. Сейчас наиболее используемым является 24-битный TrueColor, позволяющий кодировать около 16,7 млн. цветов. Однако необходимо отметить, что существуют графические системы использующие глубину цвета более чем 24 бита на пиксель.

Для лучшего понимания, что такое разрешение и глубина цвета, приведем простой пример. Вы решили отсканировать Вашу фотографию размером 10□15 см. чтобы затем обработать и распечатать на цветном принтере. Для получения приемлемого качества печати необходимо разрешение не менее 300 dpi. Считаем: 10 см = 3,9 дюйма; 15 см = 5,9 дюймов.

По вертикали: 3,9 * 300 = 1170 точек.

По горизонтали: 5,9 * 300 = 1770 точек.

Итак, число пикселей растровой матрицы 1170 * 1770 = 2 070 900.

Теперь решим, сколько цветов мы хотим использовать. Для чернобелого изображения используют обычно 256 градаций серого цвета для каждого пикселя, или 1 байт. Получаем, что для хранения нашего изображения надо 2 070 900 байт или 1,97 Мб.

Для получения качественного цветного изображения надо не менее 256 оттенков для каждого базового цвета. В модели RGB соответственно их 3: красный, зеленый и синий. Получаем общее количество байт – 3 на каждый пиксель. Соответственно, размер хранимого изображения возрастает в три раза и составляет 5,92 Мб.

Для создания макета для полиграфии фотографии сканируют с разрешением 600 dpi, следовательно, размер файла вырастает еще вчетверо.

Задание: зная, что размер экрана в пикселях 800□600, а разрешение 72 ppi, установить реальные размеры экрана в сантиметрах.

1.2. Векторное представление изображений

Для ***векторной*** графики характерно разбиение изображения на ряд графических примитивов – точки, прямые, ломаные, дуги, полигоны. Таким образом, появляется возможность хранить не все точки изображения, а координаты узлов примитивов и их свойства (цвет, связь с другими узлами и т. д.).

Вернемся к изображению на рис. 1.1. Взглянем на него по-другому. На изображении легко можно выделить множество простых объектов — отрезки прямых, ломанные, эллипс, замкнутые кривые. Представим себе, что пространство рисунка существует в некоторой координатной системе. Тогда можно описать это изображение, как совокупность простых объектов, вышеперечисленных типов, координаты узлов которых заданы вектором относительно точки начала координат (рис. 1.5).

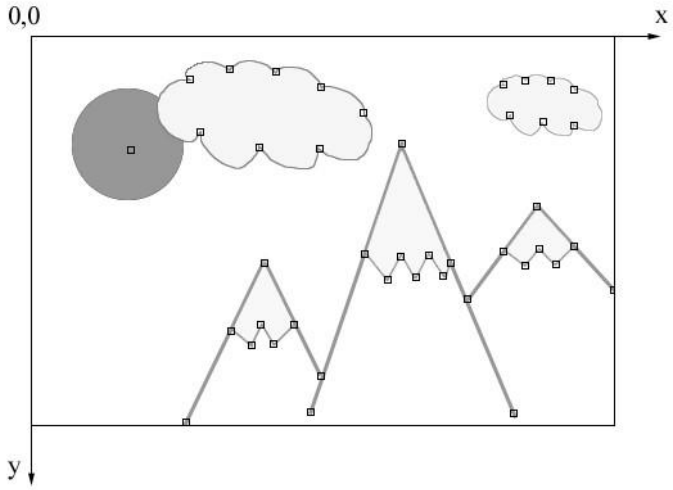


Рис. 1.5. Векторное изображение и узлы его примитивов

Проще говоря, чтобы компьютер нарисовал прямую, нужны координаты двух точек, которые связываются по кратчайшей прямой. Для дуги задается радиус и т. д. Таким образом, векторная иллюстрация – это набор геометрических примитивов.

Важной деталью является то, что объекты задаются независимо друг от друга и, следовательно, могут перекрываться между собой.

При использовании векторного представления изображение хранится в памяти как база данных описаний примитивов. Основные графические примитивы, используемые в векторных графических редакторах: точка, прямая, кривая Безье, эллипс (окружность), полигон (прямоугольник). Примитив строится вокруг его узлов (nodes). Координаты узлов задаются относительно координатной системы макета.

А изображение будет представлять из себя массив описаний – нечто типа:

```
отрезок (20,20-100,80); окружность(50,40-30);
кривая_Безье (20,20-50,30-100,50).
```

Каждому узлу приписывается группа параметров, в зависимости от типа примитива, которые задают его геометрию относительно узла. Например, окружность задается одним узлом и одним параметром – радиусом. Такой набор параметров, которые играют роль коэффициентов и других величин в уравнениях и аналитических соотношениях объекта данного типа, называют аналитической моделью примитива. Отрисовать примитив – значит построить его геометрическую форму по его параметрам согласно его аналитической модели.

Векторное изображение может быть легко масштабировано без потери деталей, так как это требует пересчета сравнительно небольшого числа координат узлов. Другой термин – «object-oriented graphics».

Самой простой аналогией векторного изображения может служить аппликация. Все изображение состоит из отдельных кусочков различной формы и цвета (даже части растра), «склеенных» между собой. Понятно, что таким образом трудно получить фотореалистичное изображение, так как на нем сложно выделить конечное число примитивов, однако существенными достоинствами векторного способа представления изображения, по сравнению с растровым, являются:

- векторное изображение может быть легко масштабировано без потери качества, так как это требует пересчета сравнительно небольшого числа координат узлов;

• графические файлы, в которых хранятся векторные изображения, имеют существенно меньший, по сравнению с растровыми, объем (порядка нескольких килобайт).

На самом деле размер векторного изображения зависит от количества объектов на изображении. И чем ближе качество векторного рисунка будет приближаться к фотореалистичному изображению, тем большей размер будет у файла.

Сферы применения векторной графики очень широки. В полиграфике – от создания красочных иллюстраций до работы со шрифтами. Все, что мы называем машинной графикой, 3D-графикой, графическими средствами компьютерного моделирования и САПР – все это сферы приоритета векторной графики, ибо эти ветви дерева компьютерных наук рассматривают изображение исключительно с позиции его математического представления.

Как видно, векторным можно назвать только способ описания изображения, а само изображение для нашего глаза всегда растровое. Таким образом, задачами векторного графического редактора являются растровая прорисовка графических примитивов и предоставление пользователю сервиса по изменению параметров этих примитивов. Все изображение представляет собой базу данных примитивов и параметров макета (размеры холста, единицы измерения и т. д.). Отрисовать изображение – значит выполнить последовательно процедуры прорисовки всех его деталей.

С другой стороны, если изображение состоит из простых объектов, то для его хранения в векторном виде необходимо не более нескольких килобайт.

1.3. Представление изображений с помощью фракталов

В последние время фракталы стали очень популярны. Большую роль в этом сыграла книга франко-американского математика Бенуа Мандельброта "Фрактальная геометрия природы", изданная в 1975 году. Что же такое фрактал?

Фрактал (лат. fractus — дробленый, сломанный, разбитый) — сложная геометрическая фигура, обладающая свойством самоподобия, то есть составленная из нескольких частей, каждая из которых подобна всей фигуре целиком. Свойство самоподобия характерно для многих природных объектов. Таким свойством обладают, например, ветки деревьев, снежинки, границы облаков и морских побережий, трещины в камнях, структуры некоторых веществ, полученных с помощью электронного микроскопа и т. д. ***Фрактальная геометрия*** позволяет описать и получить изображения таких природных объектов с помощью математических средств. В компьютерной

графике фракталы, могут использоваться не только для генерации изображений сложных объектов, но и для сжатия изображений.

Для классификации фракталов часто используют деление на следующие классы:

▪ геометрические фракталы; ▪ алгебраические фракталы; ▪ стохастические фракталы.

Существуют и другие классификации фракталов, например деление фракталов на детерминированные (алгебраические и геометрические) и недетерминированные (стохастические). Подробно рассмотрим геометрические и алгебраические фракталы.

1.3.1. Геометрические фракталы

Фракталы этого класса самые наглядные. В двухмерном случае их получают с помощью некоторой ломаной (или поверхности в трехмерном случае), называемой генератором. За один шаг алгоритма каждый из отрезков, составляющих ломаную, заменяется на ломануюгенератор, в соответствующем масштабе. В результате бесконечного повторения этой процедуры, получается геометрический фрактал.

Рассмотрим один из таких фрактальных объектов - ***триадную кривую Коха***. Построение кривой начинается с отрезка единичной длины (рис. 1.6) - это 0-е поколение кривой Кох. Далее каждое звено (в нулевом поколении один отрезок) заменяется на *образующий элемент*, обозначенный на рис.1 через **n=1.** В результате такой замены получается следующее поколение кривой Кох. В 1-ом поколении - это кривая из четырех прямолинейных звеньев, каждое длиной по **1/3**. Для получения 3-го поколения проделываются те же действия - каждое звено заменяется на уменьшенный образующий элемент. Итак, для получения каждого последующего поколения, все звенья предыдущего поколения необходимо заменить уменьшенным образующим элементом. Кривая **n**-го поколения при любом конечном **n** называется ***предфракталом***. На рис. 1.6 представлены пять поколений кривой. При **n** стремящемся к бесконечности кривая Кох становится фрактальным объектом.

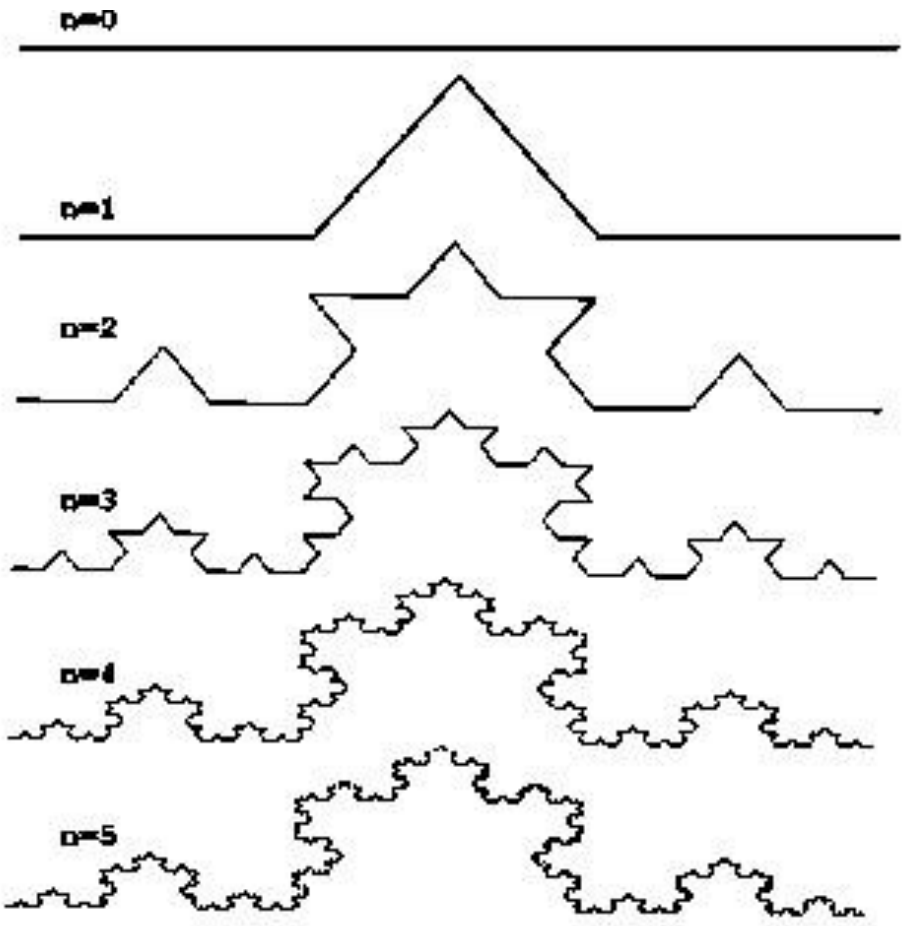


Рис. 1.6. Построение триадной кривой Коха

Три копии кривой Коха, построенные (остриями наружу) на сторонах правильного треугольника, образуют замкнутую кривую, называемую ***снежинкой Коха***.

Алгоритм построение фрактала можно описать словесно, как показано на примере построение кривой Коха, либо математически, как будет показано в следующем разделе для алгебраических фракталов. Но существует еще один способ описания алгоритма построения геометрических фракталов с помощью ***L-систем (системы Линдемайера)***. *L*-система это формальная грамматика, используемая для моделирования процессов роста и развития растений.

Изначально *L*-системы были введены при изучении формальных языков, а также использовались в биологических моделях селекции. С их помощью можно строить многие известные самоподобные фракталы, включая снежинку Коха и ковер Серпинского.

L-система определяется как кортеж $G=(V, w, P)$, где V – алфавит, представляющий набор символов, содержащих элементы которые могут быть представлены графически, w – аксиома, которая представляет собой строку символов из V, определяющая начальное состояния системы, P - представляет

собой набор правил производства новой строки, путем замены символов текущего состояния системы на ряд новых. Правила грамматики *L*-системы применяются итеративно, начиная с начального состояния.

Рассмотрим *L*-систему, где алфавит состоит всего из двух символов $V=\{a,b\}$. Аксиома $w=a$. Правила производства описываются как $p1$: $a \rightarrow ab$, $p2$: $b \rightarrow ab$. Следуя итеративному принципу, каждое поколение кривой удваивается на каждом этапе: *a*, *ab*, *abab*, *abababab*. Графически *a* и *b* отображаются как два равных отрезка, соединенных под прямым углом (рис. 1.7).

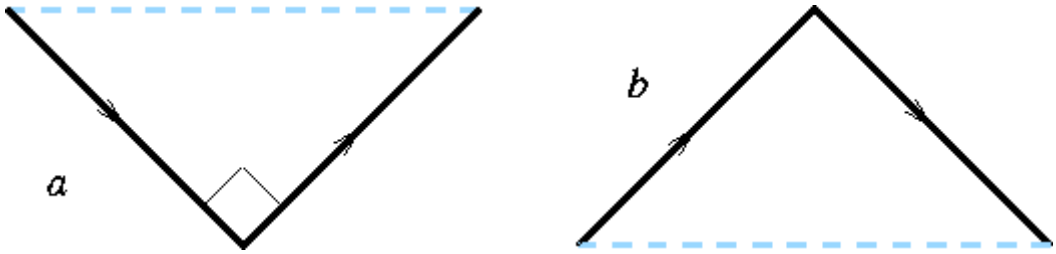


Рис. 1.7. Графическое представление алфавита *L*-системы Графическое представление правил $p1$ и $p2$, приведено на рис. 1.8. Таким образом, правила описывают замену каждого из отрезков двумя другими, соединенных под прямым углом. При этом, каждый раз, меняется угол поворота новых отрезков относительно исходного.

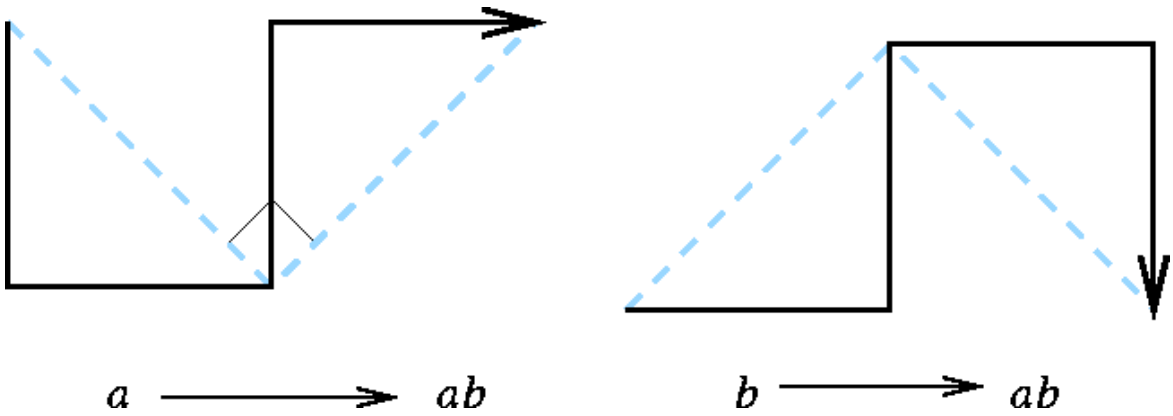


Рис. 1.8. Графическое представление правил *L*-системы

На рис. 1.9 изображены пять первых итераций процесса построения фрактала.

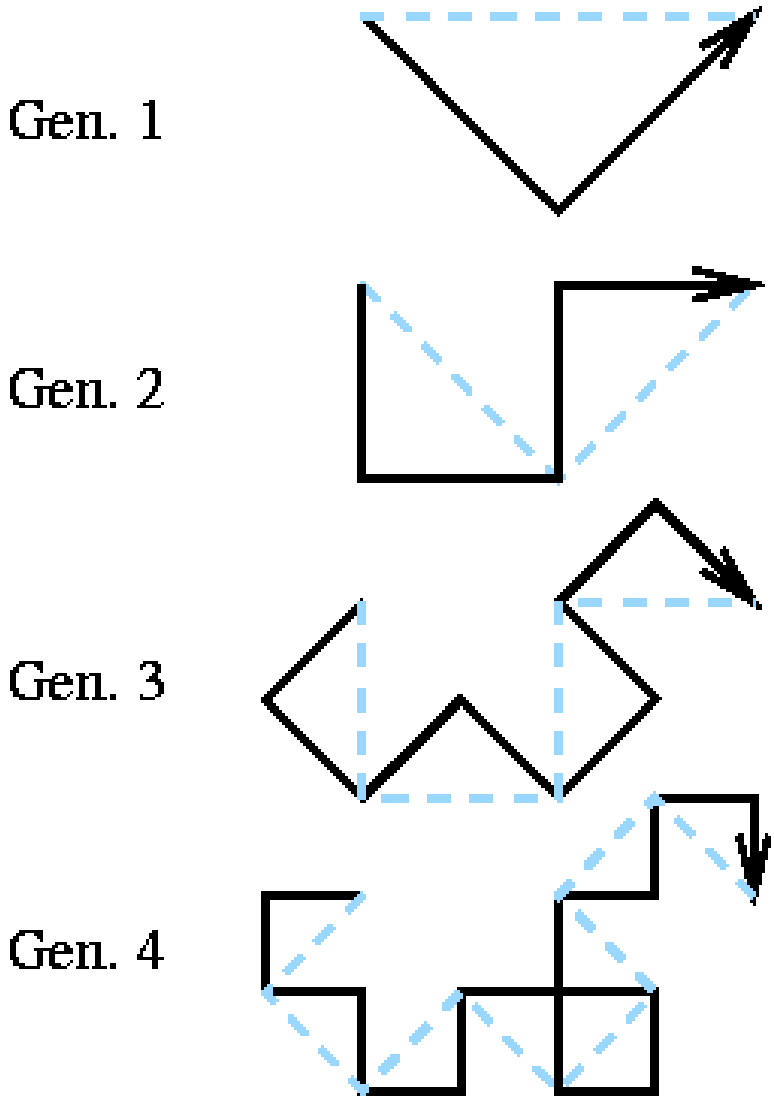


Рис. 1.9. Первые итерации построения фрактальной кривой

Такая предельная фрактальная кривая (при числе итераций стремящимся к бесконечности) называется ***драконом Хартера-Хейтуэя*** и представлена на рис. 1.10.

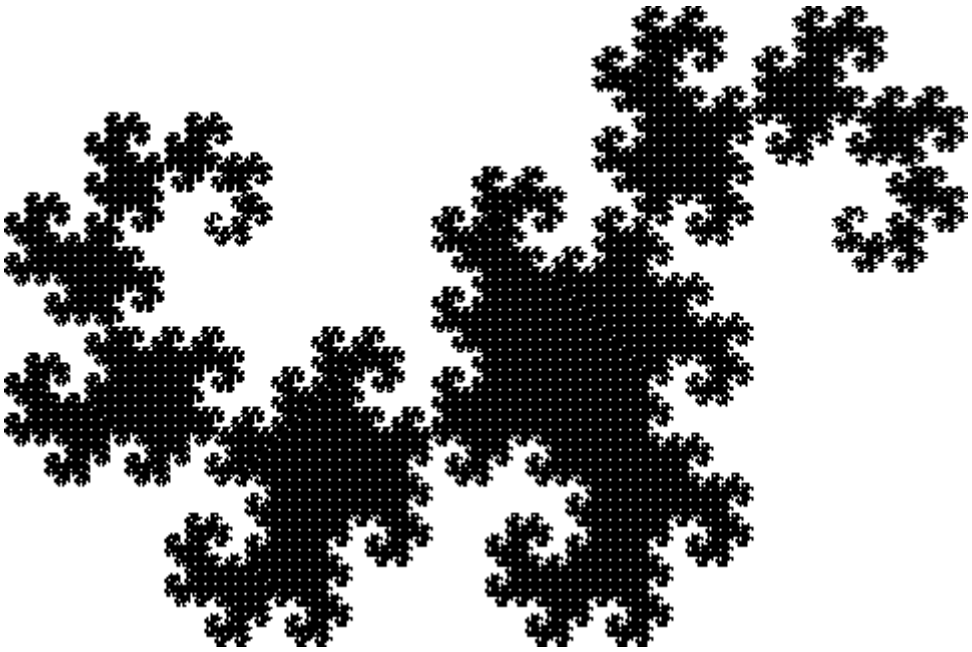


Рис. 1.10. Дракон Хартера-Хейтуэя

Некоторые из геометрических фракталов можно построить исходя из первоначально закрашенной плоской фигуры. Примером этого может служить метод построения треугольника Серпинского. Процесс построения можно описать следующим образом.

Равносторонний треугольник M_0 делится прямыми, параллельными его сторонам, на 4 равных равносторонних треугольника. Из треугольника удаляется центральный треугольник. Получается множество M_1, состоящее из 3 оставшихся треугольников "первого ранга". Поступая точно так же с каждым из треугольников первого ранга, получим множество M_2, состоящее из 9 равносторонних треугольников второго ранга. Продолжая этот процесс бесконечно, получим бесконечную последовательность M_0, M_1,…, M_n,…пересечение членов которой есть треугольник Серпинского (рис. 1.11).

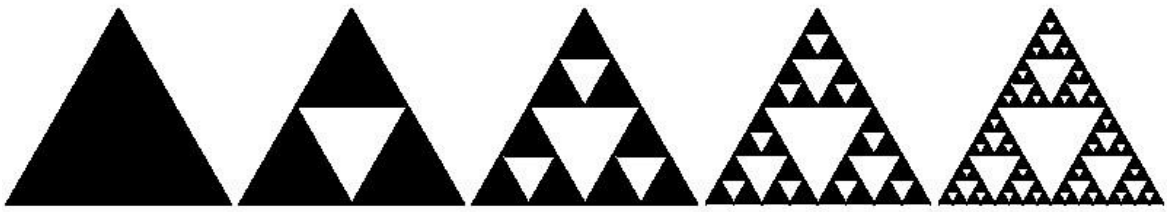


Рис. 1.11. Построение салфетки Серпинского

1.3.2. Алгебраические фракталы

Это самая крупная группа фракталов. Получают их с помощью нелинейных процессов в **n**-мерных пространствах. Наиболее изучены двухмерные процессы. Интерпретируя нелинейный итерационный процесс, как дискретную динамическую систему, можно пользоватся терминологией теории этих систем: *фазовое пространство*, *аттрактор* и т.д. [2].

Фазовое пространство — пространство, на котором представлено множество всех состояний системы, так, что каждому возможному состоянию системы соответствует точка фазового пространства.

Известно, что нелинейные динамические системы обладают несколькими устойчивыми состояниями. То состояние, в котором оказалась динамическая система после некоторого числа итераций, зависит от ее начального состояния. Поэтому каждое устойчивое состояние (или как говорят - ***аттрактор***) обладает некоторой областью начальных состояний, из которых система обязательно попадет в рассматриваемые конечные состояния.

Таким образом, фазовое пространство системы разбивается на области притяжения аттракторов. Если фазовым является двухмерное пространство, то окрашивая области притяжения различными цветами, можно получить цветовой фазовый портрет этой системы (итерационного процесса). Меняя алгоритм выбора цвета, можно получить сложные фрактальные картины с причудливыми многоцветными узорами. Неожиданностью для математиков стала возможность с помощью примитивных алгоритмов порождать очень сложные нетривиальные структуры.

В качестве примера рассмотрим множество Мандельброта.

Множество Мандельброта — это фрактал, определѐнный как множество точек *c* на комплексной плоскости, для которых итеративная последовательность

$z_0 = 0$ $z_{n+1} = z_{n2} + c$ не уходит в бесконечность.

Таким образом, вышеуказанная последовательность может быть раскрыта для каждой точки *c* на комплексной плоскости следующим образом:

$c = x + i \cdot y$

$Z_0 = 0$

$Z_1 = Z_0^2 + c$

$= x + i \cdot y$

$Z_2 = Z_1^2 + c$

$= (x + i \cdot y)^2 + x + i \cdot y$

$= x^2 + 2 \cdot i \cdot x \cdot y - y^2 + x + i \cdot y$

$= x^2 - y^2 + x + (2 \cdot x \cdot y + y) \cdot i$ $Z_3 = Z_2^2 + c = \ldots$ и так далее.

Если переформулировать эти выражения в виде итеративной последовательности значений координат комплексной плоскости x и y, т. е. заменив z_n на $x_n + i \cdot y$, а c на $p + i \cdot q$, мы получим:

$$x_{n+1} = x_n^2 - y_n^2 + p \qquad (*)$$
$$y_{n+1} = 2 \cdot x_n \cdot y_n + q$$

Суть метода построения множества Мандельброта состоит в следующем. Для каждого пикселя, отображающего некоторую точку с координатами (a, b), проводят серию вычислений по формулам (*). При этом исходные значения x_0 и y_0 для каждой новой точки (a, b) изначально всегда равны нулю. На каждом шаге, кроме очередных значений x_{i+1} и y_{i+1}, вычисляют величину $r_i = \sqrt{x_i + y_i}$. Эта величина представляет собой не что иное, как расстояние от точки (x_i, y_i) до начала координат. Точка (a, b) считается принадлежащей множеству Мандельброта, если она в процессе вычислений никогда не удаляется от начала координат на критическое расстояние, большее или равное двум. Такой пиксель окрашивается в черный цвет. Для всех прочих значений a и b величина r_i может переходить запретный рубеж в две единицы за разное количество шагов. В зависимости от этого, точку окрашивают в соответствующий цвет.

Критическое значение при программировании имеет скорость вычислений. Существенно влияют на время расчетов операции возведения в степень и извлечение квадратного корня. Возведение в квадрат целесообразно заменить умножением. Возводить координаты в квадрат следует лишь один раз, сохраняя результаты в промежуточных переменных (`x2` и `y2`). От извлечения квадратного корня вообще следует отказаться. Вместо самого значения расстояния r_i можно вычислять лишь сумму квадратов $x^2 + y^2$, сравнивая ее не с двойкой, а с четверкой.

Функции для генерации множества Мандельброта приведены ниже. В зависимости от полученного значения `k` (обратного счетчика итераций), очередная точка закрашивается в соответствующий цвет (в данном случае точка окрашивается в различные градации серого цвета). Если `k` оказывается равным нулю, то точка с координатами (a, b) принадлежит множеству Мандельброта. В отношении всех прочих точек имеет значение, насколько велико или мало значение `k`. Оно характеризует скорость убегания величины r_i за дозволенный предел при данных значениях a и b.

```
    private Color GetColor(double a, double b) //
Функция, возвращающая цвет точки (a, b)
            {
                double x = 0, y = 0, r = 0;
                //количество итераций для каждой точки
int k = 100;             while ((r<4) && (k>0))
                    {
                    Double x2 = x * x;
                    Double y2 = y * y;
Double xy = x * y;                    x = x2 - y2 + a;
y = 2 * xy + b;                   r = x2 + y2;
k--;}             k = 255*k/100;
                Color p = Color.FromArgb(255, k, k, k);
return p;
            }  private void Mbut_Click(object sender,
EventArgs e)
    //Функция построения множества Мандельброта
    //в квадрате размером 400 x 400 пикселей
            {
                //Определение области построения
const double xmin = -2;             const double xmax =
1;             const double ymin = -1.5;
const double ymax = 1.5;

                //длинна стороны квадрата
const int n = 400;             //определение приращения
по x и по y             double hx = (xmax - xmin) / n;
double hy = (ymax - ymin) / n;

                Bitmap bmp;
                bmp = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);
                 double x = xmin;             double y =
ymax;

                for (int j = 1; j < n; j++)
                {
                    for (int i = 1; i < n; i++)
                    {
                        Color c = GetColor(x, y);
bmp.SetPixel(i, j, c);                    x = x + hx;
```

```
                    }                   y = y - hy;
x = xmin;
                }
                Graphics gr = CreateGraphics();
gr.DrawImage(bmp, 0, 0);
            }
```

Вышеописанный алгоритм дает приближение к так называемому множеству Мандельброта. Множеству Мандельброта принадлежат точки, которые в течение *бесконечного* числа итераций не уходят в бесконечность (точки, закрашенные в черный цвет). Точки, принадлежащие границе множества (именно там возникает сложные структуры) уходят в бесконечность за конечное число итераций, а точки, лежащие за пределами множества, уходят в бесконечность через несколько итераций (белый фон).

Пример работы программы и приближения к множеству Мандельброта в диапазоне для x от -2 до 1 и для y от -1,5 до 1,5 приведен на рис. 1.12.

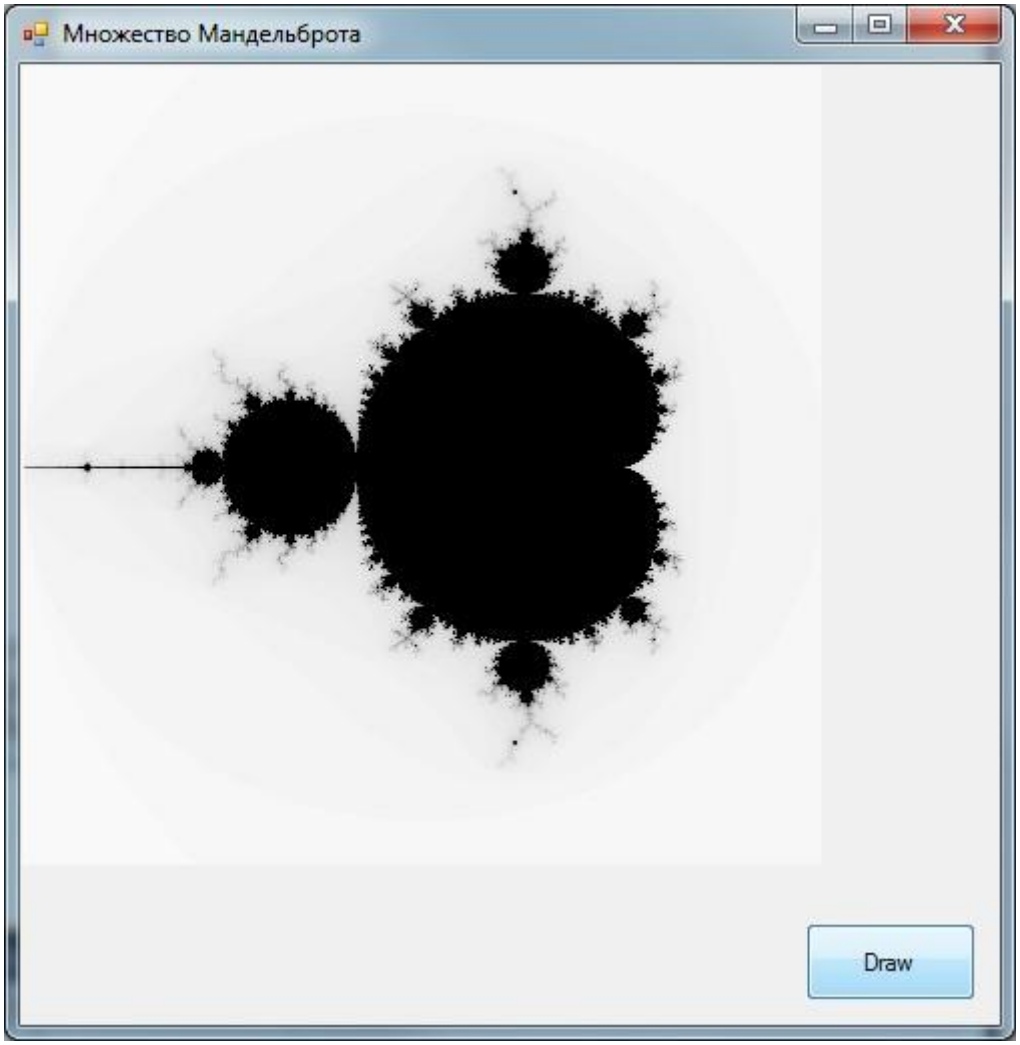


Рис. 1.12. Множество Мандельброта

Используя туже самую итеративную последовательность $z_{n+1} = z_n^2 + c$, что и для построения множества Мандельброта, можно построить другой алгебраический фрактал: ***множество Жюлиа***. Множество Жюлиа получается, если зафиксировать в формуле значение комплексной константы ($a+i \cdot b$), которая будет одинакова для всех точек, а начальные значения *x* и *y* принимать равными значениям координатам вычисляемой точки. В этом случае, код функции получения цвета точки будет выглядеть следующим образом:

```
            private Color GetColorJ(double x, double y)
            {
                //задаем произвольно начальную точку
double a = -0.55, b = -0.55             double r = 0;
int k = 100;              while ((r < 4) && (k > 0))
                {
                    Double x2 = x * x;
                    Double y2 = y * y;
Double xy = x * y;                  x = x2 - y2 + a;
y = 2 * xy + b;                 r = x2 + y2;
k--;              }
                k = 255 * k / 100;
                Color p = Color.FromArgb(255, k, k, k);
return p;
            }
```

Еще одним известным классом фракталов являются стохастические фракталы, которые получаются в том случае, если в итерационном процессе случайным образом менять какие-либо его параметры. При этом получаются объекты очень похожие на природные - несимметричные деревья, изрезанные береговые линии и т.д. Двумерные стохастические фракталы используются при моделировании рельефа местности и поверхности моря.

1.3.3. Системы итерируемых функций

Метод "Систем Итерируемых Функций" (Iterated Functions System - IFS) появился в середине 80-х годов как простое средство получения фрактальных структур.

IFS представляет собой систему функций из некоторого фиксированного класса функций, отображающих одно многомерное множество на другое. Наиболее простая IFS состоит из аффинных преобразований плоскости:

$X_1' = A*X + B*Y + C$
$Y_1' = D*X + E*Y + F$
$X_2' = G*X + H*Y + I$
$Y_2' = J*X + K*Y + L$

В 1988 году известные американские специалисты в теории динамических систем и эргодической теории Барнсли и Слоан предложили некоторые идеи, основанные на соображениях теории динамических систем, для сжатия и хранения графической информации. Они назвали свой метод методом фрактального сжатия информации. Происхождение названия связано с тем, что геометрические образы, возникающие в этом методе, обычно имеют фрактальную природу в смысле Мандельброта.

На основании этих идей Барнсли и Слоан создали алгоритм, который, по их утверждению, позволит сжимать информацию в 5001000 раз. Вкратце метод можно описать следующим образом. Изображение кодируется несколькими простыми преобразованиями (в нашем случае аффинными), т.е. коэффициентами этих преобразований.

Например, закодировав какое-то изображение двумя аффинными преобразованиями, мы однозначно определяем его с помощью 12-ти коэффициентов. Если теперь задаться какой-либо начальной точкой (например X=0 Y=0) и запустить итерационный процесс, то мы после первой итерации получим две точки, после второй - четыре, после третьей - восемь и т.д. Через несколько десятков итераций совокупность полученных точек будет описывать закодированное изображение. Но проблема состоит в том, что очень трудно найти коэффициенты IFS, которая кодировала бы произвольное изображение.

Для построения IFS применяют кроме аффинных и другие классы простых геометрических преобразований, которые задаются небольшим числом параметров.

2. Представление цвета в компьютере

2.1. Свет и цвет

Понятия света и цвета в компьютерной графике тесно связаны и являются основополагающими.

Свет может рассматриваться либо как электромагнитная волна, либо как поток фотонов. Одной из характеристик электромагнитной волны является ее длинна λ. Видимый свет имеет длину волн в диапазоне 400-700 нм. Свет принимается либо непосредственно от источника, например, от экрана

монитора, либо косвенно при отражении от поверхности объекта или преломлении в нем.

На практике мы редко сталкиваемся со светом какой то определенной длинны волны. Напротив, видимый свет практически всегда состоит из сочетания фотонов разных длин волн.

Источник или объект является ***ахроматическим***, если наблюдаемый свет содержит все видимые длины волн в приблизительно равных количествах. Ахроматический источник кажется белым, а отраженный или преломленный ахроматический свет — белым, черным или серым. Белыми выглядят объекты, ахроматически отражающие более 80% света белого источника, а черными — менее 3%.

Промежуточные значения дают различные оттенки серого.

Если воспринимаемый свет содержит длины волн в произвольных неравных количествах, то он называется ***хроматическим***.

Монохроматический - это такой свет, который имеет одну длину волны или частоту.

Понятие цвета тесно связано с тем, как человек воспринимает свет. Можно сказать, что ощущение цвета формируется человеческим мозгом в результате анализа электромагнитного излучения (света), попадающего на сетчатку глаз.

Считается, что в глазе человека существует три группы цветовых рецепторов (колбочек), каждая из которых чувствительна к определенному диапазону длин волны. Каждая группа формирует один из трех основных цветов: красный, зеленый, синий.

Если длины волн светового потока сконцентрированы у верхнего края видимого спектра (около 700 Нм), то свет воспринимается как красный. Если длины волн сконцентрированы у нижнего края видимого спектра (около 400 Нм), то свет воспринимается как синий. Если длины волн сконцентрированы в середине видимого спектра (около 550 Нм), то свет воспринимается как зеленый.

С помощью экспериментов, построенных на этой гипотезе, были получены кривые реакции глаза, показанные на рис. 2.1.

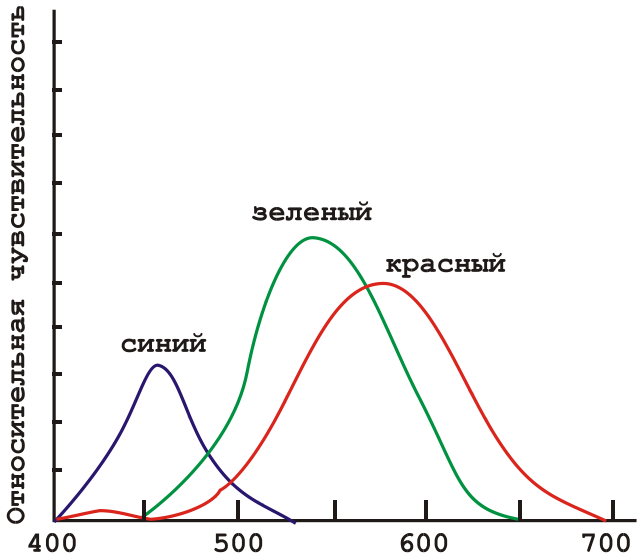


Рис. 2.1. Кривые реакции глаза

При описании цвета используют три его субъективных атрибута: ***цветовой тон***, ***насыщенность*** и ***светлоту***. Разделение признака цвета на эти взаимосвязанные компоненты есть результат мысленного процесса, существенно зависящего от навыка и обучения.

Наиболее важный атрибут цвета – ***цветовой тон (hue)*** – ассоциируется в человеческом сознании с обусловленностью окраски предмета определенным типом пигмента, краски, красителя. Тон определяется характером распределения излучения в спектре видимого света. Именно тон определяет название цвета, например «красный», «синий», «зелѐный».

Насыщенность (saturation) характеризует степень, уровень, силу выражения цветового тона. Этот атрибут в человеческом сознании связан с количеством (концентрацией) пигмента, краски, красителя. Можно сказать, что насыщенность цвета показывает, насколько данный цвет отличается от монохроматического («чистого») излучения того же светового тона. Насыщенность характеризует степень ослабления (разбавления) данного цвета белым и позволяет отличать розовый от красного, голубой от синего. Считается, что серые тона (ахроматические) не имеют насыщенности и различаются лишь по светлоте.

Светлота (lightness, value) – это различимость участков, сильнее или слабее отражающих свет. Уровень светлоты окрашенных объектов определяется при сравнении их с ахроматическими объектами и при выявлении степени их приближения к белому цвету, отражающему максимум света.

Со светлотой тесно связано физическое понятие ***яркости света (luminance)***. Чем больше яркость, тем больше светлота. Поэтому можно сказать, что светлота есть мера ощущения яркости. С другой стороны, во

многих источниках, вместо понятия светлота часто используют субъективное понятие ***яркости (brightness)***.

2.2. Цветовые модели и пространства

Как видим из вышеизложенного, описание цвета может опираться на составление любого цвета на основе основных цветов или на такие понятия, как светлота, насыщенность, цветовой тон. Применительно к компьютерной графике описание цвета также должно учитывать специфику аппаратуры для ввода/вывода изображений. В связи с необходимостью описания различных физических процессов воспроизведения цвета были разработаны различные цветовые модели. Цветовые модели позволяют с помощью математического аппарата описать определенные цветовые области спектра. Цветовые модели описывают цветовые оттенки с помощью смешивания нескольких основных цветов.

Основные цвета разбиваются на оттенки по яркости (от темного к светлому), и каждой градации яркости присваивается цифровое значение (например, самой темной – 0, самой светлой – 255). Считается, что в среднем человек способен воспринимать около 256 оттенков одного цвета. Таким образом, любой цвет можно разложить на оттенки основных цветов и обозначить его набором цифр – цветовых координат.

Таким образом, при выборе цветовой модели можно определять обычно трехмерное цветовое координатное пространство, внутри которого каждый цвет представляется точкой. Такое пространство называется пространством цветовой модели.

Профессиональные графические программы обычно позволяют оперировать с несколькими цветовыми моделями, большинство из которых создано для специальных целей или особых типов красок: CMY, CMYK, CMYK256, RGB, HSB, HLS, L*a*b, YIQ, Grayscale

(Оттенки серого) и Registration color. Некоторые из них используются редко, диапазоны других перекрываются.

2.2.1. Цветовая модель RGB

В основе одной из наиболее распространенных цветовых моделей, называемой RGB моделью, лежит воспроизведение любого цвета путем сложения трех основных цветов: красного (Red), зеленого (Green) и синего (Blue). Каждый канал - R, G или B имеется свой отдельный параметр, указывающий на количество соответствующей компоненты в конечном цвете. Например: (255, 64, 23) – цвет, содержащий сильный красный компонент, немного зелѐного и совсем немного синего. Естественно, что этот режим

наиболее подходит для передачи богатства красок окружающей природы. Но он требует и больших расходов, так как глубина цвета тут наибольшая – 3 канала по 8 бит на каждый, что дает в общей сложности 24 бита.

Поскольку в RGB модели происходит сложение цветов, то она называется ***аддитивной (additive)***. Именно на такой модели построено воспроизведение цвета современными мониторами.

Значения координат R, G и B можно считать принадлежащими отрезку [0,1], что представляет пространство RGB в виде единичного куба (рис. 2.2).

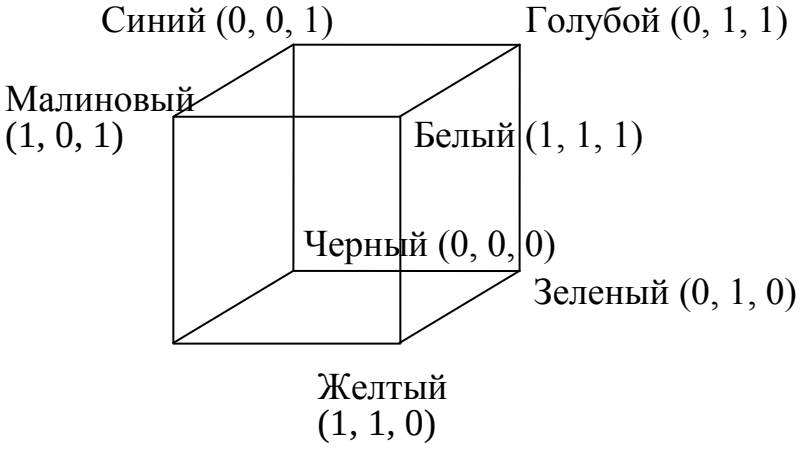


Рис. 2.2. Цветовое пространство RGB модели

2.2.2. Субтрактивные цветовые модели

Модель CMY использует также три основных цвета: Cyan (голубой), Magenta (пурпурный, или малиновый) и Yellow (желтый). Эти цвета описывают отраженный от белой бумаги свет трех основных цветов RGB модели. Поэтому можно описать соотношения между RGB и CMY моделями следующим образом:

□ *C* □ □1□ □*R*□ □*R*□ □1□ □ *C* □ □ □ □ □ □ □ □ □ □ □ □ □ □
□*M* □ □ □1□□□*G*□ □*G*□ □ □1□□□*M* □. □□ *Y* □□ □□1□□ □□*B*□□

□□*B*□□ □□1□□ □□ *Y* □□

Модель CMY является ***субтрактивной*** (основанной на вычитании) цветовой моделью. Как уже говорилось, в CMY-модели описываются цвета на белом носителе, т. е. краситель, нанесенный на белую бумагу, вычитает часть

спектра из падающего белого света. Например, на поверхность бумаги нанесли голубой (Cyan) краситель. Теперь красный свет, падающий на бумагу, полностью поглощается. Таким образом, голубой носитель вычитает красный свет из падающего белого.

Такая модель наиболее точно описывает цвета при выводе изображения на печать, т. е. в полиграфии.

Поскольку для воспроизведения черного цвета требуется нанесение трех красителей, а расходные материалы дороги, использование CMYмодели является не эффективным. Дополнительный фактор, не добавляющий привлекательности CMY-модели, – это появление нежелательных визуальных эффектов, возникающих за счет того, что при выводе точки три базовые цвета могут ложиться с небольшими отклонениями. Поэтому к базовым трем цветам CMY-модели добавляют черный (blac**K**) и получают новую цветовую ***модель CMYK***.

Для перехода из модели CMY в модель CMYK используют следующее соотношение:

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

$$C = C - K;$$

$$M = M - K;\ Y = Y - K.$$

Соотношения преобразования RGB в CMY и CMY в CMYK-модель верны лишь в том случае, когда спектральные кривые отражения для базовых цветов не пересекаются. Поэтому в общем случае можно сказать, что существуют цвета, описываемые в RGB-модели, но не описываемые в CMYK-модели. Это означает, что цветовой охват модели CMYK значительно ниже цветового охвата RGB-модели.

Существует также модель CMYK256, которая используется для более точной передачи оттенков при качественной печати изображений.

2.2.3. Модели HSV и HSL

Рассмотренные модели ориентированы на работу с цветопередающей аппаратурой и для некоторых людей неудобны. Поэтому модели HSV, HLS опираются на понятия тона, насыщенности и яркости (светлоты).

В цветовом пространстве модели HSV (Hue, Saturation, Value), иногда называемой HSB (Hue, Saturation, Brightness), используется цилиндрическая система координат, а множество допустимых цветов представляет собой шестигранный конус, поставленный на вершину.

Основание конуса представляет яркие цвета и соответствует $V = 1$. Тон (H) измеряется углом, отсчитываемым вокруг вертикальной оси OV.

При этом красному цвету соответствует угол 0□, зелѐному – 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* (насыщенность) изменяется от 0 на оси *OV* 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 *OV* (*S* = 0) соответствует ахроматическим цветам (серым тонам).

Процесс добавления белого цвета к заданному можно представить как уменьшение насыщенности *S*, а процесс добавления чѐрного цвета – как уменьшение яркости *V*. Основанию шестигранного конуса соответствует проекция RGB куба вдоль его главной диагонали.

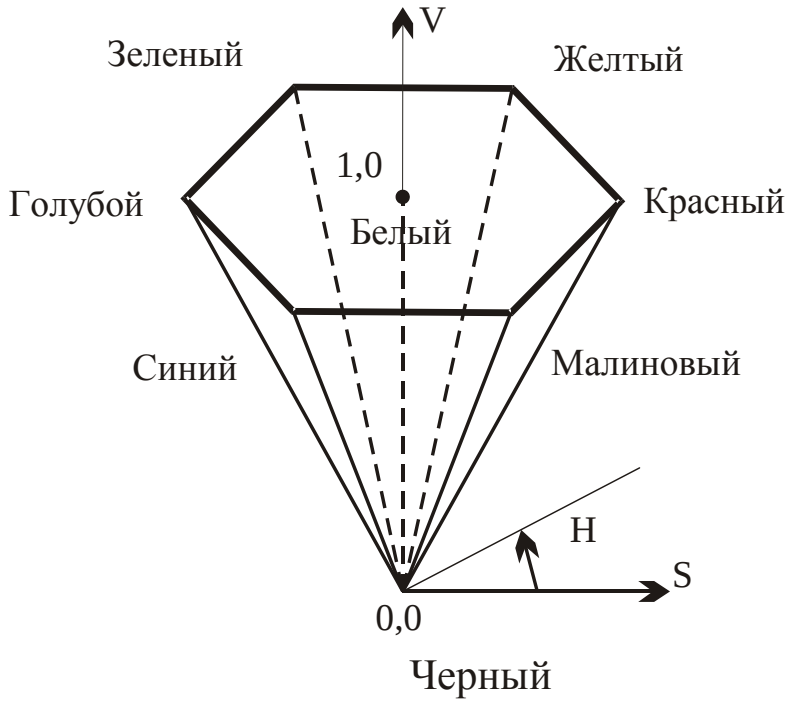


Рис. 2.3. Цветовое пространство HSV модели

Еще одним примером системы, построенной на интуитивных понятиях тона насыщенности и светлоты, является система HLS (Hue, Lightness, Saturation). Здесь множество всех цветов представляет собой два шестигранных конуса, поставленных друг на друга (основание к основанию).

Алгоритмы преобразования RGB в HSV и в HLS и обратные преобразования рассмотрены в [7].

2.3. Системы управления цветом

Ситуация, когда дизайнерам и полиграфистам приходится работать в разных цветовых пространствах, приводит к возникновению ошибок в цветопередаче на этапе перехода от одной цветовой модели к другой. Альтернативой такого подхода является использование так называемой аппаратно независимой системы управления цветом (color managing system, CMS). Суть этой технологии состоит в том, чтобы независимо от программного обеспечения цвет передавался от одного этапа обработки (например, сканирования) к другой (печати) без искажений. Таким образом, обеспечивается видимая однородность цветового пространства для всех периферийных устройств и приложений, работающих в системе. Изначальная поддержка этой технологии на платформе

Macintosh (CMS ColorSync) была долгое время причиной предпочтения этой платформы специалистами в области графики. В последнее время много говорится о работе с этой технологией корпорации Microsoft.

CMS должна учитывать, что все устройства вывода (как принтеры, так и мониторы) имеют фиксированный диапазон воспроизводимых цветов и тонов, называемый ***цветовой гаммой***. При этом цветовая гамма естественным образом ограничивается наиболее насыщенными цветами, с которыми приходится работать устройству, т. е. его основными цветами. Поэтому на экране монитора нельзя получить более насыщенный красный цвет, чем у его люминофора, или более насыщенный зеленый цвет при печати на принтере, чем в результате смешения голубой и желтой красок.

Область цветовой гаммы меньше, чем цветовой охват практически любой цветовой модели. Второй проблемой, которую должна решать CMS, является то, что устройства ввода (сканеры и цифровые камеры) не имеют цветовой гаммы, поскольку не существует резкой границы между воспринимаемыми и не воспринимаемыми ими цветами, что бы они ни фиксировали.

Несмотря на то, что у устройств ввода (сканера и цифровой камеры) отсутствует цветовая гамма, у них есть фиксированный ***динамический диапазон*** — пределы изменения уровней яркости, при которых эти устройства способны работать. Ниже определенного уровня темноты (или плотности) сканер или цифровая камера уже не в состоянии различать уровни яркости и возвращает нулевое значение, обозначающее темноту. Аналогично, выше определенного уровня яркости устройства ввода не способны фиксировать изменение яркости, что достаточно редко наблюдается у сканеров и слишком часто — у цифровых камер.

Как правило, устройства ввода обладают более широким динамическим диапазоном, чем устройства вывода. Дополнительной сложностью для CMS является не совпадение цветовых гамм различных устройств.

Для решения перечисленных проблем были разработаны так называемые ***профили устройств***, описывающие различие в представлении цвета между устройством и определенной цветовой моделью. Также, профиль устройства содержит информацию о трех параметрах, описывающих режим работы устройства:

- Цветовой гамме — цвете и яркости красителей (основных цветов); ▪ Динамическом диапазоне — цвете и яркости белой и черной точек;
- Характеристиках тоновоспроизведения красителей.

Сегодня большая часть профессионального и полупрофессионального оборудования поставляется с профилями. Однако эти профили обычно являются универсальными для определенной модели, а не для вашего конкретного изделия, поэтому необходима настройка параметров устройства в соответствии с некоторым эталоном – ***калибровка.***

Эталонные значения на описание характеристик воспроизведения цвета устанавливает Международный консорциум по цвету (ICC – International Color Consortium), созданный в 1993 г. Поэтому CMS обычно включают в себя средства калибровки, т. е. средства настройки конкретного экземпляра в соответствии с требованиями профиля ICC. Средства калибровки бывают аппаратнопрограммными и чисто программными.

Не существует идеальной CMS, одинаково пригодной для всех устройств, одинаково работающей на всех платформах и программных средствах. Наиболее удачными можно считать CMS, реализованные на уровне операционной системы. В операционной системе Windows, начиная с версии Windows Vista, используется CMS известная как Windows Color System (WCS) .

Из CMS, являющихся внешними по отношению к операционным системам, наибольшее распространение получили программы фирм, давно работающих в области цветной фотографии, печати, цифровых графических технологий: Agfa Foto Tune, Kodak DayStar Color Match. Что же делать, если CMS для вас еще недоступна, а адекватность восприятия цветов сохранить необходимо? Тут существует две альтернативы: постоянная калибровка периферийного оборудования (сканера, монитора и т. д.) или использование специальных атласов цветов (color sample card). Первый путь довольно дорогой и требующий участия специалистов. Атлас цветов представляет собой совокупность заранее сформированных стандартизированных оттенков цветов, сведенных в упорядоченную таблицу, которые можно использовать напрямую

по мере необходимости. Наиболее распространенными на сегодняшний день являются атласы фирмы Pantone.

3. Графические файловые форматы

Как уже говорилось ранее, при хранении растровых изображений, как правило, приходится иметь дело с файлами большого размера. В этой связи важной задачей является выбор соответствующего формата файла.

Форматов графических файлов существует великое множество и выбор приемлемого отнюдь не является тривиальной задачей. Для облегчения выбора воспользуемся классификациями. **По типу хранимой графической информации:**

▪ растровые (TIFF, GIF, BMP, JPEG); ▪ векторные (AI, CDR, FH7, DXF); ▪ смешанные/универсальные (EPS, PDF).

Следует учитывать, что файлы практически любого векторного формата позволяют хранить в себе и растровую графику. Однако часто это приводит к искажениям в цветопередаче, поэтому если изображение не содержит векторных объектов, то предпочтительнее использовать растровые форматы.

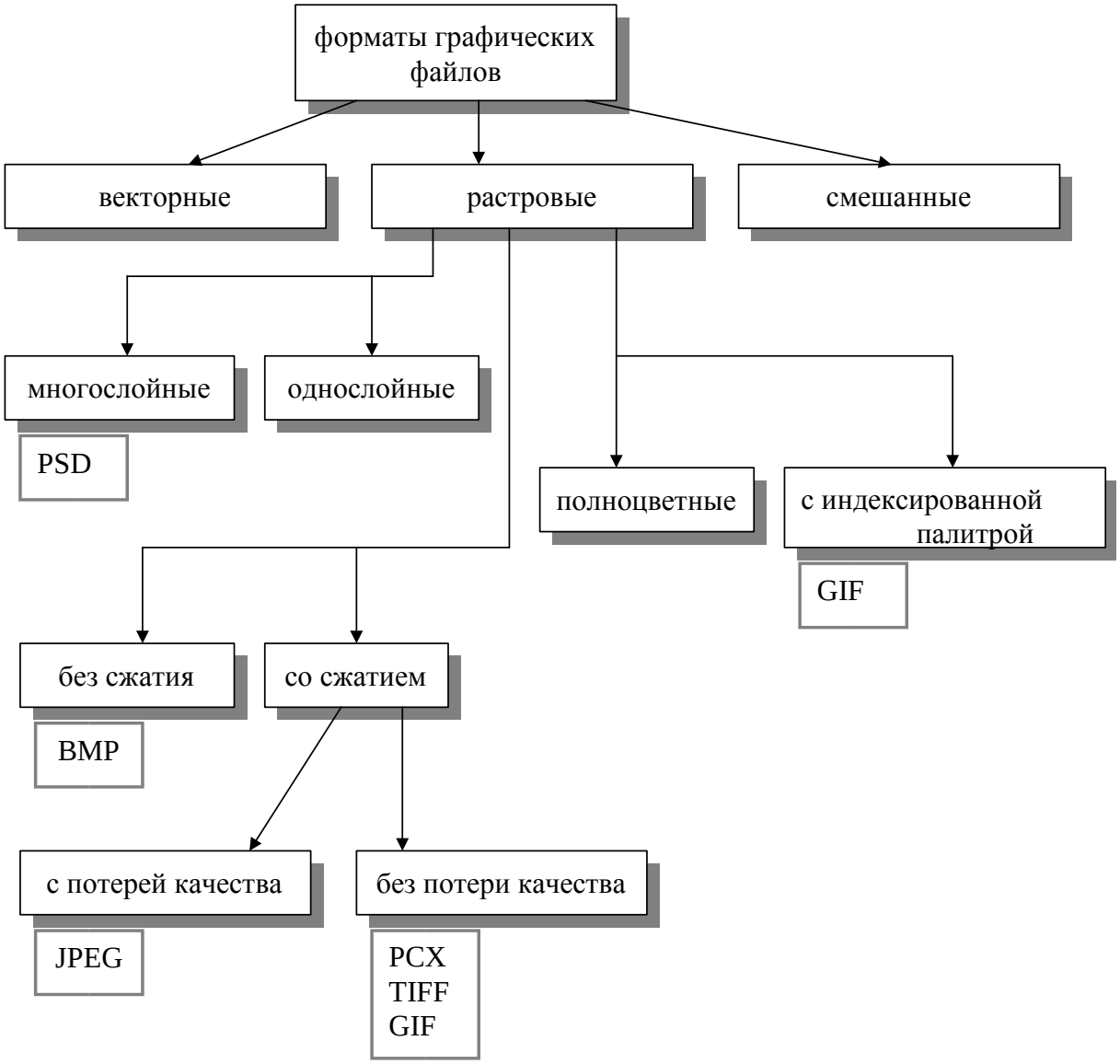


Рис. 3.1. Графические файловые форматы

Скажем несколько слов о наиболее популярных графических форматах.

3.1. BMP

BMP (от англ. BitMap Picture) — формат хранения растровых изображений. BMP был создан компанией Microsoft и широко используется в операционных системах семейства Windows.

Глубина цвета в данном формате может быть 1, 2, 4, 8, 16, 24, 32, 48 бит на пиксель, максимальные размеры изображения 65535×65535 пикселей. Однако, глубина 2 бит официально не поддерживается.

Формат BMP является примером хранения ***полноцветных изображений***. В этом случае, цвета пикселей можно определять, явно задавая несколько параметров цвета. Например, в RGB-модели конечный цвет каждого пикселя определяется тремя слагаемыми для трех основных цветов.

В формате BMP есть поддержка сжатия по ***алгоритму RLE***. Алгоритм RLE или ***алгоритм кодирования повторов*** оперирует сериями данных, то есть последовательностями, в которых один и тот же символ встречается несколько раз подряд. При кодировании строка одинаковых символов, составляющих серию, заменяется строкой, которая содержит сам повторяющийся символ и количество его повторов.

Рассмотрим изображение, состоящее из строки белых и черных пикселей. Опишем эту строку как:

WWWWBWWWWBBBWWWWBWWWW

Здесь B представляет чѐрный пиксель, а W обозначает белый. Если мы применим RLE кодирование к этой строке, то получим следующее:

4W1B4W3B4W1B4W

Файлы формата BMP могут иметь расширения .bmp, .dib и .rle. DIB означает ***аппаратно-независимый растр*** (Device Independent Bitmap). При использовании этого формата программист может получить доступ ко всем элементам структур, описывающих изображение, при помощи обычного указателя. Но эти данные не используются для непосредственного управления экраном, так как они всегда хранятся в системной памяти, а не в специализированной видеопамяти. Формат пикселя в оперативной памяти может отличаться от того формата, который должен заноситься в видеопамять для индикации точки такого же цвета. Например, в DIB-формате может использоваться 24 бита для задания пикселя, а графический адаптер в этот момент может работать в режиме HighColor с цветовой глубиной 16 бит. При этом ярко-красная точка в аппаратно-независимом формате будет задаваться тремя байтами 0x0000ff, а в видеопамяти — словом 0xF800.

DDB означает ***аппаратно-зависимый растр*** (Device Dependent Bitmap, DDB). Этот формат всегда содержит цветовые коды, совпадающие с кодами видеобуфера, но храниться он может как в системной, так и в видеопамяти. В обоих случаях он содержит только коды цвета в том формате, который обеспечит пересылку изображения из ОЗУ в видеопамять при помощи простого копирования.

Таким образом, достоинством формата BMP является простота обращения к отдельным пикселям на изображении, что может быть использовано при написании демонстрационных программ по компьютерной графике. К недостатком нужно отнести сравнительно большие размеры файлов, хранящих изображения в формате BMP, вследствие не совершенства алгоритма RLE.

Кроме поддержки полноцветных изображений формат BMP может обеспечивать хранения изображений с использованием ***индексированной палитры***. Второй подход заключается в том, что в первой части файла, хранящего изображение, хранится ***«палитра»,*** в которой с помощью одной из цветовых моделей кодируются цвета, присутствующие на изображении. А вторая часть, которая непосредственно описывает пиксели изображения, фактически состоит из индексов в палитре.

Формат BMP может использовать режим индексирования цветов при следующих значениях глубины цвета: 1 бит (2 цвета), 2 бита (4 цвета), 4 бита (16 цветов), 8 бит (256 цветов).

Благодаря использованию палитры имеется возможность адаптировать изображение к цветам, присутствующим на изображении. В таком случае изображение ограниченно не заданными цветами, а максимальным количеством одновременно используемых цветов.

Достоинством палитры является возможность существенно сократить размер файла с изображением. Недостатком является возможность потери цветов при ограниченном размере палитры.

3.2. TIFF

TIFF (англ. Tagged Image File Format) — формат хранения растровых графических изображений. Изначально был разработан компанией Aldus в сотрудничестве с Microsoft. TIFF был выбран в качестве основного графического формата операционной системы Mac OS X. В настоящее время авторские права на спецификации формата принадлежат компании Adobe.

Принцип хранения данных в TIFF основан на использовании специальных маркеров (тэгов) в сочетании с битовыми последовательностями кусков растра.

Формат TIFF поддерживает большую глубину цвета: 8, 16, 32 и 64 бит на канал при целочисленном кодировании, а также 32 и 64 бит на канал при представлении значения пикселя числами с плавающей запятой.

Структура формата гибкая и позволяет сохранять изображения в режиме цветов с палитрой, а также в различных цветовых пространствах: бинарном (двуцветном), полутоновом, с индексированной палитрой, RGB, CMYK, YCbCr, CIE Lab. Помимо традиционных цветов CMY формат поддерживает цветоделение с большим числом красок, в частности систему Hexachrome компании Pantone. В систему Hexachrome, известную как CMYKOG, добавлены оранжевые и зеленые краски для лучшего представления цветов при печати.

Помимо прочих достоинств формат TIFF позволяет сохранять растровые изображения с компрессией без потери качества. Этот формат поддерживает сжатие без потери качества по ***алгоритму LZWкомпрессии***.

Алгоритм Лемпеля — Зива — Велча (Lempel-Ziv-Welch, LZW) — это универсальный алгоритм сжатия данных без потерь данных. Алгоритм на удивление прост. Если в двух словах, то LZW-сжатие заменяет строки символов некоторыми кодами. Это делается без какоголибо анализа входного текста. Вместо этого при добавлении каждой новой строки символов просматривается уже существующая таблица строк. Сжатие происходит, когда код заменяет строку символов. Коды, генерируемые LZW-алгоритмом, могут быть любой длины, но они должны содержать больше бит, чем единичный символ. Первые 256 кодов (когда используются 8-битные символы) по умолчанию соответствуют стандартному набору символов. Остальные коды соответствуют обрабатываемым алгоритмом строкам. Простой метод может работать с 12-битными кодами. Значения кодов 0 - 255 соответствуют отдельным байтам, а коды 256 - 4095 соответствуют подстрокам.

Приведем пример LZW кодирования применительно к изображению. Так, если в изображении имеются наборы из розового, оранжевого и зелѐного пикселей, повторяющиеся 50 раз, LZW выявляет это, присваивает данному набору отдельное число (например, 7) и затем сохраняет эти данные 50 раз в виде числа 7. Метод LZW, так же, как и RLE, лучше действует на участках однородных, свободных от шума цветов, он действует гораздо лучше, чем RLE, при сжатии произвольных графических данных, но процесс кодирования и распаковки происходит медленнее.

Кроме LZW-компрессии формат TIFF поддерживает следующие алгоритмы сжатия:

- RLE;
- LZ77;
- ZIP;
- JBIG;
- JPEG;
- CCITT Group 3, CCITT Group 4.

При этом JPEG является просто инкапсуляцией формата JPEG в формат TIFF. Формат TIFF позволяет хранить изображения, сжатые по стандарту JPEG, без потерь данных (JPEG-LS).

3.3. GIF

Первая версия формата GIF (Graphics Interchange Format, «Формат для обмена графической информацией») была разработана в 1987 г. специалистами

компьютерной сети CompuServe. Этот формат сочетает в себе редкий набор достоинств, неоценимых при той роли, которую он играет в WWW. Сам по себе формат содержит уже достаточно хорошо упакованные графические данные.

Формат GIF использует для хранения изображений индексированную палитру, ограничивающую количество цветов 256 значениями. Размер палитры может быть и меньше. Если в изображении используется, скажем, 64 цвета (2^6), то для хранения каждого пикселя будет использовано ровно шесть бит и ни битом больше.

Поскольку GIF использует для сжатия LZW алгоритм, то степень сжатия графической информации сильно зависит от уровня ее повторяемости и предсказуемости, а иногда еще и от ориентации картинки. Так как LZW метод сканирует изображение по строкам, то, к примеру, плавный переход цветов (градиент), направленный сверху вниз, сожмется куда лучше, чем тех же размеров градиент, ориентированный слева направо, а последний – лучше, чем градиент по диагонали.

Изменив порядок следования данных в файле, создатели GIFa заставили картинку рисоваться не только сверху вниз, но и, если можно так выразиться, «с глубины к поверхности», – то есть становиться все четче и детальнее по мере подхода из сети новых данных.

Для этого файл с изображением тасуется при записи так, чтобы сначала шли все строки пикселей с номерами, кратными восьми (первый проход), затем четырем (второй проход), потом двум и, наконец, последний проход – все оставшиеся строки с нечетными номерами. Во время приема и декодирования такого файла каждый следующий проход заполняет «дыры» в предыдущих, постепенно приближая изображение к исходному состоянию. Поэтому такие изображения были названы ***чересстрочными (interlaced)***.

Другой полезной возможностью формата является использование прозрачности и поддержка анимационных изображений. Для создания анимации используется нескольких статичных кадров, а также информация о том, сколько времени каждый кадр должен быть показан на экране.

3.4. PNG

PNG (portable network graphics) - растровый формат хранения графической информации, использующий сжатие без потерь по алгоритму Deflate.

PNG поддерживает три основных типа растровых изображений:

- Полутоновое изображение (с глубиной цвета 16 бит);
- Цветное индексированное изображение (палитра 8 бит для цвета глубиной 24 бит);

- Полноцветное изображение (с глубиной цвета 48 бит).

Формат PNG спроектирован для замены устаревшего и более простого формата GIF, а также, в некоторой степени, для замены значительно более сложного формата TIFF. Алгоритм сжатия Deflate является свободным в отличие от защищенного патентами и соответственно платного LZW, используемого в GIF. В отличии от LZW кодирования, которое позволяет эффективно сжимать горизонтальные одноцветные области, с Deflate сжатием можно забыть про эти ограничения.

В формате GIF один из цветов в палитре может быть объявлен «прозрачным». В этом случае в программах, которые поддерживают прозрачность GIF (например, большинство современных браузеров) сквозь пиксели, окрашенные «прозрачным» цветом будет виден фон. Однако создатели PNG пошли еще дальше, разработав технологию ***альфа-канала***. Альфа-канал позволяет добиться эффекта частичной прозрачности пикселей. Также, в PNG поддерживается гаммакоррекция. ***Гамма-коррекция*** - коррекция функции яркости в зависимости от характеристик устройства вывода. Повышение показателя гамма-коррекции позволяет повысить контрастность, разборчивость тѐмных участков изображения, не делая при этом чрезмерно контрастными или яркими светлые детали снимка.

Кроме этого PNG формат имеет следующие преимущества перед GIF:

- практически неограниченное количество цветов в изображении

(GIF использует в лучшем случае 8-битный цвет);

- двумерная чересстрочная развѐртка;
- возможность расширения формата пользовательскими блоками.

К недостатком формата, можно отнести, что PNG изначально был предназначен лишь для хранения одного изображения в одном файле. Поэтому данный формат не поддерживает анимацию. Для решения этой проблемы создана модификация формата APNG.

3.5. JPEG

JPEG (Joint Photographic Experts Group, по названию организациираз работчика) - один из популярных графических форматов, основанный на алгоритме сжатия JPEG и применяемый для хранения фотоизображений и подобных им изображений.

Алгоритм JPEG разработан группой экспертов из Международной организации по стандартизации (ISO) специально для сжатия полноцветных 24-битовых изображений.

Процесс сжатия по схеме JPEG состоит из нескольких шагов. На первом шаге производится преобразование изображения из цветовой модели RGB в модель YUV, основанной на характеристиках яркости и цветности.

В модели YUV, Y-компонента – светлота (яркость). Компоненты U и V содержат информацию о цвете.

На следующем после преобразования шаге изображение разделяется на квадратные участки размером 8x8 пикселей. После этого над каждым участком производится дискретное косинус-

преобразование (ДКП). При этом выполняется анализ каждого блока, разложение его на составляющие цвета и подсчет частоты появления каждого цвета.

Если говорить научным языком, то JPG использует для сохранения ряды Фурье и при больших степенях сжатия просто отбрасывает члены ряда высшего порядка.

Можно сказать, что JPEG хранит скорость изменения цвета от пикселя к пикселю. Лишнюю с его точки зрения цветовую информацию он отбрасывает, усредняя некоторые значения. Чем выше уровень компрессии, тем больше данных отбрасывается и тем ниже качество

На следующем этапе изображение представляется строками чисел, которые можно сжимать дальше. Поскольку в результате предыдущей обработки строки содержат много нулей, последняя стадия кодирования выполняется по алгоритму Хаффмана, что дает хорошие результаты и позволяет получить файл в 10–500 раз меньше, чем BMP.

В дополнении к стандарту JPEG, ориентированным прежде всего на сжатие с потерями, был разработан стандарт на сжатие без потерь — JPEG-LS (в котором, однако, предусмотрен также режим сжатия с ограниченными потерями).

Алгоритм сжатия, лежащий в основе JPEG-LS, использует адаптивное предсказание значения текущего пикселя по окружению, включающему уже закодированные пиксели.

К недостаткам сжатия по стандарту JPEG следует отнести появление на восстановленных изображениях при высоких степенях сжатия характерных артефактов (заметных искажений изображения).

3.6. PDF

Формат PDF (Portable Document Format) предложен фирмой Adobe как независимый от платформы формат, в котором могут быть сохранены и иллюстрации (векторные и растровые), и текст, причем со множеством шрифтов и гипертекстовых ссылок. Для достижения продекларированной в названии переносимости размер PDF-файла должен быть малым. Для этого используется компрессия (для каждого вида объектов применяется свой способ). Например, растровые изображения записываются в формате JPEG.

Для работы с этим форматом компания Adobe выпустила пакет Acrobat. Бесплатная утилита Acrobat Reader позволяет читать документы и распечатывать их на принтере, но не дает возможности создавать или изменять их. Acrobat Distiller переводит в этот формат PostScript-файлы. Многие программы (Adobe PageMaker, CorelDraw, FreeHand) позволяют экспортировать свои документы в PDF, а некоторые – еще и редактировать графику, записанную в этом формате. Обычно в этом формате хранят документы, предназначенные только для чтения, но не для редактирования. Файл в формате PDF содержит все необходимые шрифты. Это удобно и позволяет не передавать шрифты для вывода (передача шрифтов не вполне законна с точки зрения авторского права).

PostScript

Это язык описания страниц, предназначеный для формирования изображений произвольной сложности и вывода их на печать. Для этого в языке имеется широкий набор графических операторов, используемых в произвольной комбинации. Все графические операторы языка, формирующие изображение, можно разделить на три группы. Это:

- **векторная графика**, позволяющая рисовать прямые линии, дуги, кривые произвольного размера, ориентации, ширины, закрашивать площади любого размера, формы, цвета; цвет для линий или заливок может задаваться в любом из цветовых пространств языка; любой описанный на языке контур может быть границей клипирования изображения; контур клипирования задаèт границы рисуемого изображения;
- **работа с текстом** – для вывода текста произвольного размера в различных гарнитурах, размещая его с произвольной ориентацией в произвольном месте страницы; текст полностью интегрирован с графикой – все текстовые символы трактуются как графические фигуры и могут обрабатываться любым из графических операторов;
- **растровые изображения** позволяют выводить на листе сканированные рисунки или фотографии с масштабированием и

ориентацией, источником растра может быть как текущий файл, содержащий программу на PostScripte, так и внешний; считывание цветовых слоев может вестись как из одного файла, так и из нескольких сепарированных. Изображение, описываемое на языке PostScript, никак не зависит от разрешающей способности выходного устройства и его цветовой глубины (числа цветов). Приближение к конкретным разрешающим возможностям выходного устройства – это процесс, не связанный с описанием изображения на языке PostScript, и выполняется для каждого выходного устройства по-своему. В этом и заключается устройство-независимость языка PostScript. Качество изображения определяется конкретным выходным устройством, его физическими ограничениями.

Формирование изображения на выходном устройстве является двухступенчатым процессом:

1. Приложение генерирует устройство независимое изображение на языке PostScript.
2. Система обработки изображения (интерпретатор) интерпретирует изображение (программу) и приближает его к характеристикам конкретного выходного устройства.

4. Растровые алгоритмы

Большинство графических устройств являются растровыми, представляя изображение в виде прямоугольной матрицы (сетки, целочисленной решетки) пикселей (растра), и большинство графических библиотек содержат внутри себя достаточное количество простейших растровых алгоритмов. На рис. 4.1 приведена система растровых алгоритмов.

Рис. 4.1. Классификация растровых алгоритмов

4.1. Алгоритмы растеризации

Прежде чем перейдем к непосредственному рассмотрению возможности перевода математического описания объекта (линии и пр.) в растровую форму, рассмотрим понятие связности. ***Связность*** – возможность соединения двух пикселей растровой линией, т. е. последовательным набором пикселей. Возникает вопрос, когда пиксели (x_1, y_1) и (x_2, y_2) можно считать соседними. Для этого вводятся два понятия связности:

1. Четырехсвязность: пиксели считаются соседними, если либо их *x*-координаты, либо их *y* – координаты отличаются на единицу:

$$|x_1 - x_2| + |y_1 - y_2| \le 1;$$

2. Восьмисвязность: пиксели считаются соседними, если их *x*координаты и *y*-координаты отличаются не более чем на единицу:

$$|x_1 - x_2| \le 1,\ |y_1 - y_2| \le 1.$$

На рис. 4.2 изображены четырехсвязная и восьмисвязная линии.

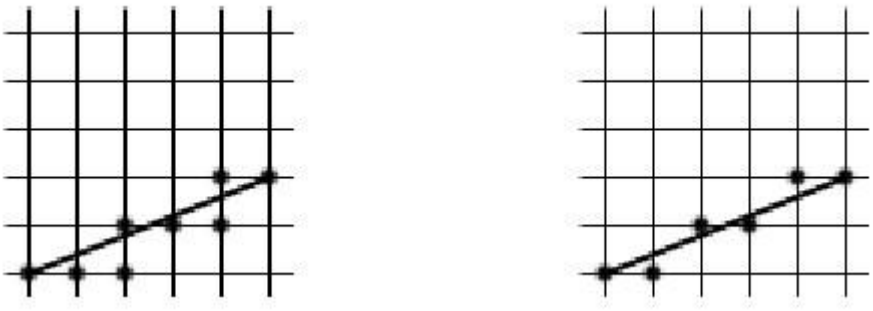


Рис. 4.2. Классификация растровых алгоритмов

При переводе объектов в растровое представление существуют, алгоритмы, как использующие четырехсвязность, так использующие восьмисвязность.

4.1.1. Растровое представление отрезка. Алгоритм Брезенхейма

Рассмотрим задачу построения растрового изображения отрезка, соединяющего точки $A(x_a, y_a)$ и $B(x_b, y_b)$. Для простоты будем считать, что $0 \le y_b - y_a \le x_b - x_a$. Тогда отрезок описывается уравнением: $y = y_a + \frac{y_b - y_a}{x_b - x_a}(x - x_a)$, $x \in [x_a, x_b]$ или $y = kx + b$.

Отсюда получаем простейший алгоритм растрового представления отрезка:

```
    private void MyLine(int xa, int ya, int xb, int yb,
    Color color)     {
        double k = ((double)(yb - ya)) / (xb - xa);
double b = ya - k * xa;     Bitmap bmp;
        bmp = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);     for (int x = xa; x <= xb;
x++){
             bmp.SetPixel(x, (int) (k*x + b), color);}
        Graphics gr = CreateGraphics();
gr.DrawImage(bmp, 0, 0);
        }
```

Вычислений значений функции $y = kx + b$ можно избежать, используя в цикле рекуррентные соотношения, так как при изменении x на 1 значение y меняется на k:

```
    private void MyLineN(int xa, int ya, int xb, int yb,
Color color)
```

```
        {
        double k = ((double)(yb - ya)) / (xb - xa);
        double y = ya;      Bitmap bmp;
        bmp = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);      for (int x = xa; x <= xb;
x++, y+= k){          bmp.SetPixel(x, (int)y, color);}
Graphics gr = CreateGraphics();      gr.DrawImage(bmp, 0,
0);
        }
```

Приведенные простейшие пошаговые алгоритмы построения отрезка имеют ряд недостатков:

1. Выполняют операции над числами с плавающей точкой, а желательно было бы работать с целочисленной арифметикой;
2. На каждом шаге выполняется операция округления, что также снижает быстродействие.

Эти недостатки устранены в следующем алгоритме Брезенхейма.

Как и в предыдущем случае, будем считать, что тангенс угла наклона отрезка принимает значение в диапазоне от 0 до 1. Рассмотрим *i*-й шаг алгоритма (рис. 4.3). На этом этапе пиксель P_{i-1} уже найден как ближайший к реальному отрезку. Требуется определить, какой из пикселей (T_i или S_i) будет установлен следующим.

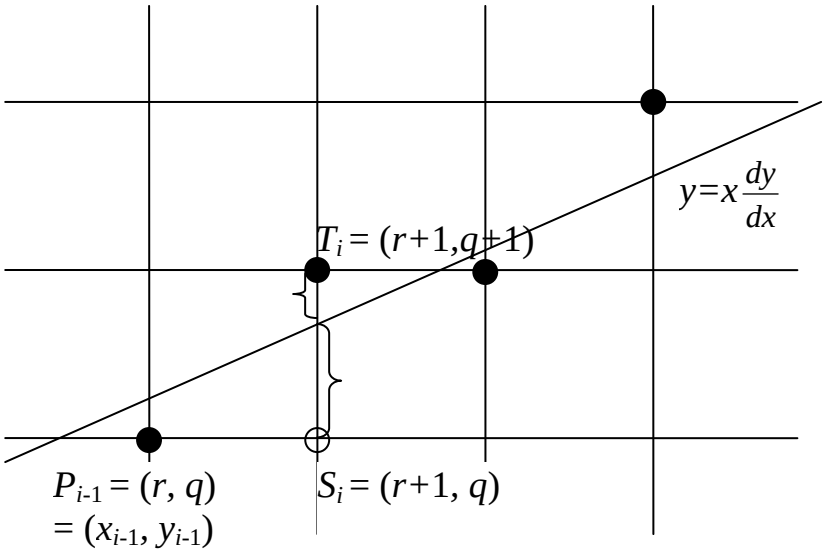


Рис. 4.3. і-й шаг алгоритма Брезенхейма

В алгоритме используется управляющая переменная d_i, которая на каждом шаге пропорциональна разности между S и T. Если $S < T$, то S_i ближе к отрезку, иначе выбирается T_i.

Пусть изображаемый отрезок проходит из точки (x_1, y_1) в точку (x_2, y_2). Исходя из начальных условий, точка (x_1, y_1) ближе к началу координат. Тогда перенесем оба конца отрезка с помощью преобразования $T(-x_1, -y_1)$, так чтобы первый конец отрезка совпал с началом координат. Начальной точкой отрезка стала точка (0, 0), конечной точкой – (dx, dy), где $dx = x_2 - x_1$, $dy = y_2 - y_1$ (рис. 4.4).

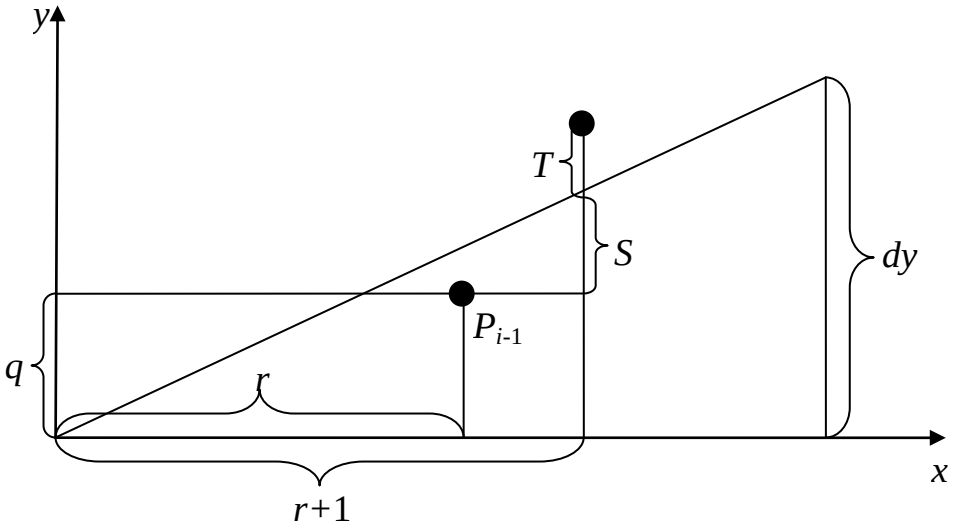


Рис. 4.4. Вид отрезка после переноса в начало координат Уравнение прямой в этом случае будет иметь вид:

$$y = x \frac{dy}{dx}.$$

Обозначим координаты точки P_{i-1} после переноса через (r, q). Тогда $S_i = (r+1, q)$ и $T_i = (r+1, q+1)$.

Из подобия треугольников на рис. 4.4 можно записать, что

$$\frac{dy}{dx} = \frac{S - q}{r + 1}.$$

Выразим S: $S = \frac{dy}{dx}(r + 1) - q$.

T можно представить как $T = 1 - S$. Используем предыдущую формулу

$T = 1 - S = 1 - \frac{dy}{dx}(r + 1) - q$. Найдем разницу $S - T$: $S - T = \frac{dy}{dx}(r + 1) - q - 1 + \frac{dy}{dx}(r + 1) - q = 2\frac{dy}{dx}(r + 1) - 2q - 1$.

Помножим левую и правую часть на dx: $dx\ (S - T) = 2\ dy\ (r + 1) - 2\ q\ dx - dx = 2\ (r\ dy - q\ dx) + 2\ dy - dx$.

Величина dx положительная, поэтому неравенство $dx\ (S - T) < 0$ можно использовать в качестве проверки при выборе S_i. Обозначим $d_i = dx\ (S - T)$, тогда

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

Поскольку $r = x_{i-1}$ и $q = y_{i-1}$, то

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

Прибавляя 1 к каждому индексу найдем d_{i+1}:

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

Вычитая d_i из d_{i+1} получим

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

Известно, что $x_i - x_{i-1} = 1$, тогда

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

Отсюда выразим d_{i+1}:

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

Таким образом, получили итеративную формулу вычисления управляющего коэффициента d_{i+1} по предыдущему значению d_i. С помощью управляющего коэффициента выбирается следующий пиксель – S_i или T_i.

Если $d_i \geq 0$, тогда выбирается T_i и $y_i = y_{i-1} + 1$, $d_{i+1} = d_i + 2\ (dy - dx)$.

Если $d_i < 0$, тогда выбирается S_i и $y_i = y_{i-1}$ и $d_{i+1} = d_i + 2\ dy$.

Начальные значения d_1 с учетом того, что $(x_0, y_0) = (0, 0)$, $d_1 = 2\ dy - dx$.

Преимуществом алгоритма является то, что для работы алгоритма требуются минимальные арифметические возможности: сложение, вычитание и сдвиг влево для умножения на 2.

Реализация этого алгоритма выглядит следующим образом:

```
    private void BLine(int x1, int y1, int x2, int y2,
    Color color)     {
        int dx, dy, inc1, inc2, d, x, y, Xend;     dx =
Math.Abs(x2 - x1);     dy = Math.Abs(y2 - y1);     d =
(dy << 1) - dx;     inc1 = dy << 1;      inc2 = (dy - dx)
<< 1;      if (x1 > x2)        {        x = x2;        y
= y2;        Xend = x1;        }     else        {
x = x1;         y = y1;        Xend = x2;
           };     Bitmap bmp;
        bmp = new Bitmap(this.ClientSize.Width,
this.ClientSize.Height);     bmp.SetPixel(x, y, color);
        while (x < Xend)
          {       x++;
          if (d < 0) d = d + inc1;       else {y++; d = d
+ inc2;};       bmp.SetPixel(x, y, color);
          }
        Graphics gr = CreateGraphics();
gr.DrawImage(bmp, 0, 0);
        }
```

Если *dy* > *d*x, то необходимо будет использовать этот же алгоритм, но пошагово увеличивая *y* и на каждом шаге вычислять *x*.

4.1.2. Растровая развёртка окружности

Существует несколько очень простых, но не эффективных способов преобразования окружностей в растровую форму. Например, рассмотрим для простоты окружность с центром в начале координат. Ее уравнение записывается как $x^2 + y^2 = R^2$. Решая это уравнение относительно *y*, получим

$$y = \pm\sqrt{R^2 - x^2}.$$

Чтобы изобразить четвертую часть окружности, будем изменять *x* с единичным шагом от 0 до *R* и на каждом шаге вычислять *y*. Вторым простым методом растровой развертки окружности является использование вычислений *x* и *y* по формулам $x = R\cos\alpha$, $y = R\sin\alpha$ при пошаговом изменении угла α от 0° до 90°.

Для упрощения алгоритма растровой развѐртки стандартной окружности можно воспользоваться еѐ симметрией относительно координатных осей и прямых $y = \pm x$; в случае, когда центр окружности не совпадает с началом

координат, эти прямые необходимо сдвинуть параллельно так, чтобы они прошли через центр окружности. Тем самым достаточно построить растровое представление для 1/8 части окружности, а все оставшиеся точки получить симметрией (см. рис. 4.5).

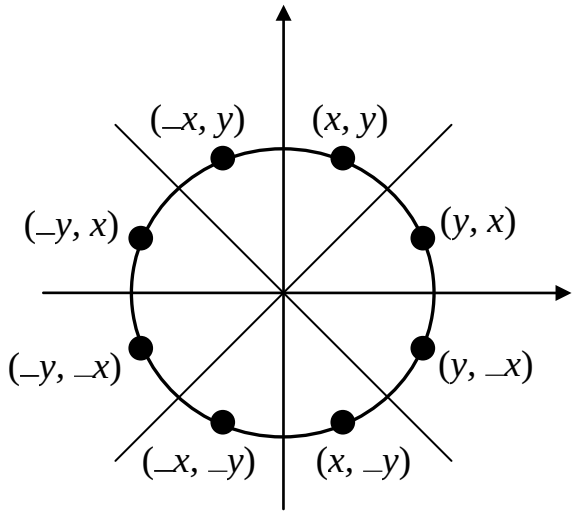


Рис. 4.5. Восьмисторонняя симметрия

Рассмотрим участок окружности из второго октанта $x \in [0, R/\sqrt{2}]$. Далее опишем алгоритм Брезенхейма для этого участка окружности.

На каждом шаге алгоритм выбирает точку P_i (x_i, y_i), которая является ближайшей к истинной окружности. Идея алгоритма заключается в выборе ближайшей точки при помощи управляющих переменных, значения которых можно вычислить в пошаговом режиме с использованием небольшого числа сложений, вычитаний и сдвигов.

Рассмотрим небольшой участок сетки пикселей, а также возможные способы (от A до E) прохождения истинной окружности через сетку (рис. 4.6).

Предположим, что точка P_{i-1} была выбрана как ближайшая к окружности при $x = x_{i-1}$. Теперь найдем, какая из точек (S_i или T_i) расположена ближе к окружности при $x = x_{i-1} + 1$.

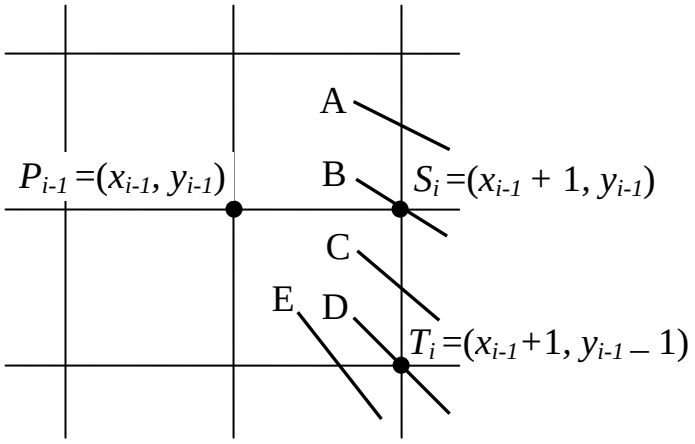


Рис. 4.6. Варианты прохождения окружности через растровую сетку

Заметим, что ошибка при выборе точки P_i (x_i, y_i) была равна

$$D(P_i) = (x_i^2 + y_i^2) - R^2.$$

Запишем выражение для ошибок, получаемых при выборе точки S_i или T_i:

$$D(S_i) = [(x_{i-1} + 1)^2 + (y_{i-1})^2] - R^2;$$
$$D(T_i) = [(x_{i-1} + 1)^2 + (y_{i-1} - 1)^2] - R^2.$$

Если $| D(S_i) | \geq | D(T_i) |$, то T_i ближе к реальной окружности, иначе выбирается S_i.

Введем $d_i = | D(S_i) | - | D(T_i) |$.

T_i будет выбираться при $d_i \geq 0$, в противном случае будет устанавливаться S_i.

Опуская алгебраические преобразования, запишем d_i и d_{i+1} для разных вариантов выбора точки S_i или T_i.

$$D_1 = 3 - 2R.$$

Если выбирается S_i (когда $d_i < 0$), то $d_{i+1} = d_i + 4 x_{i-1} + 6$.

Если выбирается T_i (когда $d_i \geq 0$), то $d_{i+1} = d_i + 4 (x_{i-1} - y_{i-1}) + 10$.

Существует модификация алгоритма Брезенхейма для эллипса.

4.1.3. Кривые Безье

Кривые Безье были разработаны в 60-х годах XX века независимо друг от друга Пьером Безье из автомобилестроительной компании «Рено» и Полем де Касталье (Кастельжо) из компании «Ситроен», где применялись для проектирования кузовов автомобилей. Благодаря простоте задания и манипуляции, кривые Безье нашли широкое применение в компьютерной графике для представления гладких линий.

Построение кривых Безье является решением задачи аппроксимации, то есть построения кривой по узловым точкам. При этом кривая не обязательно проходит через данные точки. В случае кривых Безье, кривая проходит через первую и последнюю точку, а промежуточные точки играют роль рычагов, задающих форму кривой. Кривая Безье это параметрическая кривая, задаваемая выражением:

$$B(t) = \sum_{i=0}^{n} P_i b_{i,n}(t), \quad 0 < t < 1$$

где P_i – функция компонент векторов опорных вершин, а $b_{i,n}(t)$ – базисные функции кривой Безье, называемые также полиномами Бернштейна.

$$b_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}$$

где n – степень полинома, i – порядковый номер опорной вершины. Наглядный метод построения этих кривых, предложенный де Кастелье, основан на разбиении отрезков, соединяющих исходные точки в отношении t (значение параметра), а затем в рекурсивном повторении этого процесса для полученных отрезков: $P_{i,j}(t) = (1-t) P_{i,j-1}(t) + t P_{i+1,j-1}(t)$

Нижний индекс – номер опорной точки, верхний индекс – уровень разбиения. Уравнение n-ого порядка задается $P(t) = P_0^n(t)$

Вид кривой зависит от количества опорных точек. Наиболее часто используются линейные, квадратные, кубические и кривые четвертого порядка.

Линейные кривые

При $n = 1$ кривая представляет собой отрезок прямой линии, опорные точки P_0 и P_1 определяют его начало и конец. Кривая задаётся уравнением:

$$B(t) = (1-t) P_0 + t P_1 \qquad t \in [0, 1].$$

Квадратные кривые

Квадратная кривая Безье (n = 2) задаётся 3-я опорными точками: P_0, P_1 и P_2. Кривая задаётся уравнением:

$$B(t) = (1-t)^2 P_0 + 2t(1-t) P_1 + t^2 P_2 \qquad t \in [0, 1].$$

Данное уравнение можно получить с помощью метода де Касталье следующим образом. Обозначим опорные точки, как

$P\ i_i$, 0,2, начало кривой положим в точке P_0 ($t = 0$) , а конец - в точке P_2 ($t = 1$); для каждого t Є [0, 1] найдем точку $P\ t_0^2(\ )$:

$P\ t_0^1(\ )\ (1\ t\ P)_0\ tP_1$,

$P\ t_1^1(\ )\ (1\ t\ P)_1\ tP_2$,

$P\ t_0^2(\ )\ (1\ t\ P\ t)_0^1(\ )\ tP\ t_1^1(\ )\ (1\ t)^2 P_0\ 2\ (1t\ t\ P)_1\ t\ P^2{}_2$, таким образом, получим кривую второго порядка.

Точки $P\ t_0^1(\ )$ и $P\ t_1^1(\ )$ лежат на кривых первого порядка (отрезках) соединяющие опорные точки P_0 с P_1 и P_1 с P_2 соответственно. Точки, образующие кривую Безье, $P\ t_0^2(\ )$ лежат на отрезке, соединяющем $P\ t_0^1(\ )$ с $P\ t_1^1(\ )$ пропорционально изменению параметра t (рис. 4.7).

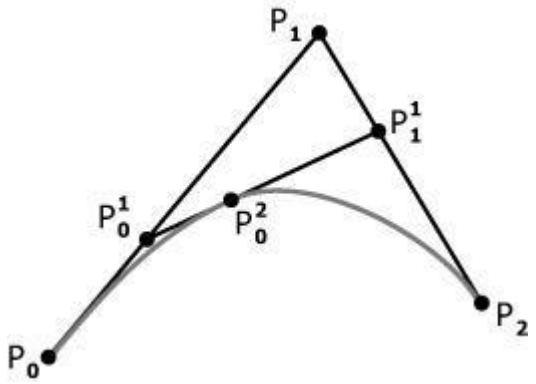


Рис. 4.7. Кривая Безье с тремя опорными точками

Кубические кривые

В параметрической форме кубическая кривая Безье ($n = 3$) описывается следующим уравнением:

$$B(t) = (1–t)^3\ P_0 + 3t(1\text{-}t)^2\ P_1 + 3t^2(1\text{-}t)\ P_2 + t^3\ P_3 \qquad t \in [0, 1].$$

Для получения этой формулы используем аналогичным способом метод де Касталье.

—

Обозначим опорные точки, как $P\ i_i$, 0,3, начало кривой положим в точке P_0 ($t = 0$) , а конец - в точке P_3 ($t = 1$); для каждого t Є [0, 1] найдем точку $P\ t_0^3(\ )$:

$P\ t_0^1(\ )\ \square\ (1\square t\ P)_0\ \square tP_1,$

$P\ t_1^1(\ )\ \square\ (1\square t\ P)_1\ \square tP_2,$

$P\ t_2^1(\ )\ (1\square\ \square t\ P)_2\ \square tP_3,$

$P\ t_0^2(\ )\ \square\ (1\square t\ P\ t)_0^1(\ )\square tP\ t_1^1(\ )\ \square\ (1\square t)^2P_0\ \square 2\ (1t\qquad \square t\ P)_1\ \square t\ P^2{}_2,$

$P\ t_1^2(\ )\ (1\square\ \square t\ P\ t)_1^1(\ )\square tP\ t_2^1(\ )\ (1\square\ \square t)^2P_1\ \square 2\ (1t\ \square t\ P)_2\ \square t\ P^2{}_3,$

$P\ t_0^3(\ )\ \square\ \square(1\ t\ P\ t)_0^2(\ )\square tP\ t_1^2(\ )\ \square$

$\square\ \square(1\ t)^{2\ 1}P\ t_0(\ )\square 2\ (1t\ \square t\ P\ t)_1^1(\ )\square t\ P\ t^{2\ 1}{}_2(\ )\ \square,\ \square\ \square(1\ t\ P)^3{}_0\ \square 3\ (1t\ \square t)^2P_1$

$\square 3t\ P^2{}_2\ \square t\ P^3{}_3.$

таким образом, получим кривую третьего порядка (рис. 4.8).

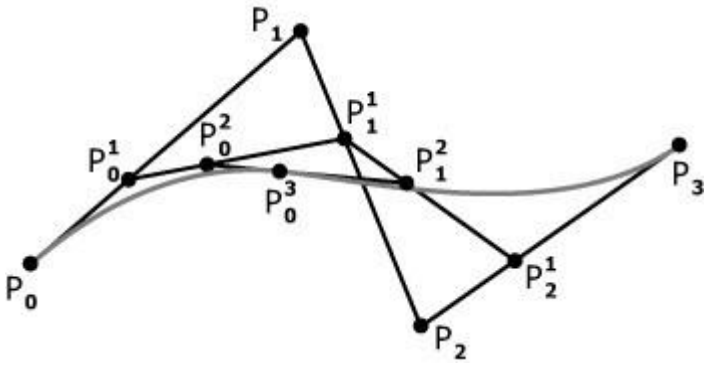


Рис. 4.8. Кривая Безье с четырьмя опорными точками

Четыре опорные точки P_0, P_1, P_2 и P_3, заданные в двух или трехмерном пространстве определяют форму кривой.

Линия берѐт начало из точки P_0 направляясь к P_1 и заканчивается в точке P_3 подходя к ней со стороны P_2. То есть кривая не проходит через точки P_1 и P_2, они используются для указания еѐ направления. Длина отрезка между P_0 и P_1 определяет, как скоро кривая повернѐт к P_3.

В матричной форме кубическая кривая Безье записывается следующим образом:

$\square\ \square P_0{}_3\qquad t_2\qquad t\qquad 1\square\square M_B\square\ \square\square\ \square\square$

$\square PP_{12},$

$$B\,t(\,)\;\square\square\square t$$

$$\square\;\square$$

$$\square\;\square P_3$$

где M_B называется базисной матрицей Безье:

$$\square 1 \qquad 3 \qquad \square$$

$$\square\; 3\,M6$$

$$_{B}\square\square \qquad\qquad 0\;0\;\square$$

$$0$$

$$\square 3 \qquad\qquad .$$

$$\square \qquad\qquad \square$$

$$\square$$

$$1 \qquad\qquad \square\square$$

Сплайн Безье

Кривые Безье обладают рядом свойств:

1. непрерывность заполнения сегмента между начальной и конечной точками;
2. инвариантность относительно аффинных преобразований. Как следствие поворот вокруг точки, масштабирование и изменение пропорций кривой Безье не нарушает ее

стабильности;

3. кривая Безье принадлежит выпуклой оболочке опорных точек, т.е. кривая всегда располагается внутри фигуры, образованной линиями, соединяющими контрольные точки;
4. если все опорные точки лежат на одной прямой, то кривая Безье вырождается в отрезок, соединяющий эти точки;
5. кривая Безье симметрична, то есть обмен местами между начальной и конечной точками (изменение направления траектории) не влияет на форму кривой;
6. степень многочлена, представляющего кривую в аналитическом виде, на единицу меньше числа опорных точек.

Одним из главных свойств кубических кривых Безье является возможность составления из них сплайнов.

Понятие сплайна пришло из машиностроения, где сплайном называли гибкую линейку, закрепив которую в нужных местах, добивались плавной формы кривой, и затем чертили ее по этой линейке.

В математике под сплайном обычно понимают кусочно-заданную функцию, совпадающую с функциями более простой природы на каждом элементе разбиения своей области определения.

В случае с кривыми Безье сплайн обычно составляется из нескольких кубических кривых. Если точка P_n является стыковочной точкой, для соблюдения условия гладкости кривой в ней, необходимо, что бы выполнялось условие коллинеарности с соседними узловыми точками (рис. 4.9).

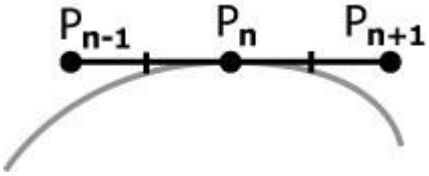


Рис. 4.9. Соблюдение условия гладкости на стыке кривых Безье

Однако даже такие достаточно развитые средства аппроксимации кривыми Безье не позволяют построить окружность, описываемую параметрически как:

$$x = \cos t \quad y = \sin t ,$$

так как *sin* и *cos* для достаточно хорошего приближения требуют многочленов высокой степени.

4.1.4. Закраска области, заданной цветом границы

Рассмотрим область, ограниченную набором пикселей заданного цвета и точку (x, y), лежащую внутри этой области.

Задача заполнения области заданным цветом в случае, когда эта область не является выпуклой, может оказаться довольно сложной. Приведем простейший рекурсивный алгоритм заполнения области:

```
    private void MyPixelFill(int x, int y, Color
border_color, Color c_color)
    {
    Color c = bmp.GetPixel(x, y);
    if ((c.ToArgb() != border_color.ToArgb()) &&
```

```
        (c.ToArgb() != c_color.ToArgb()))
        {
        bmp.SetPixel(x, y, c_color);
        MyPixelFill(x - 1, y, border_color, c_color);
        MyPixelFill(x + 1, y, border_color, c_color);
        MyPixelFill(x, y - 1, border_color, c_color);
        MyPixelFill(x, y + 1, border_color, c_color);
        }
    }
```

Заметим, что корректное использование метода `GetPixel` возможно, если предварительно объекты типа `Graphics` и `Bitmap` связаны и инициализируются следующим образом: `bmp = new Bitmap(this.ClientSize.Width, this.ClientSize.Height); gr = Graphics.FromImage(bmp);`

Этот алгоритм является слишком неэффективным, так как для всякого уже отрисованного пикселя функция вызывается ещѐ 4 раза и, кроме того, этот алгоритм требует слишком большого объѐма стека изза большой глубины рекурсии. Поэтому для решения задачи закраски области предпочтительнее алгоритмы, способные обрабатывать сразу целые группы пикселей, т. е. использовать их «связность» или «когерентность». Если данный пиксель принадлежит области, то, скорее всего, его ближайшие соседи также принадлежат данной области.

Группой таких пикселей обычно выступает полоса, определяемая правым пикселем. Для хранения правых определяющих пикселей используется стек. Словесно опишем улучшенный алгоритм, использующий когерентность пикселей.

Сначала заполняется горизонтальная полоса пикселей, содержащих начальную точку. Затем, чтобы найти самый правый пиксель каждой строки, справа налево проверяется строка, предыдущая по отношению к только что заполненной полосе. Адреса найденных пикселей заносятся в стек. То же самое выполняется и для строки, следующей и за последней заполненной полосой. Когда строка обработана таким способом, в качестве новой начальной точки используется пиксель, адрес которого берется из стека. Для него повторяется вся описанная процедура. Алгоритм заканчивает свою работу, если стек пуст.

4.1.5. Заполнение многоугольника

Часто возникает задача заполнения многоугольников, заданных набором вершин.

Задача заполнения многоугольников решается в два этапа:

1) сначала проводится операция отсечения многоугольника;

2) затем производится заполнение полученных многоугольников.

Этап отсечения необходим для определения реальных областей многоугольника, которые будут выведены на экран. Это необходимо, если многоугольник больше или выходит за пределы экрана или окна вывода.

Отсечение многоугольников

Отсечение многоугольников чаше всего проводится для отбрасывания частей многоугольника, выходящих за границу прямоугольной области, которая определяет экран или область окна. Однако отсечение может проводиться относительно и другого многоугольника. При этом порождается новый многоугольник или несколько новых многоугольников.

Рассмотрим алгоритм Сазерленда-Ходгмана (Sutherland-Hodgman). В алгоритме используется стратегия «разделяй и властвуй», которая позволяет решение общей задачи свести к решению ряда простых и похожих подзадач. Примером такой подзадачи является отсечение многоугольника относительно одной отсекающей границы. Последовательное решение четырех таких задач позволяет провести отсечение относительно прямоугольной области (рис. 4.10).

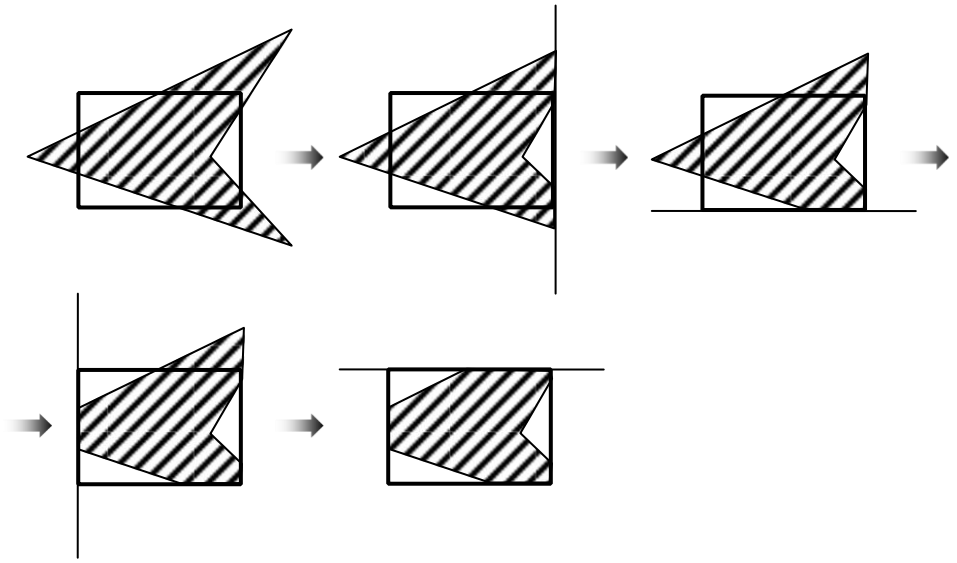


Рис. 4.10. Последовательное отсечение многоугольника

На вход алгоритма поступает последовательность вершин многоугольника V_1, V_2, ..., V_n. Ребра многоугольника проходят от V_i к V_{i+1} от V_n к V_1. С помощью алгоритма производится отсечение относительно ребра и выводится другая последовательность вершин, описывающая усеченный многоугольник.

Алгоритм «обходит» вокруг многоугольника от V_n к V_1 и обратно к V_n, проверяя на каждом шаге соотношение между последовательными вершинами и отсекающей границей. Необходимо проанализировать четыре случая (рис. 4.11).

Рассмотрим изображенное на рис. 4.11 ребро многоугольника, соединяющее вершину *s* с вершиной *p*. В первом случае ребро полностью лежит внутри отсекающей границы, к выходному списку добавляется вершина *p*. Во втором случае в качестве вершин выводится точка пересечения *i*, поскольку ребро пересекает границу, а начальная точка была выведена при анализе первого случая. В третьем случае обе вершины находятся за пределами границы и ни одна из них не выводится. В четвертом случае к выходному списку добавляется и точка пересечения *i*, и вершина *p*.

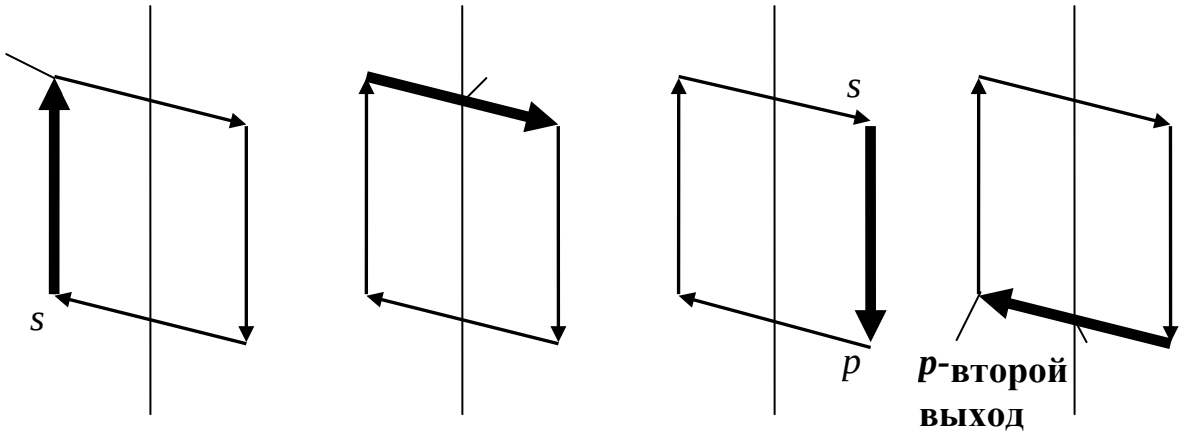


Рис. 4.11. Случаи, возникающие при отсечении многоугольников

При отсечении многоугольника описанным алгоритмом возникает проблема, связанная с тем, что появляются ребра, частично совпадающие с границей окон. Лишние ребра можно устранить, введя дополнительную обработку или воспользовавшись более общим и более сложным алгоритмом Вейлера – Азертона.

Заполнение многоугольников

Рассмотрим, каким образом можно заполнить многоугольник, задаваемый замкнутой ломаной линией без самопересечений.

Простейший способ закраски многоугольника состоит в проверке принадлежности каждой точки этому многоугольнику. Более эффективные алгоритмы используют тот факт, что соседние пиксели, вероятно, имеют одинаковые характеристики (кроме пикселей граничных ребер). Это свойство называется ***пространственной когерентностью.***

В случае с многоугольником когерентность пикселей определяется вдоль сканирующей строки. Сканирующие строки обычно изменяются от «верха» многоугольника до его «низа». Характеристики пикселей изменяются только там, где ребро многоугольника пересекает строку. Эти пересечения делят сканирующую строку на области закрашенных и не закрашенных пикселей.

Рассмотрим, какие случаи могут возникнуть при делении многоугольника на области сканирующей строкой.

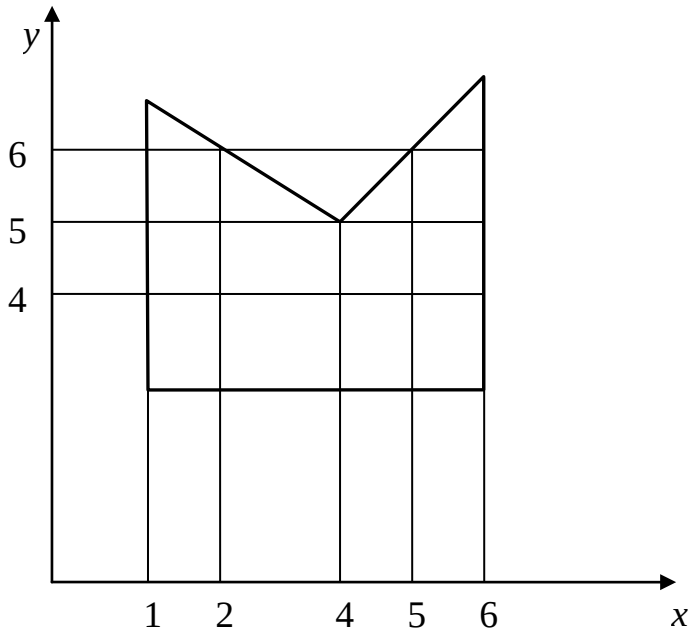


Рис. 4.12. Прохождение сканирующих строк по многоугольнику

1. Простой случай. Например, на рис. 4.12 сканирующая строка $y = 4$ пересекает многоугольник при $x = 1$ и $x = 6$. Получается три области: $x < 1$; $1 \le x \le 6$; $x > 8$. Сканирующая строка $y = 6$ пересекает многоугольник при $x = 1$; $x = 2$; $x = 5$; $x = 6$. Получается пять областей: $x < 1$; $1 \le x \le 2$; $2 < x < 5$; $5 \le x \le 6$; $x > 6$. В этом случае x сортируется в порядке возрастания. Далее список иксов рассматривается попарно. Между парами точек пересечения закрашиваются все пиксели. Для $y = 4$ закрашиваются пиксели в интервале (1, 6), для $y = 6$ закрашиваются пиксели в интервалах (1, 2) и (5, 6).

2. Сканирующая строка проходит через вершину (рис. 4.13). Например, по сканирующей строке $y = 3$ упорядоченный список x получится как (2, 2, 4). Вершина многоугольника была учтена дважды, и поэтому закрашиваемый интервал получается неверным: (2, 2). Следовательно, при пересечении вершины сканирующей строкой она должна учитываться единожды. И список по x в приведенном примере будет (2, 4).

3. Сканирующая строка проходит через локальный минимум или максимум (см. рис. 4.12 при $y = 5$). В этом случае учитываются все пересечения вершин сканирующей строкой. На рис. 4.12 при $y = 5$ формируется список x (1, 4, 4, 6). Закрашиваемые интервалы (1, 4) и (4, 6). Условие нахождения локального минимума или максимума определяется при рассмотрении концевых вершин для ребер, соединенных в вершине. Если у обоих концов координаты y больше, чем y вершины пересечения, то вершина – локальный минимум. Если меньше, то вершина пересечения – локальный максимум.

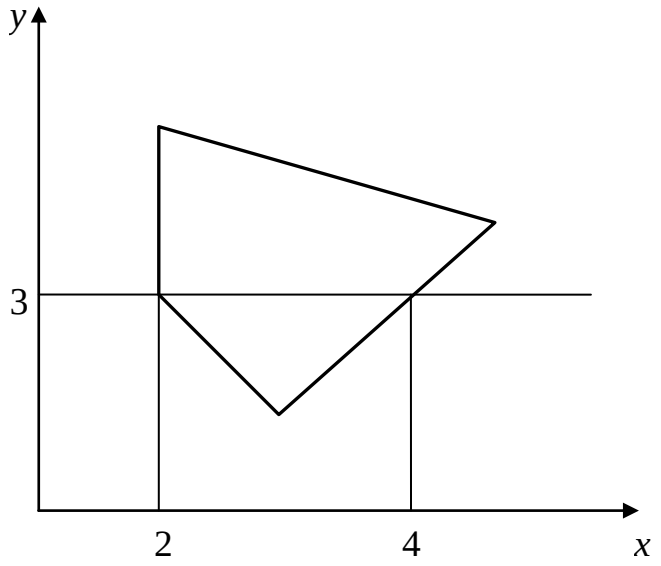


Рис. 4.13. Прохождение сканирующий строки через вершину

Для ускорения работы алгоритма используется список активных ребер (САР). Этот список содержит те ребра многоугольника, которые пересекают

сканирующую строку. При пересечении очередной сканирующей строки вершины многоугольника, из САР удаляются ребра, которые находятся выше, и добавляются концы, которые пересекает сканирующая строка. При работе алгоритма находятся пересечения сканирующей строки только с ребрами из САР.

4.2. Методы устранения ступенчатости

Основная причина появления лестничного эффекта заключается в том, что отрезки, ребра многоугольника, цветовые границы и пр. имеют непрерывную природу, тогда как растровые устройства дискретны.

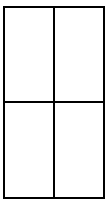
Лестничный эффект проявляется:

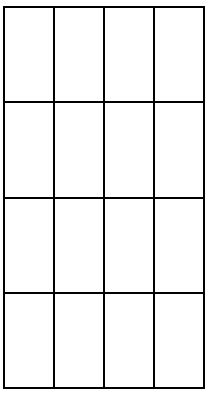
1) при визуализации мелких деталей; 2) при прорисовке ребер и границ; 3) при анимации мелких деталей.

Опишем основные методы устранения ступенчатости.

4.2.1. Метод увеличения частоты выборки

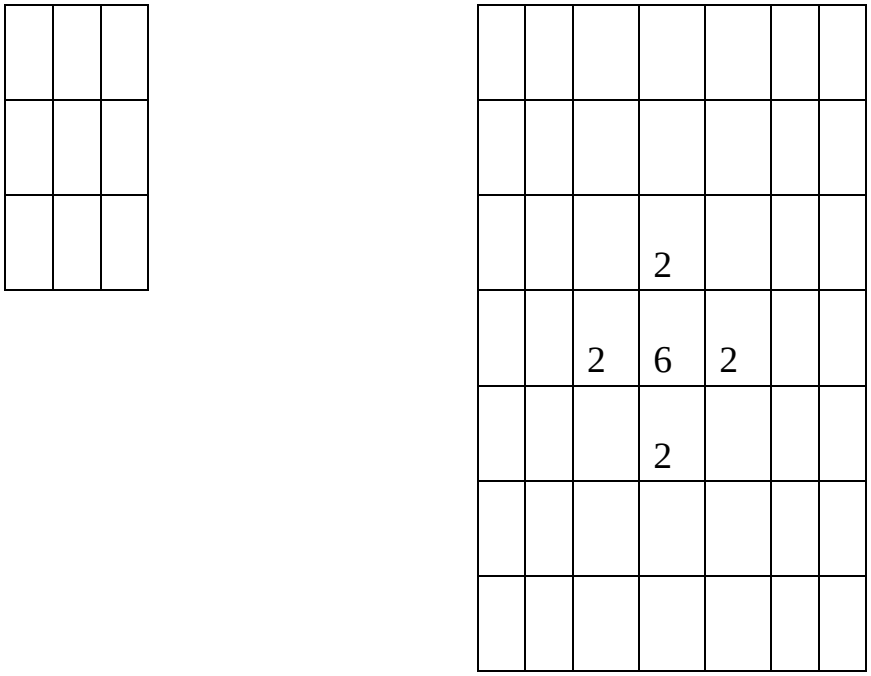
Первый метод устранения ступенчатости связан с увеличением частоты выборки. Увеличение частоты выборки достигается с помощью увеличения разрешения растра. Таким образом учитываются более мелкие детали. Каждый пиксель делится на подпиксели в процессе формирования растра более высокого разрешения. Для получения атрибутов дисплейного пикселя определяются атрибуты в центре каждого подпикселя, которые потом усредняются. Подпиксели в этом случае распределяются равномерно и их атрибуты учитываются одинаково.





Увеличение разрешения в два Увеличение разрешения в раза четыре раза

В некоторой степени можно получить лучшие результаты, если рассматривать больше подпикселей и учитывать их влияние с помощью весов при определении атрибутов.



Числа обозначают относительные веса каждого подпикселя.

4.2.2. Метод, основанный на использовании полутонов

В этом эвристическом методе интенсивность пикселя на ребре устанавливается пропорционально площади части пикселя, находящегося внутри многоугольника.

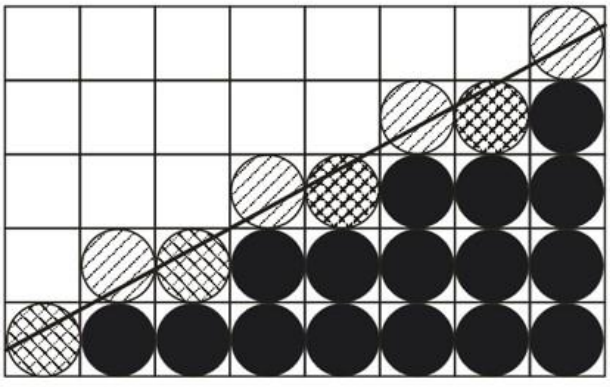


Рис. 4.14. Закраска ребра

На рис. 4.14 приводится многоугольник, ребро которого сгенерировано основным алгоритмом Брезенхейма. Пиксели в этом ребре и внутренние пиксели многоугольника полностью закрашены. Пиксели, находящиеся выше

ребра, закрашиваются с различной интенсивностью, которая пропорциональна площади пикселя, попадающего внутрь многоугольника.

4.3. Методы обработки изображений

Часто в компьютерной графике возникает задача обработки изображений. Обработка, как правило, заключается в наложении на изображение каких-либо эффектов – это размытие, резкость, деформация, шум и т. д., а также в регулировке уровня яркости и контраста.

4.3.1. Яркость и контраст

Яркость и контраст являются субъективными характеристиками изображения, воспринимаемыми человеком.

Яркость (brightness) представляет собой характеристику, определяющую то, на сколько сильно цвета пикселей отличаются от чѐрного цвета. Например, если оцифрованная фотография сделана в солнечную погоду, то ее яркость будет значительной. С другой стороны, если фотография сделана вечером или ночью, то еѐ яркость будет невелика.

В цветовом пространстве RGB , яркость можно рассматривать как среднее арифметическое μ красного, зеленого и синего цвета координаты. В цветовых моделях как HSB или HSV значение яркости указано на прямую.

Контраст (contrast) представляет собой характеристику того, насколько большой разброс имеют цвета пикселей изображения. Чем больший разброс имеют значения цветов пикселей, тем больший контраст имеет изображение.

По аналогии с терминами теории вероятностей можно отметить, что яркость представляет собой как бы математическое ожидание значений выборки, а контраст – дисперсию значений выборки.

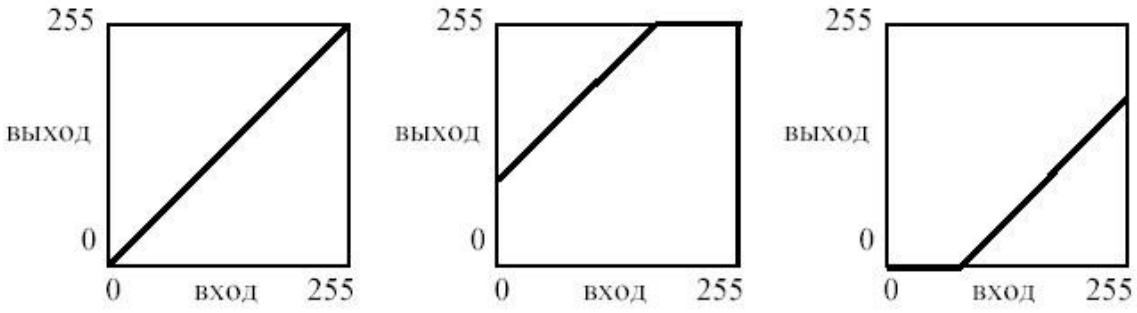
Яркость и контраст могут рассматриваться не только для всего изображения, но и для отдельных фрагментов. Таким образом, возникают понятия локальной яркости и локального контраста.

Часто требуется изменить яркость или контраст изображения. Рассмотрим функцию, областью определения и значений которой являются значения цветовых компонент в модели RGB. Аргументом функции является цвет пикселя исходного изображения. Значение функции представляет собой цвет пикселя обработанного изображения. Для изменения яркости/контраста функция применяется для каждого пикселя изображения.

Для нормализации выходных значений функции (они должны принадлежать отрезку [0, 1], как для каждого компонента модели RGB) используется так называемая арифметика с насыщением. В арифметике с насыщением при возникновении переполнений или заѐмов фиксируется

наибольшее представимое или наименьшее представимое значения соответственно. Например, если в результате преобразования оказывается, что значение какого-либо компонента модели RGB меньше 0, то берѐтся значение, равное 0. На практике же каждый элемент матрицы изображения с 16777216 цветами представляет собой 24-битное значение, где каждый компонент модели RGB представлен 8ю битами. Поэтому вместо интервала [0, 1] используется интервал [0, 255].

Если яркость и контраст изображения никак не меняются в процессе преобразования, то функция имеет график, представленный на рис. 4.15, а. Из рисунка видно, что функция в этом случае просто передаѐт на выход значение своего аргумента.



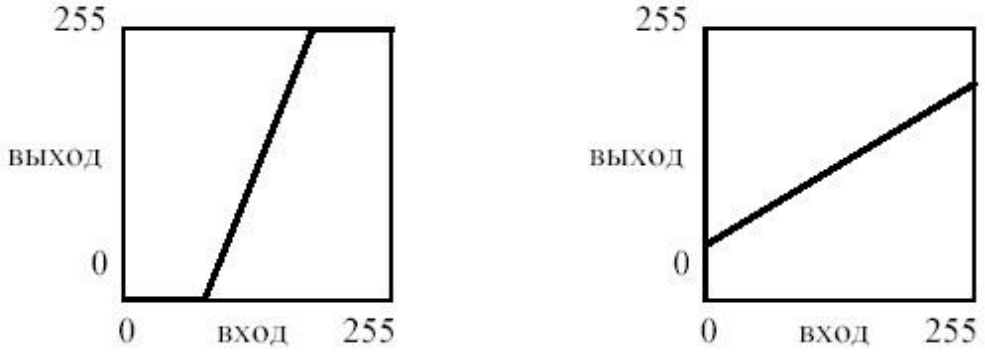
а б в

Рис. 4.15. Графики яркости

Яркость для рассматриваемой функции представляет собой сдвиг прямой линии в вертикальном направлении. Яркость изображения увеличивается пропорционально сдвигу прямой. Если прямая сдвигается вверх (рис. 4.15, б), яркость изображения увеличивается, а если прямая сдвигается вниз (рис. 4.15, в) – уменьшается.

Поскольку используется арифметика с насыщением, то при установке определѐнной яркости изображения либо оно полностью окажется засвеченным, либо полностью затемнѐнным.

При использовании преобразования контраста прямая линия меняет свой наклон. При увеличении контраста изображения (рис. 4.16, а) наклон прямой увеличивается, при уменьшении контраста – уменьшается (рис. 4.16, б). При этом сдвиг прямой в горизонтальном направлении означает, что помимо контраста изменяется и яркость изображения.



а б

Рис. 4.16. Графики контрастности

Комбинации наклона и сдвига прямой позволяют одновременно изменять и яркость, и контраст изображения. Например, на рис. 4.17 представлен график функции, усиливающей контраст и увеличивающей яркость изображения.

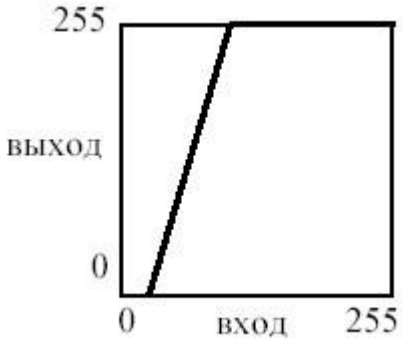


Рис. 4.17. Увеличение яркости и контрастности

Преобразование яркости/контраста может быть применено и к отдельным компонентам модели RGB, например к компоненту красного цвета. Тогда яркость/контраст будут изменяться только для красного компонента, а для других компонент они останутся неизменными. Более того, можно задавать различные преобразования яркости/контраста одновременно для каждого компонента модели RGB.

4.3.2. Масштабирование изображения

Масштабирование изображения позволяет сжать или растянуть его по горизонтали и/или вертикали. При этом изменяется ширина и/или высота изображения. Для масштабирования задаются масштабные коэффициенты – то, насколько нужно сжать/растянуть изображение по горизонтали или вертикали. Масштабные коэффициенты могут задаваться в нормализованной, процентной или непосредственной форме. В нормализованной форме за единицу принимаются размеры исходного изображения. Значения меньше единицы

указывают на сжатие изображения, значения больше единицы – на растяжение. В процентной форме нормализованные значения умножаются на 100 %. В непосредственной форме новые размеры по горизонтали и вертикали задаются в виде количества пикселей по тому или другому измерению.

Возникает вопрос о том, каким образом определять цвета при изменении размеров изображения. Существует два основных подхода к этой проблеме:

1. Цвет пикселя в масштабированном изображении принимается равным цвету ближайшего к нему пикселя исходного изображения.

2. Использование интерполяции. В этом случае цвет пикселя масштабируемого изображения вычисляется, как значение некоторой интерполирующей функции от цветов соседних пикселей в исходном изображении.

При использовании билинейной интерполяции цвет вычисляется, как взвешенная сумма ближайших четырѐх пикселей исходного изображения (при увеличении) или как взвешенная сумма группы пикселей (при уменьшении).

Первый подход достаточно прост, но не всегда даѐт приемлемое качество обработанного изображения. Например, если новый размер намного больше старого, то возникает блочная структура изображения, т. е. каждый пиксель исходного изображения соответствует квадратной области пикселей одного и того же цвета в обработанном изображении. Эта аномалия представлена на рис. 4.18.

С другой стороны, если новый размер намного меньше старого, то при масштабировании одному пикселю обработанного изображения соответствует группа пикселей исходного изображения, причѐм в процессе масштабирования фактически выбирается случайный пиксель из этой группы.

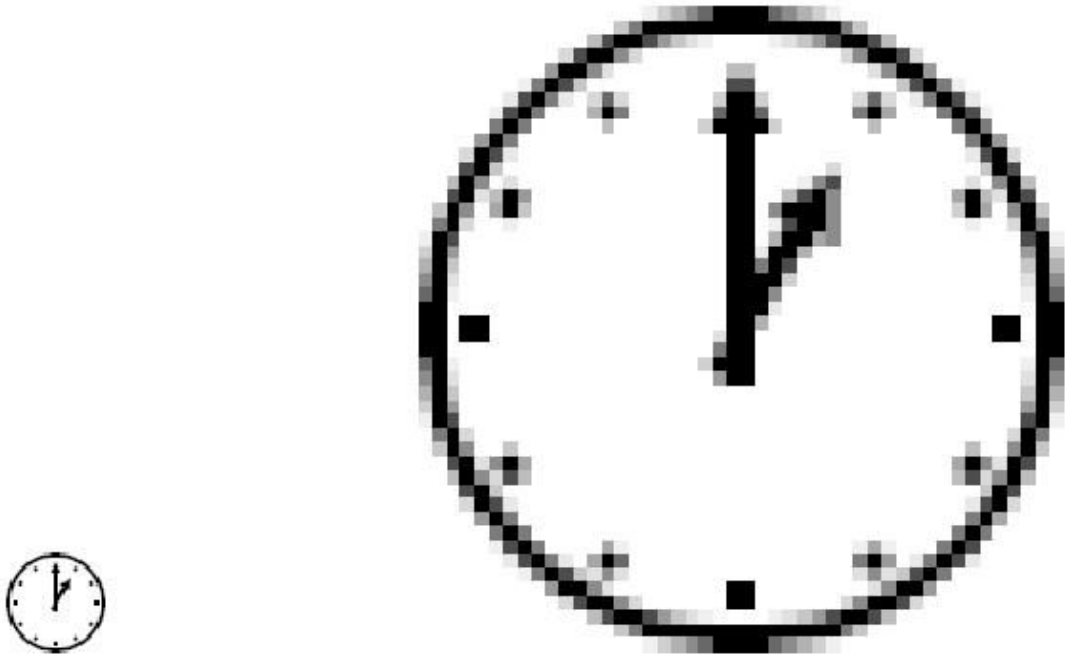


Рис. 4.18. Некорректное увеличение

Подход, использующий интерполяцию, позволяет достичь более высокого качества изображения, но более сложен в реализации. Обычно используется билинейная или бикубическая интерполяция. Бикубическая интерполяция позволяет получить изображение с более высоким качеством, чем билинейная интерполяция. Однако следует заметить, что при дальнейшем повышении порядка интерполяции качество получаемого изображения может улучшаться незначительно.

Приведем простейшую формулу, которая позволяет определить ближайший пиксель исходного изображения (без использования интерполяции):

$$C_{new}[i][j] = C_{old}\,[k_1 \cdot i]\,[k_2 \cdot j],\ \text{где}\ i \in 0, H_{old} - 1,\ j \in 0, W_{old} - 1;$$

$$k_1 = \frac{H_{old}}{H_{new}}, \qquad k_2 = \frac{W_{old}}{W_{new}},$$

Параметр W определяет размер изображения по горизонтали, измеряемый в пикселях. Параметр H определяет размер по вертикали. Параметры i и j определяют соответственно строку и столбец матрицы изображения и изменяются в пределах высоты и ширины изображения соответственно.

4.3.3. Преобразование поворота

Преобразование поворота, также как и при рассмотрении плоских геометрических объектов, позволяет поворачивать исходное изображение на заданный угол. Поворот осуществляется вокруг центра изображения. При этом возможны два варианта поворота:

1. Области изображения, вышедшие за его границы при повороте отсекаются, а незаполненные части заполняются каким-либо цветом.
2. Рассчитывается новый размер изображения на основе угла поворота таким образом, чтобы повèрнутое изображение целиком поместилось в новые размеры. Незаполненные части изображения также заполняются каким-либо цветом.

В любом случае для расчèта преобразования поворота может быть использована следующая формула:

$$C_{old}[\ ][\ ]a\ b\ a,\ \square[0,H_{old}\square\ \square\ \square 1]\qquad b\ [0,W_{old}\square 1];$$

$$C_{new}[i][j] = \square\square\ \ C\ a,\ \square[0,H_{old}\square\ \ \square\ \ \square 1]\ b\ [0,W_{old}\square 1];$$

$$\left|\ H^{new};\ b\ \square\ j\square\cos(\ )\square\ \square W^{new};\ a\ \square i\square\sin(\ )\square\ \square\ \right|\ \_\_\_\_$$

$$2\quad 2\ i\ \square\ 0,H_{old}\square 1,\ j\ \square\ 0,W_{old}\square 1.\ \_\_\_\_\_$$

В этой формуле параметр *C* определяет цвет, которым заполняются пустые участки изображения. Параметр φ определяет угол поворота по часовой стрелке в радианах.

Приведèнная формула округляет преобразованные координаты. Однако можно использовать и билинейную интерполяцию, когда цвет пикселя вычисляется как взвешенная сумма цветов четырèх соседних пикселей.

4.4. Цифровые фильтры изображений

Цифровые фильтры позволяют накладывать на изображение различные эффекты, например: размытие, резкость, деформацию, шум и т. д.

Цифровой фильтр представляет собой алгоритм обработки изображения. Большая группа цифровых фильтров имеет один и тот же алгоритм, но эффект, накладываемый фильтром на изображение, зависит от коэффициентов, используемых в алгоритме.

Фильтрация изображений является одной из самых фундаментальных операций компьютерного зрения, распознавания образов и обработки изображений. Фактически, с той или иной фильтрации исходных изображений начинается работа подавляющего большинства методов.

4.4.1. Линейные фильтры

Линейные фильтры представляют собой семейство фильтров, имеющих очень простое математическое описание. Вместе с тем они позволяют добиться самых разнообразных эффектов. Будем считать, что задано исходное полутоновое изображение A, и обозначим интенсивности его пикселей $A(x, y)$. ***Линейный фильтр*** определяется вещественнозначной функцией F, заданной на растре. Данная функция называется ядром фильтра, а сама фильтрация производится при помощи операции дискретной свертки [1] (взвешенного суммирования):

$$B(x, y) = \sum_i \sum_j F(i, j) \cdot A(x - i, y - j).$$

Результатом служит изображение B. В приведенной формуле не определены пределы суммирования. Обычно ядро фильтра отлично от нуля только в некоторой окрестности N точки (0, 0). За пределами этой окрестности $F(i, j)$ или в точности равно нулю, или очень близко к нему, так что можно им пренебречь. Поэтому суммирование производится по $(i, j) \in N$, и значение каждого пикселя $B(x, y)$ определяется пикселями изображения A, которые лежат в окне N, центрированном в точке (x, y) (обозначим это множество $N(x, y)$). Ядро фильтра, заданное на прямоугольной окрестности N, может рассматриваться как матрица $m \times n$, где длины сторон являются нечетными числами. При задании ядра матрицей M_{kl}, ее следует центрировать:

$$F(i, j) = M_{i+\frac{m-1}{2}, j+\frac{n-1}{2}}$$

Также нуждается в дополнительном прояснении ситуация, когда пиксель (x, y) находится в окрестности краев изображения. В этом случае $A(x+i, y+j)$ может соответствовать пикселю A, лежащему за границами изображения A. Данную проблему можно разрешить несколькими способами:

- не проводить фильтрацию для таких пикселей, обрезав изображение B по краям или закрасив их, к примеру, черным цветом;
- не включать соответствующий пиксель в суммирование, распределив его вес $F(i, j)$ равномерно среди других пикселей окрестности $N(x, y)$;
- доопределить значения пикселей за границами изображения при помощи экстраполяции;

[1] Свертка - это математическая операция двух функций f и g, порождающая третью функцию, которая обычно может рассматриваться как модифицированная версия одной из первоначальных.

- доопределить значения пикселей за границами изображения, при помощи зеркального отражения.

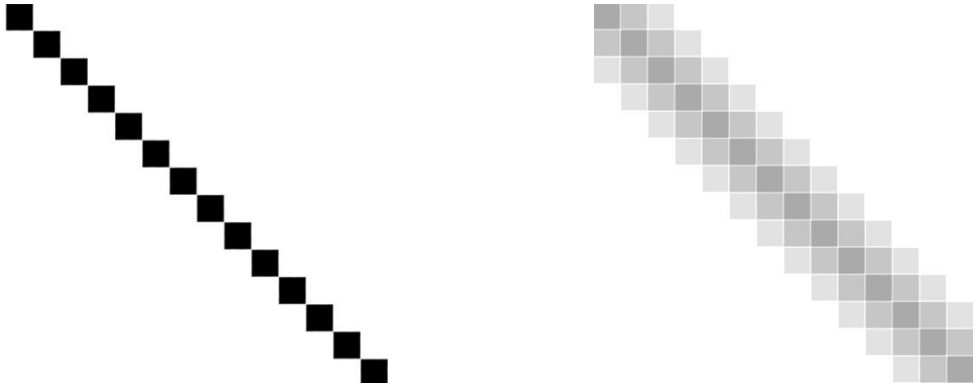
Выбор конкретного способа нужно производить с учетом конкретного фильтра и особенностей конкретного приложения. Разобрав общее определение линейных фильтров, перейдем к примерам.

4.4.2. Сглаживающие фильтры

Результатом применения сглаживающего фильтра является размытие изображения, устранение резких цветовых переходов. Простейший прямоугольный сглаживающий фильтр радиуса r задается при помощи матрицы размера $(2r + 1) \times (2r + 1)$, все значения которой равны:

$$\frac{1}{(2r+1)^2},$$

а сумма по всем элементам матрицы равна, таким образом, единице. При фильтрации с данным ядром значение пикселя заменяется на усредненное значение пикселей в квадрате со стороной $2r+1$ вокруг него. Пример фильтрации при помощи прямоугольного фильтра приведен на рис. 4.19.



а) б)

Рис. 4.19. Пример использования сглаживающего фильтра

В этом случае на исходное изображение (рис. 4.19, а) наложен прямоугольный фильтр размером 3 на 3 пикселя. Ядро фильтра в этом случае выглядит следующим образом:

$$M_{blur} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}.$$

Результат фильтрации приведен на изображении справа (рис. 4.19, б). Отметим, что сумма всех элементов ядра фильтра дает в результате единицу. Потому можно сказать, что при использовании такого фильтра в целом яркость всего изображения не меняется. Однако, в следствии усреднения значений цветов пикселей, контрастность изображения уменьшается, что и видно на рис. 4.19.

Как видим из приведенного примера, сглаживающие фильтры могут применяться для устранения лестничного эффекта, а также для шумоподавления.

Шум – дефект на изображении, вносимый фотосенсорами и электроникой устройств, или возникающий при использовании аналоговых устройств. Шум на изображении проявляется в виде случайным образом расположенных элементов растра (точек), имеющих размеры близкие к размеру пикселя. Шум отличается от изображения более светлым или тѐмным оттенком серого и цвета и/или по цвету. Причиной появления шумов на изображении является: зернистость плѐнки, грязь, пыль, царапины, отслоение фотографической эмульсии. Если рассматривать цифровые устройства то причиной возникновения цифрового шума является: тепловой шум матрицы, шум переноса заряда, шум квантования АЦП, усиление сигналов в цифровом фотоаппарате, грязь, пыль на сенсоре.

Применение линейной фильтрации с прямоугольным ядром имеет существенный недостаток: пиксели на расстоянии r от обрабатываемого оказывают на результат тот же эффект, что и соседние. Более эффективное шумоподавление можно осуществить, если влияние пикселей друг на друга будет уменьшаться с расстоянием. Этим свойством обладает ***гауссовский фильтр*** с ядром:

$$F_{gauss}(i, j) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right)$$

Гауссовский фильтр имеет ненулевое ядро бесконечного размера. Однако ядро фильтра очень быстро убывает к нулю при удалении от точки (0, 0), и потому на практике можно ограничиться сверткой с окном небольшого размера вокруг (0, 0) (например, взяв радиус окна равным 3σ).

Гауссовская фильтрация также является сглаживающей. Однако, в отличие от прямоугольного фильтра, образом точки при гауссовой фильтрации

будет симметричное размытое пятно, с убыванием яркости от середины к краям, что гораздо ближе к реальному размытию от расфокусированных линз.

4.4.3. Контрастоповышающие фильтры

Если сглаживающие фильтры снижают локальную контрастность изображения, размывая его, то контрастоповышающие фильтры производят обратный эффект. Ядро контрастоповышающего фильтра имеет значение, большее 1, в точке (0, 0), при общей сумме всех значений, равной 1.

Например, контрастоповышающим фильтром является фильтр с ядром, задаваемым матрицей:

$$M_{1contr} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

либо матрицей:

$$M_{2contr} = \frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Эффект повышения контраста достигается за счет того, что фильтр подчеркивает разницу между интенсивностями соседних пикселей, удаляя эти интенсивности друг от друга (рис. 4.20). Этот эффект будет тем сильней, чем больше значение центрального члена ядра. Результат работы фильтра на исходном изображении (рис. 4.20, а) для M_1^{contr} представлен на рис. 4.20, б, для M_2^{contr} на рис. 4.20, в.

а) б) в)

Рис. 4.20. Пример использования контарстноповышающего фильтра

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

4.4.4. Разностные фильтры

Разностные фильтры часто используются для нахождения границ в изображениях. При этом используют дифференциальный оператор, вычисляющий приближенное значение градиент [1] яркости изображения. Результатом применения такого оператора в каждой точке изображения является либо вектор градиента яркости в этой точке, либо его норма.

Результат показывает, насколько «резко» или «плавно» меняется яркость изображения в каждой точке, а значит, вероятность нахождения точки на грани, а также ориентацию границы. На практике, вычисление величины изменения яркости (вероятности принадлежности к границе) надежнее и проще в интерпретации, чем расчет направления.

Если на изображении будет присутствовать однотонная область или область с плавными переходами цветов, то в результирующем изображении подобные участки будут закрашены черным цветом. Там, где имеются перепады (резкие переходы, края), крутизна изменения яркости высока и в конечном изображении в таких местах появятся яркие светлые линии.

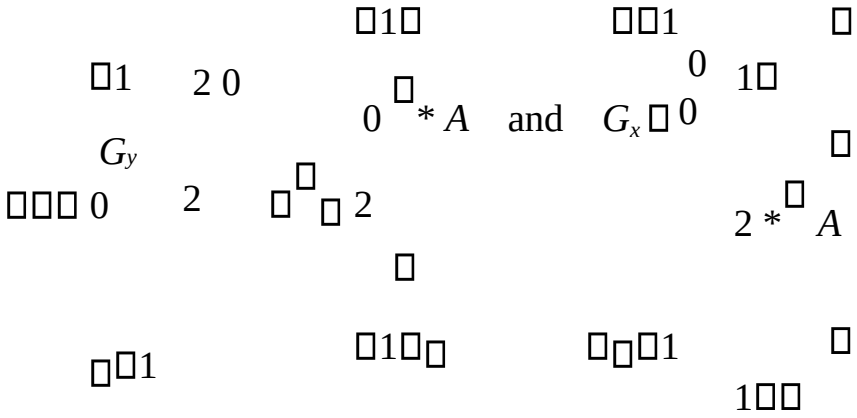
Математически, градиент функции двух переменных для каждой точки изображения (которой и является функция яркости) — двумерный вектор, компонентами которого являются производные яркости изображения по горизонтали и вертикали. В каждой точке изображения градиентный вектор ориентирован в направлении наибольшего увеличения яркости, а его длина соответствует величине изменения яркости.

Одним из способов нахождения границ на изображении является реализация фильтра или оператора Собеля (Sobel), который позволяет найти неточное приближение градиента яркости изображения.

Строго говоря, оператор использует ядра 3×3, с которыми сворачивают исходное изображение для вычисления приближенных значений производных по горизонтали и по вертикали. Формально оператор Собеля определяется следующим образом:

[1] Градиент — вектор, показывающий направление наискорейшего возрастания некоторой величины, значение которой меняется от одной точки пространства к другой.

Пусть A исходное изображение, а G_x и G_y - два изображения, где каждая точка содержит приближенные производные по x и по y. Они вычисляются следующим образом:



где * обозначает двухмерную операцию свертки (операцию линейной фильтрации, рассмотренную ранее).

Координата x здесь возрастает «направо», а y — «вниз». В каждой точке изображения приближенное значение величины градиента можно вычислить, используя полученные приближенные значения производных:

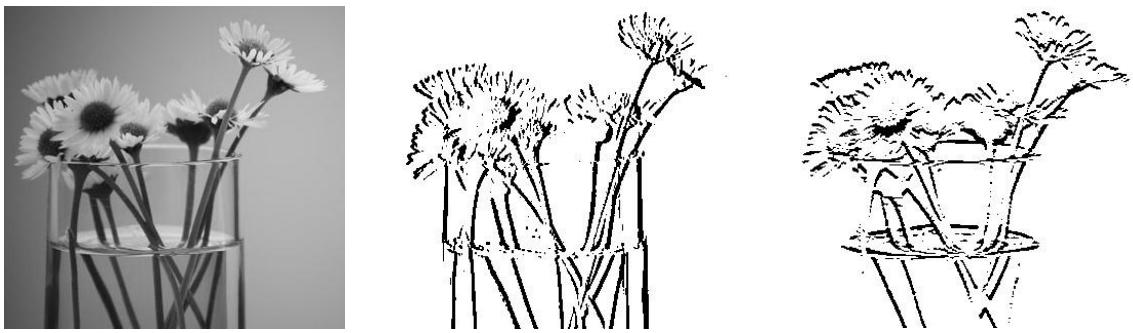
$$G \approx \sqrt{G_x^2 + G_y^2}$$

Используя эту информацию, мы также можем вычислить направление градиента:

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$

где, к примеру, угол Θ равен нулю для вертикальной границы, у которой тѐмная сторона слева.

Операции свертки G_x и G_y можно использовать отдельно для нахождения вертикальных и горизонтальных границ. На рис. 4.21 приведен пример применения свертки G_x (рис. 4.21, б) и G_y (рис. 4.21, в) к исходному изображению (рис. 4.21, а). В случае совместного использования двух операций свертки G_x и G_y можем получить результат представленный на рис. 4.22.



а) б) в)

Рис. 4.21. Пример нахождения вертикальных и горизонтальных границ

В отличии от сглаживающих и контрастоповышающих фильтров, не меняющих среднюю интенсивность изображения (сумма элементов ядра равна единице), в результате применения разностных операторов получается, как правило, изображение со средним значением пикселя близким к нулю (сумма элементов ядра равна нулю). При этом пиксели со значениями близкими к нулю можно отображать белым цветом. Пиксели значения яркости, которых получились большего некоего порога, можно отображать черным цветом.

Заметим, что выделение всех пикселей, значения которых по модулю больше некоторого порога, является некоторой нелинейной локальной операцией, которую можно рассматривать как простейший пример нелинейной фильтрации.

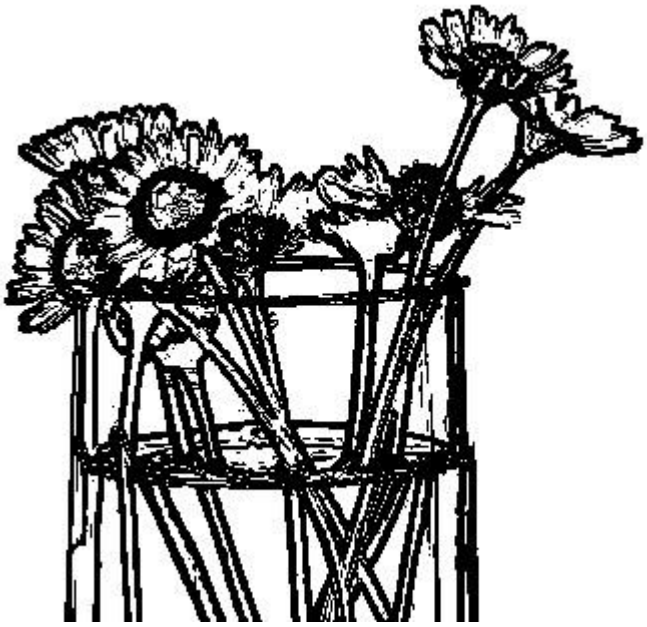


Рис. 4.22. Пример использования оператора Собеля

Кроме фильтра Собеля можно использовать оператор Прюита (Prewitt) или оператор Щарра (Scharr), подобные алгоритму оператора

Собеля, за исключением использования других матриц.

У алгоритмов нахождения границ существует несколько недостатков. 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.

4.4.5. Нелинейные фильтры

Основное отличие нелинейного фильтра от линейного заключается в том, что выход нелинейного фильтра формируется нелинейным образом от данных исходного изображения.

Линейные фильтры, несмотря на разнообразие производимых ими эффектов, не позволяют проделывать некоторые самые естественные операции. Хорошим примером служит пороговая фильтрация, упомянутая выше. Результатом пороговой фильтрации служит бинарное изображение, определяемое следующим образом:

$$B(x, y) = \begin{cases} 1, & \text{если } A(x, y) \geq \gamma \\ 0, & \text{иначе} \end{cases}$$

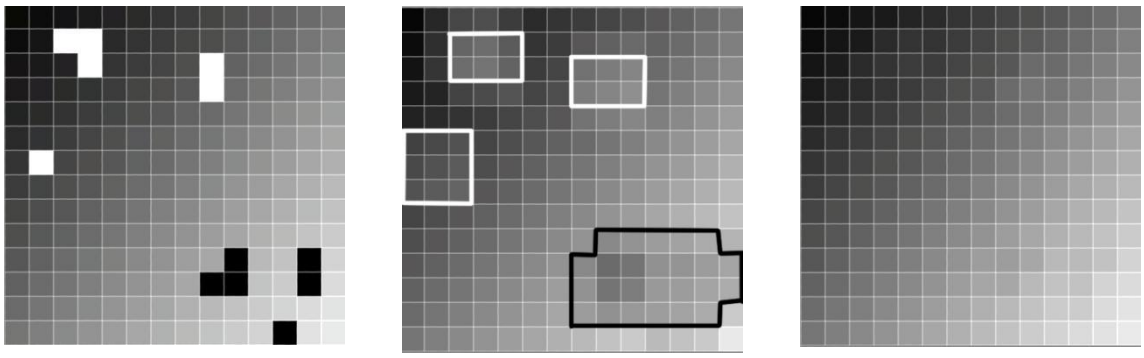
Величина γ является ***порогом фильтрации***. Пороговая фильтрация может быть использовано как предварительный этап обработки изображения перед его векторизацией.

Более сложным фильтром, нелинейным фильтром, использующим окрестность пикселя, является ***медиана*** или ***медианный фильтр***. Здесь, так же, как и в линейных фильтрах, по пикселям передвигается окно, которое охватывает пиксели, участвующие в формировании итоговой интенсивности. Значения внутри этого окна воспринимается как одномерный массив, который сортируется в порядке возрастания. Значение, находящееся в середине отсортированного массива, поступает на выход фильтра.

Таким образом, медианная фильтрация способна эффективно справляться с импульсными помехами, когда помехи независимо воздействуют на отдельные пиксели. Примером таких помех служат "битые" и "горячие" пиксели при цифровой съемке, "снеговой" шум и т.

п. Например, рис. 4.23, а. Преимущество медианной фильтрации (рис. 4.23, в) перед линейной сглаживающей фильтрацией (рис. 4.23, б) заключается

в том, что интенсивность пикселя шума будет заменена интенсивностью фоновых пикселей, а не будет перераспределена на соседние пиксели как при использовании сглаживающего фильтра.



а) б) в)

Рис. 4.23. Удаление шумов фильтрами размытия и медианным фильтром

5. Преобразования растровых изображений

5.1. Векторизация с помощью волнового алгоритма

Векторизация – преобразование изображения из растрового представления в векторное. Векторизация - процесс, обратный растеризации. Векторизация проводится, как правило, в случае, если результат векторизации подлежит дальнейшей обработке исключительно в программах векторной графики, с целью повышения качества изображения, для создания изображения, пригодного для масштабирования без потери качества, если дальнейшая обработка изображения будет осуществляться на специфическом оборудовании (плоттеры, станки с ЧПУ).

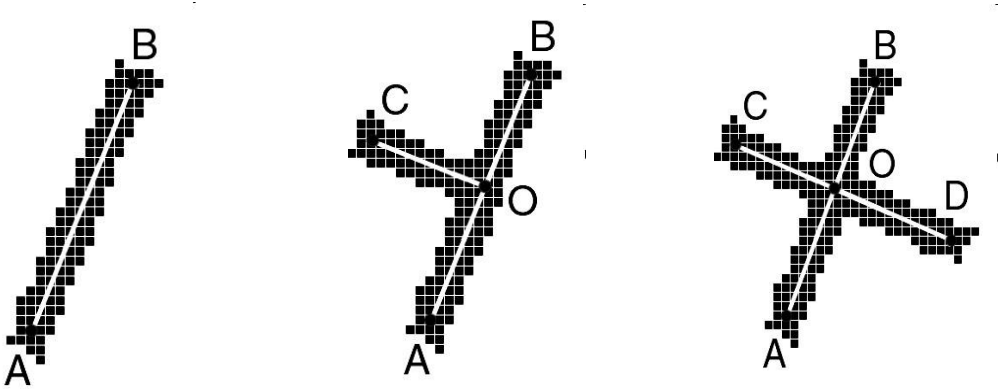
Различают следующие виды векторизации:

- Автоматическая векторизация — перевод растровых изображений чертежей (сканированных копий, фотографий) в электронный вид с помощью специального программного обеспечения.
- Ручная векторизация — перевод бумажных чертежей в электронный вид перечерчиванием каждого документа вручную в системах автоматизированного проектирования.

Одним из алгоритмов для нахождения векторного представления из растрового изображения (отсканированного чертежа) является волновой алгоритм. На вход алгоритма поступает бинарное растровое изображение, которое может быть получено из исходного путем пороговой фильтрации. Под бинарным растровым изображением понимается двумерная матрица из черных

и белых пикселей, в которой объект задается черными пикселями, а фон – белыми пикселями.

На растровом изображении могут присутствовать отрезки, соединения и пересечения отрезков (рис. 5.1). Включение дуг в изображение в данном пособии не рассматривается.



а) б) в)

Рис. 5.1. Объекты растрового изображения

Под изображением ***отрезка*** на растре будем понимать такое множество черных точек растра, что можно провести отрезок прямой *AB* такой, что по обе стороны от этого отрезка будет лежать примерно равное количество точек, и расстояния от отрезка до ближайших крайних точек изображения будут отличаться не более чем на наперед заданную величину (рис. 5.1, а).

Под ***соединением*** отрезков будем понимать два отрезка такие, что одна из вершин одного отрезка лежит на другом отрезке. В этом случае они будут иметь одну общую точку и один из отрезков будет разделен на два (рис. 5.1, б).

Пересекающиеся отрезки изображения будем представлять в виде отрезков меньшей длины, имеющих одну общую вершину (рис. 5.1, в).

Задачей алгоритма является построение нагруженного графа, у которого нагрузка вершин – пары координат *x*, *y* соответствующих узловым точкам изображения.

За узловые точки будем принимать точки соединения изображения линий в растровом изображении. Толщиной линии будем считать ее ширину в пикселях.

Алгоритм реализуется в два этапа:

- Построение скелета изображения с помощью сферической волны;
- Оптимизация полученного скелета.

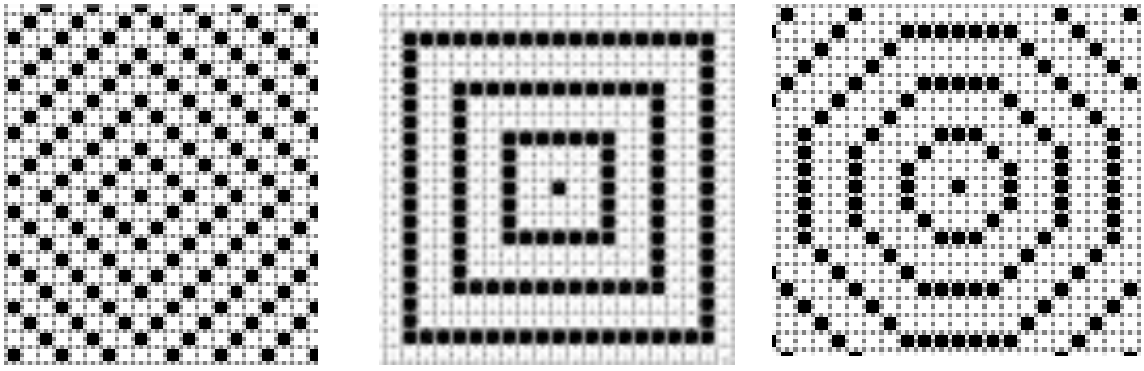
До построения скелета, с целью улучшения качества изображения возможно применение различных фильтров. В частности, желательно избавить изображение от случайного шума.

5.1.1. Построение скелета изображения

Для построения скелета изображения на объекте (отрезке) выбирается начальная (затравочная) точка, от которой начинается распространяться волна, путем добавления пикселей сначала к затравочной точке потом к уже добавленным. Волна генерируется по определенным законам, использующих четырех или восьмисвязанность пикселей.

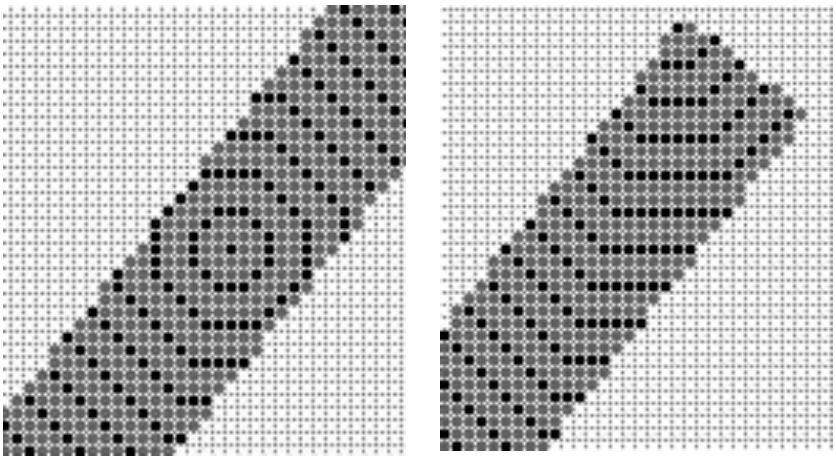
При 4-х связном растре распространение идет в форме ромба (рис. 5.2, а), при 8-связном – в виде квадрата (рис. 5.2, б). Если попеременно использовать 4-ч и 8-и связанное распространение то форма волны будет близка к кругу (рис. 5.2, в).

При распространении сферической волны на отрезке прямой наблюдаются следующие эффекты: не более чем через $2N$ шагов распространение волны приобретает устойчивый характер вне зависимости от начальной точки распространения волны (Рисунок 5). Причем N – ширина линии в пикселях.



а) б) в)

Рис. 5.2. Формы распространения волн на растровом изображении



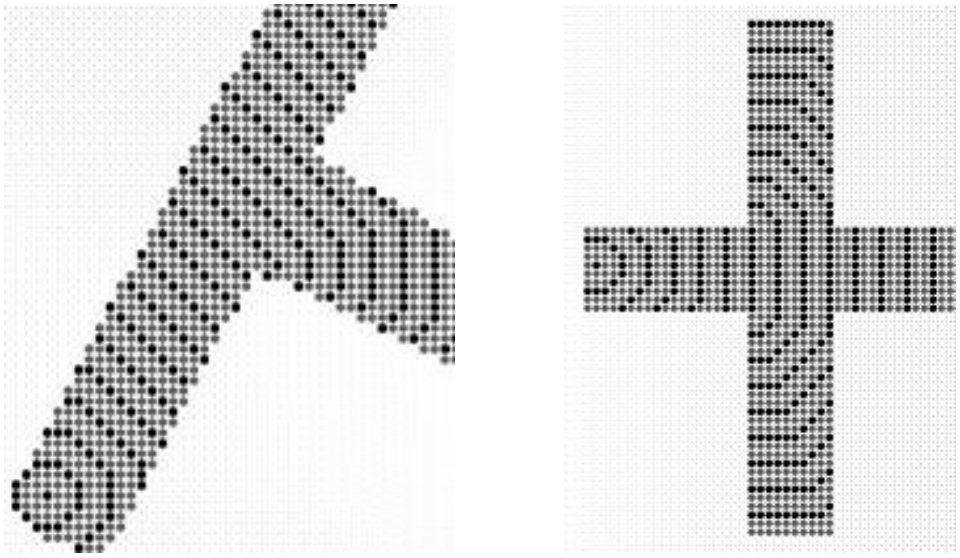
а) б)

Рис. 5.3. Распространение волны на отрезке

При отличии формы отрезка от прямой, распространение волны также предсказуемо, причем необходимо отметить хорошие огибающие свойства сферической волны.

Узловые точки определяются на каждой генерации волны или через определенное количество генераций волн. В качестве узловой точки берется центр отрезка, образуемого крайними точками необходимой генерации волны.

При достижении волной места соединения двух или более отрезков наблюдается разделение волны на несколько дочерних волн, сохраняющих поведение материнской волны (рис. 5.4). Момент разделения довольно просто отслеживается путем анализа ―ширины‖ волн, т.е. количества точек образующих очередную генерацию волны: перед разделением наблюдается увеличение ―ширины‖ волны с дальнейшим разделением волны на две (иногда более) дочерние волны.



а) б)

Рис. 5.4. Распространение волны на соединении и пересечении отрезков

Выявление мест увеличения ―ширины‖ волны и разделения волны на дочерние позволяет установить точку предполагаемого соединения двух отрезков. Определение увеличения ―ширины‖ волны производится путем сравнения ―ширины‖ очередной генерации волны и ее среднего значения за N предыдущих генераций (N задается заранее). Причем мы получаем две крайние точки (A, B) трассируемого отрезка. После разделения волны на две полуволны, мы получаем еще две пары точек (C, D) и (E, F) (например, рис. 5.5).

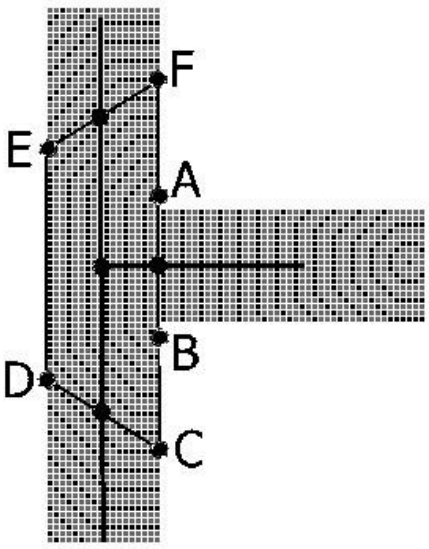
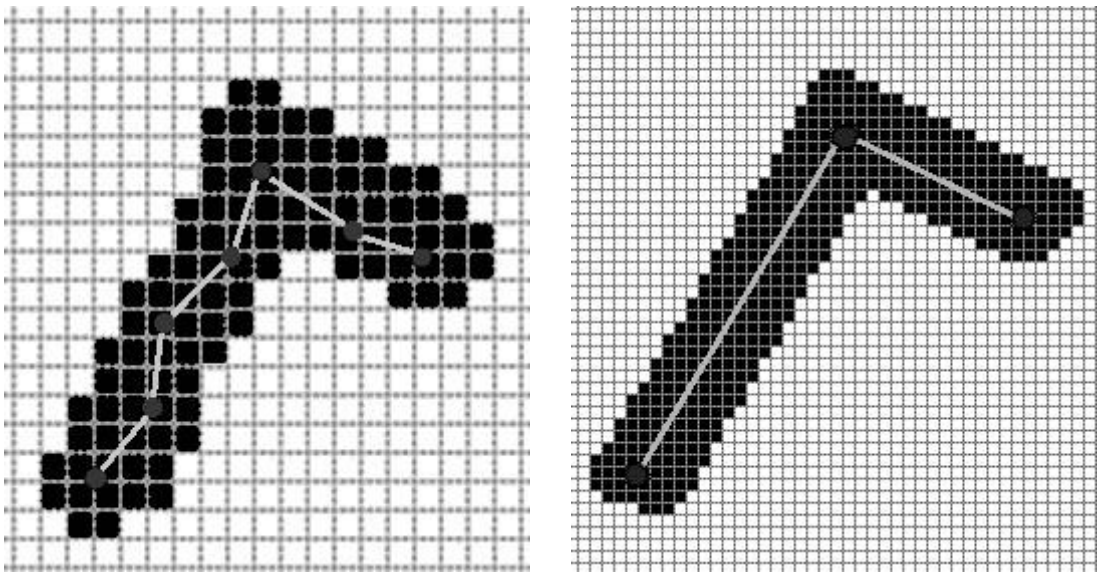


Рис. 5.5. Пример определения точки соединения

С помощью анализа этих отрезков (AB, CD, EF) можно оценить положение точки соединения отрезков. Первоначально установим место соединения как центр масс этого многоугольника.

5.1.2. Оптимизация скелета изображения

Полученный скелет изображения не является оптимальным. Это связано прежде всего с тем, что мы имеем дело с растровым изображением, а значит, изображение имеет искажения тем большие, чем меньше разрешение изображения (рис. 5.6).



а) б)

Рис. 5.6. Влияние разрешения на скелет изображения

Для уменьшения влияния искажений на получаемый скелет необходимо произвести оптимизацию скелета, полученного отслеживанием пути сферической волны по изображению объекта. В получаемом скелете возможно представление одного отрезка некоторой последовательностью ребер. Избиться от этого можно анализом последовательности ребер, оценивая отклонение получающейся линии от прямой. При этом точки, образующие последовательность ребер, должны отклоняться от коррелирующей прямой не больше, чем на заранее заданную величину, соизмеримую с шириной линии. В случае, если отклонение находится в допустимых пределах, необходимо в скелете заменить соответствующую последовательность ребер на одно

(рис. 5.7).

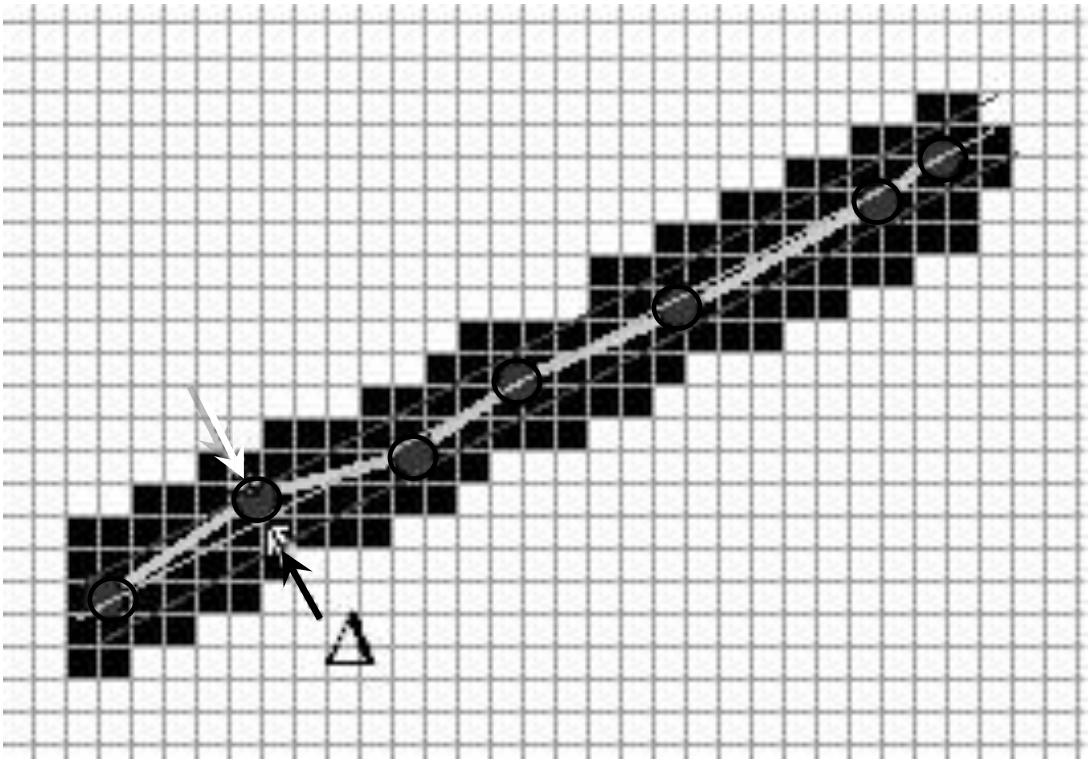
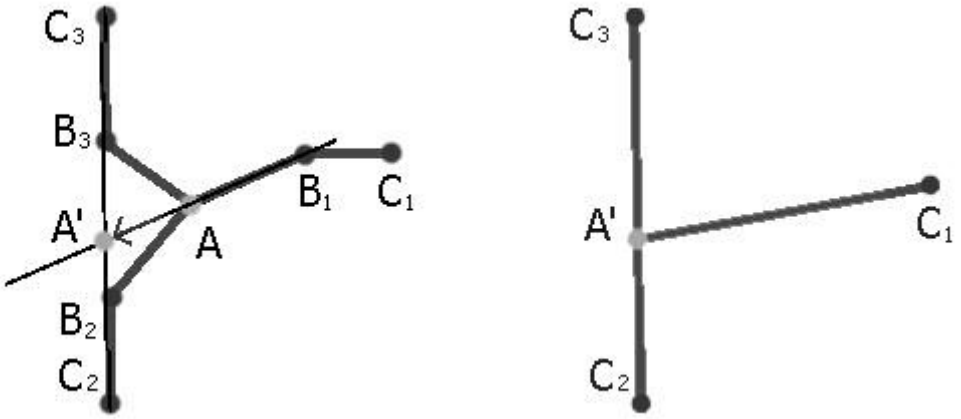


Рис. 5.7. Оптимизация отрезков

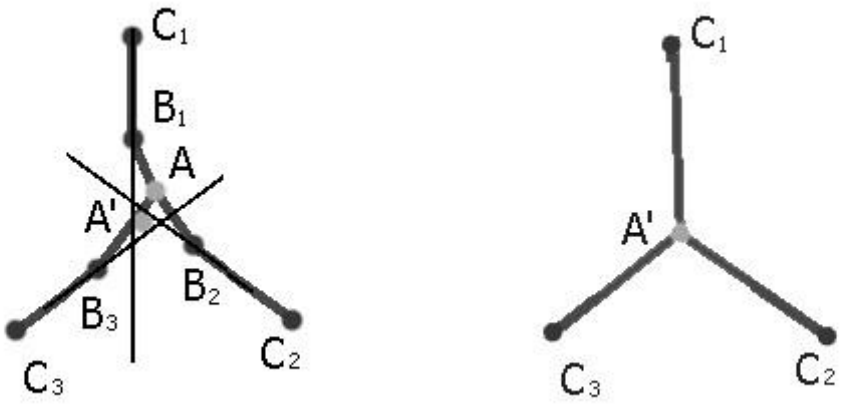
Кроме оптимизации отрезков проводится оптимизация точек соединения отрезков. Наиболее часто встречающиеся искажения (рис. 5.8) исправляются с помощью анализа прилежащих к выделенной точке (A) отрезков (AB_1, B_1C_1, AB_2, B_2C_2, AB_3, B_3C_3). Анализ заключается в поиске такой пары отрезков C_xB_x, B_yC_y из (B_1C_1, B_2C_2, B_3C_3), что $C_xB_xB_yC_y$ максимально коррелируются прямой. Тогда необходимо точку A переместить в точку пересечения прямых C_xC_y и AC_2, а затем удалить из графа точки B_1, B_2 и B_3.



а) б)

Рис. 5.8. Оптимизация точек соединения (первый вариант)

Другим вариантом искажения является случай соединения трех отрезков в одной точке (рис. 5.9). В этом случае невозможно нахождение пары отрезков коррелируемых прямой. Точка A должна быть перемещена в центр треугольника образуемого прямыми B_1C_1, B_2C_2 и B_3C_3. Затем точки B_1, B_2 и B_3 необходимо удалить из графа.



а) б)

Рис. 5.9. Оптимизация точек соединения (второй вариант)

5.2. Сегментация изображений

Сегментация – это процесс разбиения изображения на неперекрывающиеся области (сегменты), покрывающие все изображение и однородные по некоторому признаку. Все пиксели в сегменте похожи по некоторой характеристике или вычисленному свойству, например по цвету, яркости или текстуре. Соседние сегменты значительно отличаются по этой характеристике.

Цель сегментации заключается в упрощении и/или изменении представления изображения, чтобы его было проще и легче анализировать. Сегментация и выделение границ объектов играют важную роль в системах компьютерного зрения и применяются для задач распознавания сцен и выделения (определения) объектов.

Методы сегментации можно разделить на два класса: автоматические – не требующие взаимодействия с пользователем и интерактивные – использующие пользовательский ввод непосредственно в процессе работы. Например, к интерактивной сегментации можно отнести алгоритм инструмента «Волшебная палочка», реализованный во многих растровых редакторах.

Далее рассмотрим только автоматические методы. Задачи автоматической сегментации делятся на два класса:

- выделение областей изображения с известными свойствами;
- разбиение изображения на однородные области.

Между этими двумя постановками задачи есть принципиальная разница. В первом случае задача сегментации состоит в поиске определенных областей, о которых имеется априорная информация (например, мы знаем цвет, форму областей, или интересующие нас области представляют собой изображения известного объекта). Методы этой группы узко специализированы для каждой конкретной задачи. Сегментация в такой постановке используется в основном в задачах машинного зрения (анализ сцен, поиск объектов на изображении).

Во втором случае никакая априорная информация о свойствах областей не используется, зато на само разбиение изображения накладываются некоторые условия (например, все области должны быть однородны по цвету и текстуре). Так как при такой постановке задачи сегментации не используется априорная информация об изображенных объектах, то методы этой группы универсальны и применимы к любым изображениям. В основном сегментация в этой постановке применяется на начальном этапе решения задачи, для того чтобы получить представление изображения в более удобном виде для дальнейшей работы.

5.2.1. Методы, основанные на кластеризации

В постановке задачи сегментации прослеживается аналогия с задачей кластеризации [1]. Для того чтобы свести задачу сегментации к задаче кластеризации, достаточно задать отображение точек изображения в некоторое пространство признаков и ввести метрику (меру близости) на этом

[1] Кластерный анализ (англ. Data clustering) — задача разбиения заданной выборки объектов (ситуаций) на подмножества, называемые кластерами, так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались.

пространстве признаков. В качестве признаков точки изображения можно использовать представление ее цвета в некотором цветовом пространстве, примером метрики (меры близости) может быть евклидово расстояние между векторами в пространстве признаков.

После сведения задачи сегментации к задаче кластеризации можно воспользоваться любыми методами кластерного анализа. Наиболее популярные методы кластеризации, используемые для сегментации изображений – **метод *к*-средних.**

Алгоритм *k*-средних — это итеративный метод, который используется, чтобы разделить изображение на *K* кластеров. Опишем словесно алгоритм следующим образом:

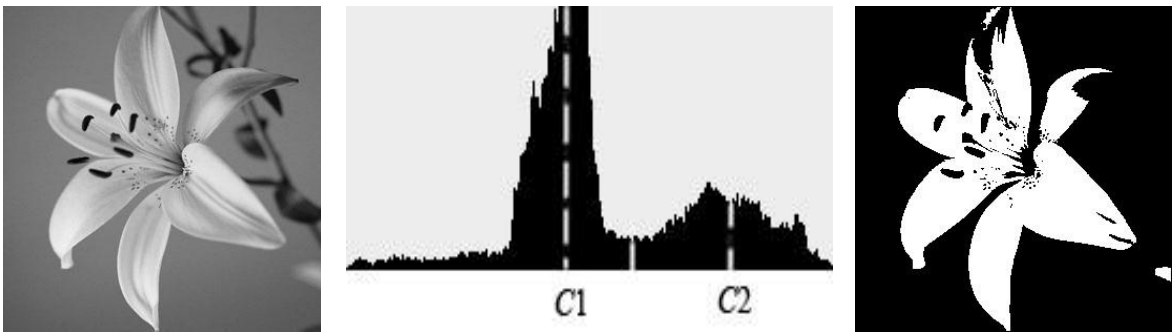
1. Выбрать *K* центров кластеров, случайно или на основании некоторой эвристики.
2. Поместить каждый пиксель изображения в кластер, центр которого ближе всего к этому пикселю.
3. Заново вычислить центры кластеров, усредняя все пиксели в кластере.
4. Повторять шаги 2 и 3 до сходимости (например, когда пиксели будут оставаться в том же кластере).

Здесь в качестве расстояния обычно берѐтся сумма квадратов или абсолютных значений разностей между пикселем и центром кластера. Разность обычно основана на цвете, яркости, текстуре и местоположении пикселя, или на взвешенной сумме этих факторов. *K* может быть выбрано вручную, случайно или эвристически.

Этот алгоритм гарантированно сходится, но он может не привести к оптимальному решению. Качество решения зависит от начального множества кластеров и значения *K*.

Применить метод *k*-средних к изображению можно рассматривая пространство яркостей пикселей. В этом случае кластеризация является одномерной задачей по одному признаку – яркости. Одномерное пространство яркостей пикселей, хорошо представимо гистограммой

(рис. 5.10).



а) б) в)

Рис. 5.10. Применение *k*-средних для сегментации изображений по яркости Для приведенного изображения (рис. 5.10, а) метод *k*-средних для *K* равного двум определит два кластера с центрами в точках *C*1 и *C*2 (рис. 5.10, б), и изображение будет разделено на два сегмента, закрашенных черным и белым цветом (рис. 5.10, в). В этом случае можно сказать, что метод *k*-средних провел пороговую фильтрацию и получил бинарное изображение.

Одна из проблем методов кластеризации, состоит в том, что пространственное расположение точек либо не учитывается совсем, либо учитывается косвенно (например, используя координаты точки как один из признаков). Поэтому обычно после кластеризации точек изображения проводят процедуру выделения связных компонент.

Методы кластеризации, также, плохо работают на зашумленных изображениях: часто теряют отдельные точек регионов, образуется много мелких регионов, и. т. п.

Подобные результаты, приведенные выше (рис. 5.10), можно получить, не используя методов кластерного анализа. В этом случае используется анализ гистограммы. При этом гистограмма вычисляется по всем пикселям изображения и еѐ минимумы и максимумы используются, чтобы найти сегменты на изображении.

Методы с использованием гистограммы очень эффективны, когда сравниваются с другими методами сегментации изображений, потому что они требуют только один проход по пикселям. Недостатком таких методов является дискретная природа гистограммы, что затрудняет поиск значительных минимумов и максимумов на ней.

5.3. Алгоритм разрастания регионов

Алгоритм разрастания регионов (—region growing‖) является методом автоматической сегментации и учитывает пространственное расположение точек напрямую.

Метод разрастания регионов основан на следующей идее. Сначала по некоторому правилу выбираются центры регионов (seeds), к которым поэтапно присоединяются соседние точки, удовлетворяющих некоторому критерию. Процесс разрастания регионов останавливается, когда ни одна точка изображения не может быть присоединена ни к одному региону.

Применяются разные критерии, на основании которых точка присоединяется или не присоединяется к региону: близость (в некотором смысле) точки к центру региона; близость к соседней точке, присоединенной к региону на предыдущем шаге; близость по некоторой статистике региона; стоимость кратчайшего пути от точки до центра региона, и т. п.

В основном процедура разрастания регионов используется для получения отдельных регионов, однако, применяя эту процедуру последовательно или одновременно для нескольких регионов, можно получить разбиение всего изображения. Существуют различные стратегии выбора зерен (seeds) и выращивания регионов.

Простейшая стратегия заключается в сканировании изображения сверху вниз, слева направо. На *i*-м шаге этого алгоритма рассматривается точка *A*, и сформированы регионы *B* и *C* слева и выше от пикселя *A* (рис. 5.11).

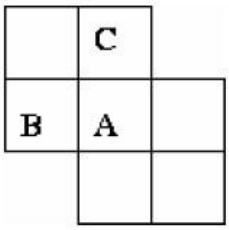


Рис. 5.11. *i* шаг алгоритма разрастания регионов

Определим $I(A)$ как яркость пикселя *A*, а $Cl_{avg}(B)$ и $Cl_{avg}(C)$ как средние яркости областей, которым принадлежат точки *B* и *C* соответственно. Обозначим через δ пороговое значение, которое задает чувствительность метода при создании нового сегмента.

На *i*-м шаге алгоритма проверяются следующие условия и выполняются соответствующие действия:

1. Если $|I(A) - Cl_{avg}(B)| > \delta$ & $|I(A) - Cl_{avg}(C)| > \delta$, то создаем новую область и присоединяем к ней пиксель *A*.
2. Если $|I(A) - Cl_{avg}(B)| \le \delta$ xor[1] $|I(A) - Cl_{avg}(C)| \le \delta$, то создаем новую область и присоединяем к ней пиксель *A*.
3. Если $|I(A) - Cl_{avg}(B)| \le \delta$ & $|I(A) - Cl_{avg}(C)| \le \delta$, то проверяем:

[1] xor – логическая операция «исключающая ИЛИ»

1. Если $|Cl_{avg}(B) - Cl_{avg}(C)| \le \delta$, то сливаем области B и C.
2. Если $|Cl_{avg}(B) - Cl_{avg}(C)| > \delta$, то добавляем пиксель A к тому классу, отклонение от которого минимально.

Недостатком описанного алгоритма является высокие вычислительные затраты.

6. Компьютерная геометрия

6.1. Двумерные преобразования

Компьютерная геометрия есть математический аппарат, положенный в основу компьютерной графики. В свою очередь, основу компьютерной геометрии составляют различные преобразования точек и линий. При использовании машинной графики можно по желанию изменять масштаб изображения, вращать его, смещать и трансформировать для улучшения наглядности перспективного изображения. Все эти преобразования можно выполнить на основе математических методов, которые мы будем рассматривать далее.

Преобразования, как и компьютерную геометрию, разделяют на двумерные (или преобразования на плоскости) и трехмерные (или пространственные). Вначале рассмотрим преобразования на плоскости.

Для начала заметим, что точки на плоскости задаются с помощью двух ее координат. Таким образом, геометрически каждая точка задается значениями координат вектора относительно выбранной системы координат. Координаты точек можно рассматривать как элементы матрицы $[x,y]$, т. е. в виде вектор-строки или вектор-столбца. Положением этих точек управляют путем преобразования матрицы.

Точки на плоскости xy можно перенести в новые позиции путем добавления к координатам этих точек констант переноса: $[x^* y^*] = [x\ y] + [a\ b] = [x+a\ y+b]$.

Таким образом, для перемещения точки на плоскости надо к матрице ее координат прибавить матрицу коэффициентов преобразования.

Рассмотрим результаты матричного умножения матрицы $[x,y]$, определяющей точку P, и матрицы преобразований 2×2 общего вида: $[x\ y]\begin{bmatrix} a & b \\ c & d \end{bmatrix} = [ax+cy\quad bx+dy] = [x^* y^*]$.

Проведем анализ полученных результатов, рассматривая x^* и y^* как преобразованные координаты. Для этого исследуем несколько частных случаев.

Рассмотрим случай, когда $a = d = 1$ и $c = b = 0$. Матрица преобразований приводит к матрице, идентичной исходной: $[x\ y]\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = [1x+0y\ \ 0x+1y] = [x\ y] = [x^*\ y^*]$.

При этом изменений координат точки P не происходит. Если теперь $d = 1$, $b = c = 0$, $a = \text{const}$, то:

$$[x\ y]\begin{bmatrix}a & 0\\ 0 & 1\end{bmatrix} = [ax+0y\ \ 0x+1y] = [ax\ y] = [x^*\ y^*].$$

Как видно, это приводит к изменению масштаба в направлении x, так как $x^*=ax$. Следовательно, данное матричное преобразование эквивалентно перемещению исходной точки в направлении x.

Теперь положим $b = c = 0$, т. е. $[x\ y]\begin{bmatrix}a & 0\\ 0 & d\end{bmatrix} = [ax+0y\ \ 0x+dy] = [ax\ dy] = [x^*\ y^*]$.

В результате получаем изменение масштабов в направлениях x и y. Если $a \neq d$, то перемещения вдоль осей неодинаковы. Если $a = d > 1$, то имеет место увеличение масштаба координат точки P. Если $0 < a=d < 1$, то будет иметь место уменьшение масштаба координат точки P.

Если a или (и) d отрицательны, то происходит отображение координат точек. Рассмотрим это, положив $b = c = 0$; $d = 1$ и $a = -1$, тогда $[x\ y]\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix} = [-1x+0y\ \ 0x+1y] = [-x\ y] = [x^*\ y^*]$.

Произошло отображение точки относительно оси y. В случае $b = c = 0$, $a = 1$, $d = -1$, отображение происходит относительно оси x. Если $b = c = 0$, $a = d < 0$, то отображение будет происходить относительно начала координат.

Заметим, что отображение и изменение масштаба вызывают только диагональные элементы матрицы преобразования.

Теперь рассмотрим случай, когда $a = d = 1$, а $c = 0$, т. е. $[x\ y]\begin{bmatrix}1 & 0\\ b & 1\end{bmatrix} = [x\ (bx+y)] = [x^*\ y^*]$.

Координата x точки P не изменяется, в то время как y^* линейно зависит от начальных координат. Этот эффект называется *сдвигом*. Аналогично, когда $a = d = 1$, $b = 0$, преобразование осуществляет сдвиг пропорционально координате y.

Заметим, что преобразование общего вида, примененное к началу координат, не приведет к изменению координат точки (0,0). Следовательно, начало координат инвариантно при общем преобразовании. Это ограничение преодолевается за счет использования однородных координат.

Если подвергнуть общему преобразованию различные геометрические фигуры, то можно установить, что параллельные прямые преобразуются в параллельные прямые, середина отрезка – в середину отрезка, параллелограмм – в параллелограмм, точка пересечения двух линий – в точку пересечения преобразованной пары линий.

Преобразование единичного квадрата

Четыре вектора положения точек единичного квадрата с одним углом в начале координат записываются в виде

A 0 0

B 1 0

C 1 1

D 0 1

Применение общего матричного преобразования $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ к

единичному квадрату приводит к следующему:

A 0 0 A^*

B $1\ 0 \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} a & b \end{bmatrix}$ B^*.

C $1\ 1 \begin{bmatrix} c & d \end{bmatrix} \begin{bmatrix} a+c & b+d \end{bmatrix}$ C^*

D 0 1 $\begin{bmatrix} c & d \end{bmatrix}$ D^*

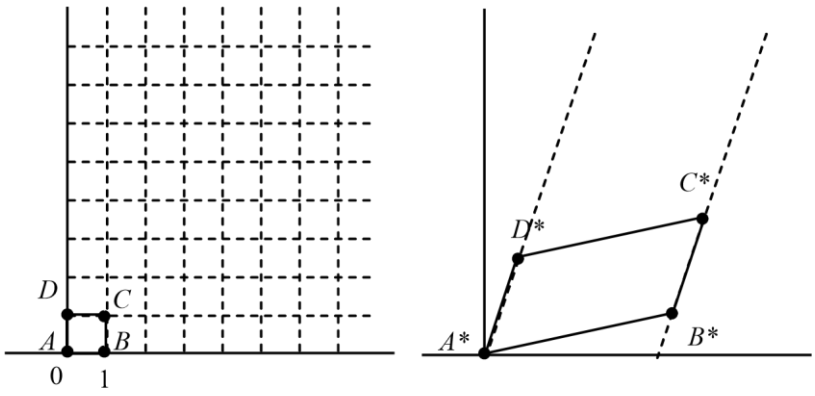


Рис. 6.1. Преобразования единичного квадрата

Из полученного соотношения можно сделать вывод, что координаты *B** определяются первой строкой матрицы преобразования, а координаты *D** второй строкой этой матрицы. Таким образом, если координаты точек *B** и *D** известны, то общая матрица преобразования определена. Воспользуемся этим свойством для нахождения матрицы преобразования для вращения на произвольный угол.

Общую матрицу 2□2, которая осуществляет вращение фигуры относительно начала координат, можно получить из рассмотрения вращения единичного квадрата вокруг начала координат.

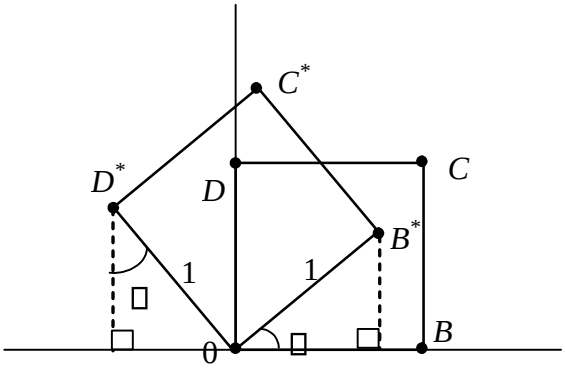


Рис. 6.2. Вращение единичного квадрата

Как следует из рис. 6.2, точка *B* с координатами (1,0) преобразуется в точку B^*, для которой x^*=(1)cos □ и *y*=(1)sin □, а точка *D*, имеющая координаты (0,1) переходит в точку D^* с координатами x^*=(-1)sin □ и y^*=(1)cos □. Матрица преобразования общего вида записывается так:

□ cos□ sin□□

□ □ sin□ cos□ □.

□

Для частных случаев. Поворот на 90□ можно осуществить с помощью матрицы преобразования

□ 0 1□

□□10□□.

□

Если использовать матрицу координат вершин, то получим, например:

□3 □1□ □ 1 3□

□□□□4211 □□□□□□□□□□01 10□□□□□□□□□11 42□□□□ .

Поворот на 180□ получается с помощью матрицы $\begin{matrix} \square & \square 1 \\ \square 0 & \end{matrix}$ $\square^{0\ \square}_{1\ \square\square}$.

□

Отображение

В то время как чистое двумерное вращение в плоскости *xy* осуществляется вокруг оси, перпендикулярной к этой плоскости, отображение определяется поворотом на 180□ вокруг оси, лежащей в плоскости *xy*.

Такое вращение вокруг линии $y = x$ происходит при использовании

□0 1□ матрицы $\square_{1\ 0}\square\square$.

□

Преобразованные новые выражения определяются соотношением

□8 1□ □1 8□

□□□7632□□□□□□□10 10□□□ □□□32 76□□□□.

□

Вращение вокруг $y = 0$ получается при использовании матрицы

□1 0 □

□0 □1□□.

□

6.1.1. Однородные координаты

Преобразования переноса, масштабирования и поворота записываются в матричной форме в виде

$$P^* = P \cdot T,$$

$$P^* = P \cdot S,$$

$$P^* = P \cdot R.$$

Очевидно, что перенос, в отличие от масштабирования и поворота, реализуется с помощью сложения. Это обусловлено тем, что вводить константы переноса внутрь структуры общей матрицы размером 2×2 не представляется возможным. Желательным является представление преобразований в единой форме – с помощью умножения матриц. Эту проблему можно решить за счет введения третьей компоненты в векторы точек $[x\ y]$ и $[x^*\ y^*]$, т. е. представляя их в виде $[x\ y\ 1]$ и $[x^*\ y^*\ 1]$. Матрица преобразования после этого становится матрицей размером 3×2:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ m & n \end{bmatrix}.$$

Это необходимо, поскольку число столбцов в матрице, описывающей точку, должно равняться числу строк в матрице преобразования для выполнения операции умножения матриц. Таким образом,

$$[x\ y\ 1]\cdot\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ m & n \end{bmatrix} = [x + m\ \ y + n] = [x^*\ y^*],$$

$[m\ n]$ откуда следует, что константы m, n вызывают смещение x^* и y^* относительно x и y. Поскольку матрица 3×2 не является квадратной, она не имеет обратной матрицы. Эту трудность можно обойти, дополнив матрицу преобразования до квадратной размером 3×3. Например,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}.$$

$[m\ n\ 1]$

Заметим, что третья компонента векторов положения точек не изменяется при добавлении третьего столбца к матрице преобразования. Используя эту матрицу в соотношении, получаем преобразованный вектор $[x^*\ y^*\ 1]$. Добавление третьего элемента к вектору положения и третьего столбца к матрице преобразования позволяет выполнить смещение вектора положения. Третий элемент здесь можно рассматривать как дополнительную координату

вектора положения. Итак, вектор положения [*x y* 1] при воздействии на него матрицы 3□3 становится вектором положения в общем случае вида [*X Y H]*. Представленное преобразование было выполнено так, что [X *Y H]* = [*x* y** 1].

Преобразование, имеющее место в трехмерном пространстве, в нашем случае ограничено плоскостью, поскольку *H* = 1. Если, однако, □*p*□

третий столбец □ □*q*□ □ матрицы преобразования *T* размера 3 x 3 отличен от

□□*s*□□

0, то в результате матричного преобразования получим [*x y* 1] =*[X Y H],* где *H* □ 1.

Плоскость, в которой теперь лежит преобразованный вектор положения, находится в трехмерном пространстве. Однако сейчас нас не интересует то, что происходит в трехмерном пространстве.

Итак, найденные *x** и *y** получены с помощью пучка лучей, проходящих через начало координат. Результат преобразований показан на рис. 6.3.

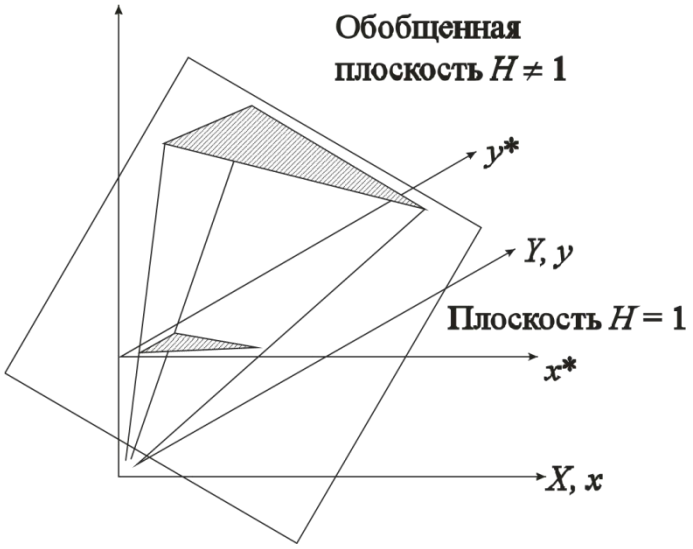


Рис. 6.3. Геометрическое представление однородных координат

Из рассмотрения подобных треугольников видно, что *H* / *X* □1/ *x** и *H* /*Y* □1/ *y**. Рассматривая три компоненты, запишем это в виде

□*x* y** 1□□□□□ — — *HX HY* 1□□□. Представление двумерного вектора трехмерным или в общем случае *n*-мерного вектора *(n* + 1)-мерным называют *однородным координатным воспроизведением*. При однородном координатном воспроизведении *n*-мерного вектора оно выполняется в *(n* + 1)-мерном

пространстве, и конечные результаты в n-мерном пространстве получают с помощью обратного преобразования. Таким образом, двумерный вектор [x y] представляется трехкомпонентным вектором $[hx\ hy\ h]$. Разделив компоненты вектора на однородную координату h, получим

$$\frac{hx}{h} \text{ и } y \quad \frac{hy}{h} . x$$

Не существует единственного однородного координатного представления точки в двумерном пространстве. Например, однородные координаты (12, 8, 4), (6, 4, 2) и (3, 2, 1) представляют исходную точку [3 2]. Для простоты вычислений выбираем [x y 1], чтобы представить непреобразованную точку в двумерных однородных координатах.

Преобразование $[x^* y^*] [x\ y] \begin{bmatrix} a & c \\ d & b \end{bmatrix}$

в дополнительных координатах задается выражением в однородных координатах в виде

$$[X\ Y\ H] [x\ y\ 1] \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} .$$

Выполнение указанных выше преобразований показывает, что $X = x^*$, $Y = y^*$, а $H = 1$. Равенство единице дополнительной координаты означает, что преобразованные однородные координаты равны исходным координатам.

В общем случае $H \neq 1$, и преобразованные обычные координаты получаются за счет нормализации однородных координат, т. е.

$$x^* = \frac{X}{H} \text{ и } y^* = \frac{Y}{H}.$$

Геометрически все преобразования x и y происходят в плоскости $H = 1$ после нормализации преобразованных однородных координат.

Преимущество введения однородных координат проявляется при использовании матрицы преобразований общего вида порядка 3×3

$$\begin{bmatrix} a & b & p \\ c & d & q \\ m & n & s \end{bmatrix},$$

с помощью которой можно выполнять и другие преобразования, такие как смещение, операции изменения масштаба и сдвига, обусловленные матричными элементами *a, b, c и d.* Указанные операции рассмотрены ранее.

Чтобы показать воздействие третьего столбца матрицы преобразований 3□3, рассмотрим следующую операцию:

□1 0 *p*□

□*X Y H*□□ □*x y* 1□□ □01 *q* □ □ □*x y* □*px*□ *qy* □1□□;

□□0 0 1□□

здесь *X = x, Y = y,* а *H = px + qy* + 1. Переменная *H,* которая определяет плоскость, содержащую преобразованные точки, представленные в однородных координатах, теперь образует уравнение плоскости в трехмерном пространстве.

Это преобразование показано на рис. 6.4, где линия *AB***,** лежащая в плоскости *xy,* спроектирована на линию *CD* плоскости *pX + qY —H* + 1 =

0.

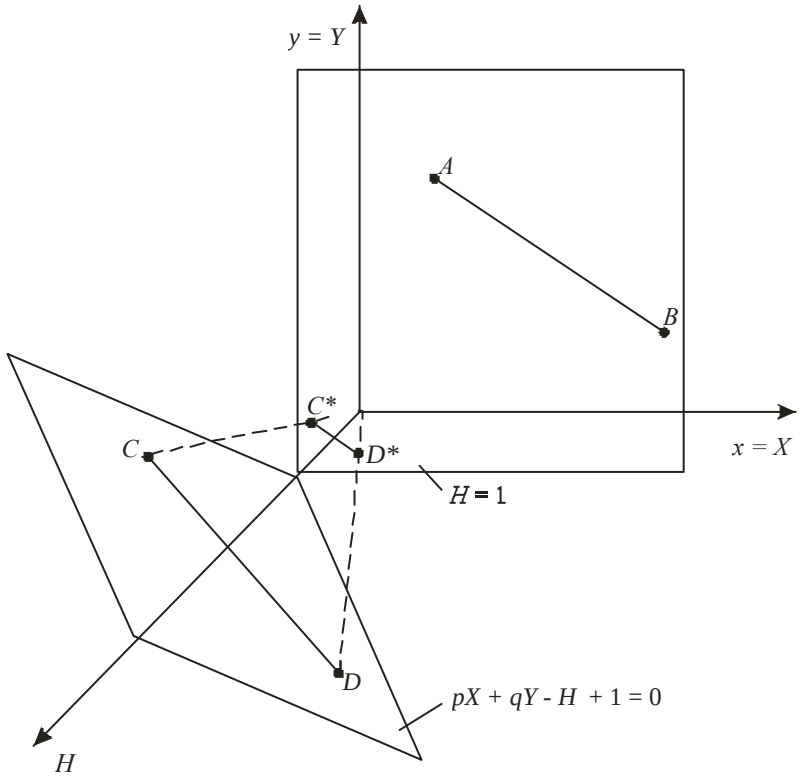


Рис. 6.4. Преобразование отрезка в однородных координатах На рисунке величина *p = q* = 1.Выполним нормализацию для того, чтобы получить обычные координаты:

$$x^* = \frac{X}{} - \frac{X}{\qquad},$$

$$y^* = \frac{H}{Y} \Big/ \frac{pX + qY + 1}{Y} = \frac{H}{pX + qY + 1}$$

Полагая $p = q = 1$, для изображенных на рисунке точек A и B с координатами соответственно (1, 3) и (4, 1) получим

$$x^* = \frac{1}{1+3+1} = \frac{1}{5} \text{ и } y^* = \frac{3}{5}.$$

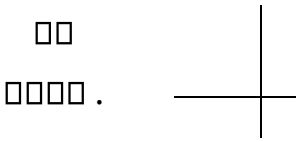
После преобразования A в C^* и B в D^* имеем

$$x^* = \frac{4}{1+4+1} = \frac{2}{3} \text{ и } y^* = \frac{1}{6}.$$

Однородные координаты для точек C^* и D^*, показанные на рисунке, соответственно равны $\left(\frac{1}{5}, \frac{3}{5}, 1\right)$ и $\left(\frac{2}{3}, \frac{1}{6}, 1\right)$.

Результатом нормализации является перевод трехмерной линии CD в ее проекцию C^*D^* на плоскость $H = 1$. Как показано на рисунке, центром проекции является начало координат.

Основная матрица преобразования размером 3×3 для двумерных однородных координат может быть подразделена на четыре части:



Как мы видим, *a, b, c* и *d* осуществляют изменение масштаба, сдвиг и вращение; *m* и *n* выполняют смещение, а *p* и *q* — получение проекций. Оставшаяся часть матрицы, элемент *s*, производит полное изменение масштаба. Чтобы показать это, рассмотрим преобразование

$$[X\ Y\ H] = [x\ y\ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix}$$

Здесь $X = x$, $Y = y$, а $H = s$. Это дает $x^* = x/s$ и $y^* == y/s$. В результате преобразования [*x y* 1] —> [*x/s y/s* 1] имеет место однородное изменение масштаба вектора положения. При $s < 1$ происходит увеличение, а при $s > 1$ — уменьшение масштаба.

6.1.2. Двумерное вращение вокруг произвольной оси

Выше было рассмотрено вращение изображения около начала координат. Однородные координаты обеспечивают поворот изображения вокруг точек, отличных от начала координат. В общем случае вращение около произвольной точки может быть выполнено путем переноса центра вращения в начало координат, поворотом относительно начала координат, а затем переносом точки вращения в исходное положение. Таким образом, поворот вектора положения [*x y* 1] около точки *(m, n)* на произвольный угол может быть выполнен с помощью преобразования

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} X & Y & H \end{bmatrix}.$$

Выполнив две операции умножения матриц, можно записать

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ m(1-\cos\theta)+n\sin\theta & n(1-\cos\theta)-m\sin\theta & 1 \end{bmatrix} = \begin{bmatrix} X & Y & H \end{bmatrix}$$

Предположим, что центр изображения имеет координаты (4, 3) и желательно повернуть изображение на 90° против часовой стрелки вокруг центральной его оси. Действие, выполненное с помощью матрицы

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ & 0 \\ -\sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

вызывает вращение вокруг начала координат, а не вокруг оси. Как сказано выше, необходимо вначале осуществить перенос изображения таким образом, чтобы желаемый центр вращения находился в начале координат. Это осуществляется с помощью матрицы переноса

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -3 & 1 \end{bmatrix}.$$

Затем следует применить матрицу вращения и, наконец, привести результаты к началу координат посредством обратной матрицы. Вся операция

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -3 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 7 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix} = \begin{bmatrix} X & Y & H \end{bmatrix}$$

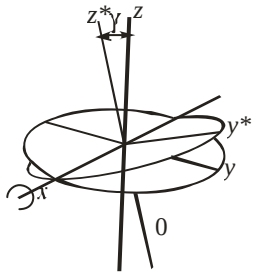
□□□4 □31□□ □□ 0 01□□ □□43 1□□

может быть объединена в одну матричную операцию путем выполнения матричных преобразований вида

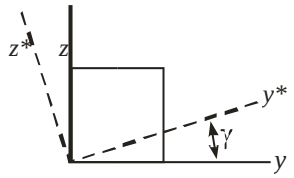
□ 0 1 0□

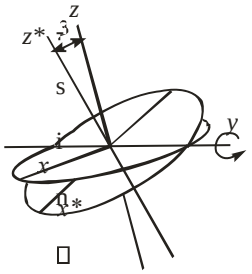
□*X Y H*□□ □*x y* 1□□ □ □1 0 0 □ .

□□ 7 □1 1□□ В результате будет получено *x* =X/H* и *y* = Y/H.* Двумерные вращения около каждой оси ортогональной системы представлены на рис. 6.5.



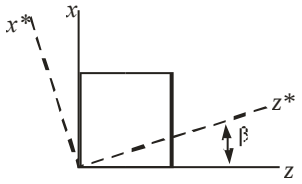
$$\begin{bmatrix} 0 & \cos\square & \sin\square \\ 1 & 0 & 0 \end{bmatrix}$$

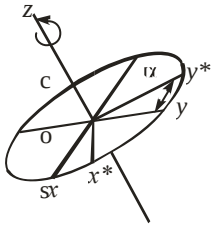




□

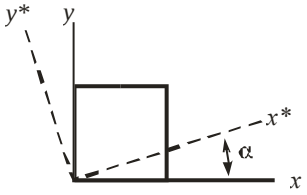
$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$





□ a

$$\begin{bmatrix} -\sin\square & \cos\square & 0 \end{bmatrix}$$



cos□ 0 sin□

-sin□ 0 cos□

б

cos□ sin□ 0

0 0 1

в

Рис. 6.5. Вращение: а — вокруг оси *x;* б — вокруг оси y; в — вокруг оси z

6.2. Трехмерные преобразования

Рассмотрим трехмерную декартовую систему координат, являющуюся ***правосторонней***. Примем соглашение, в соответствии с которым будем считать положительными такие повороты, при которых (если смотреть с конца полуоси в направлении начала координат) поворот на 90□ против часовой стрелки будет переводить одну полуось в другую. На основе этого соглашения строится следующая таблица, которую можно использовать как для правых, так и для левых систем координат:

Если ось вращения	Положительным будет направление поворота
X	От *y* к *z*
Y	От *z* к *x*
Z	От *x* к *y*

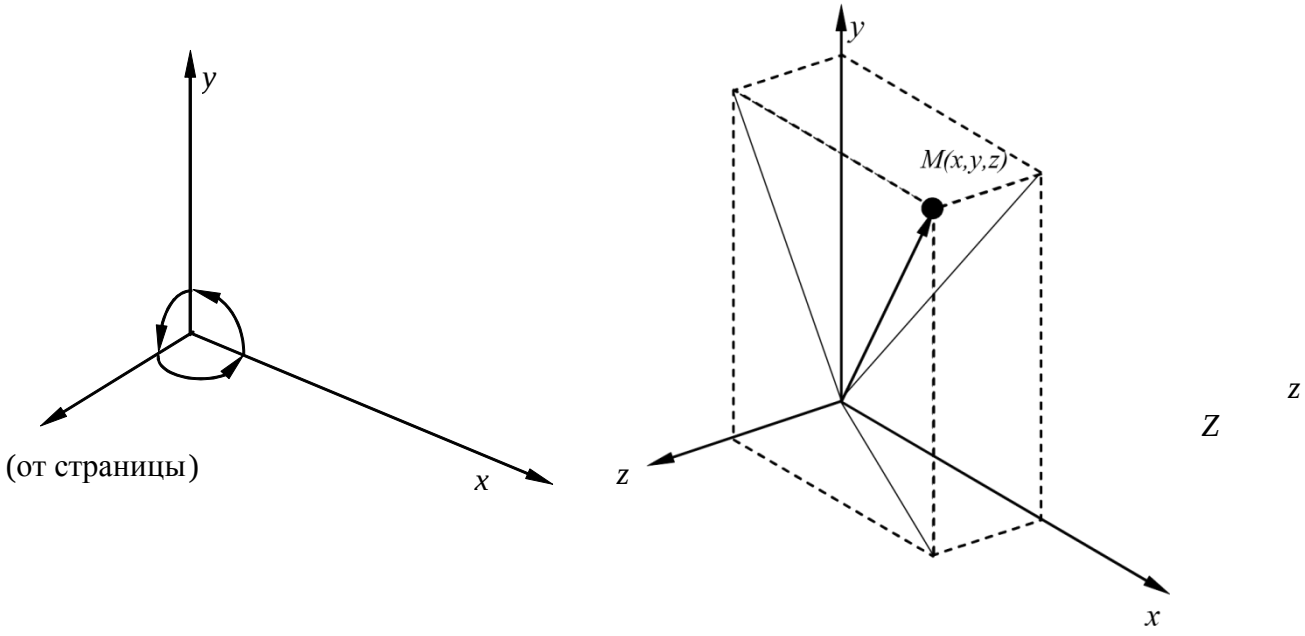


Рис. 6.6. Трехмерная система координат

Аналогично тому, как точка на плоскости описывается вектором (*x*,*y*), точка в трехмерном пространстве описывается вектором (*x*,*y*,*z*).

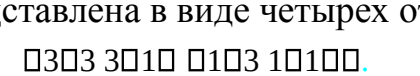
Как и в двухмерном случае, для возможности реализаций трехмерных преобразований с помощью матриц перейдем к однородным координатам:

[*x*,*y*,*x*,1] или [*X*,*Y*,*Z*,*H*]

[*x**,*y**,*z**1] = [$\frac{X}{H}, \frac{Y}{H}, \frac{Z}{H}$,1], где $H \neq 1$, $H \neq 0$.

Обобщенная матрица преобразования 4×4 для трехмерных однородных координат имеет вид

Эта матрица может быть представлена в виде четырех отдельных частей:



- Матрица 3×3 осуществляет линейное [1] преобразование в виде изменения масштаба, сдвига и вращения.
- Матрица 1×3 производит перенос.
- Матрица 3×1- преобразования в перспективе.
- Скалярный элемент 1×1 выполняет общее изменение масштаба.

Рассмотрим воздействие матрицы 4×4 на однородный вектор [*x*,*y*,*z*,1]:

1. Трехмерный перенос – является простым расширением двумерного:

$$T(Dx,Dy,Dz)=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Dx & Dy & Dz & 1 \end{bmatrix},$$

т. е. $[x,y,z,1]*T(Dx,Dy,Dz)=[x+Dx,y+Dy,z+Dz,1]$.

[1] ***Линейное преобразование*** трансформирует исходную линейную комбинацию векторов в некоторую линейную их комбинацию.

2. Трехмерное изменение масштаба

Рассмотрим **частичное** изменение масштаба. Оно реализуется следующим образом:

$$S(Sx,Sy,Sz,1) = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

т. е. $[x,y,z,1]*S(Sx,Sy,Sz)=[Sx*x,Sy*y,Sz*z,1]$.

Общее изменение масштаба получается за счет 4-го диагонального элемента, т. е.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & S \end{bmatrix} = [x\ y\ z\ S] = [x^*\ y^*\ z^*\ 1] = [X, Y, Z, 1].\ [x\ y\ z\ 1] * \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Такой же результат можно получить при равных коэффициентах частичных изменений масштабов. В этом случае матрица преобразования такова:

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

$$\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Трехмерный сдвиг

Недиагональные элементы матрицы 3□3 осуществляют сдвиг в трех измерениях, т. е.

$$[x\ y\ z\ 1]*\begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ h & i & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}=[x+yd+hz,\ bx+y+iz,\ cx+fy+z,\ 1].$$

4. Трехмерное вращение

Двухмерный поворот, рассмотренный ранее, является в то же время трехмерным поворотом вокруг оси Z . В трехмерном пространстве поворот вокруг оси Z описывается матрицей

$$R_z(\theta)=\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица поворота вокруг оси X имеет вид

$$R_x(\theta)=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица поворота вокруг оси Y имеет вид

$$R_y(\theta)=\begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$\begin{bmatrix} & & & \\ & & & \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Результатом произвольной последовательности поворотов вокруг осей *x, y, z* является матрица

$$A=\begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ h & i & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Подматрицу 3×3 называют ортогональной, так как ее столбцы являются взаимно ортогональными единичными векторами.

Матрицы поворота сохраняют длину и углы, а матрицы масштабирования и сдвига нет.

6.3. Проекции

В общем случае проекции преобразуют точки, заданные в системе координат размерностью *n*, в системы координат размерностью меньше чем *n*.

Будем рассматривать случай проецирования трех измерений в два. Проекция трехмерного объекта (представленного в виде совокупности точек) строится при помощи прямых проекционных лучей, которые называются ***проекторами*** и которые проходят через каждую точку объекта и, пересекая картинную плоскость, образуют ***проекцию***.

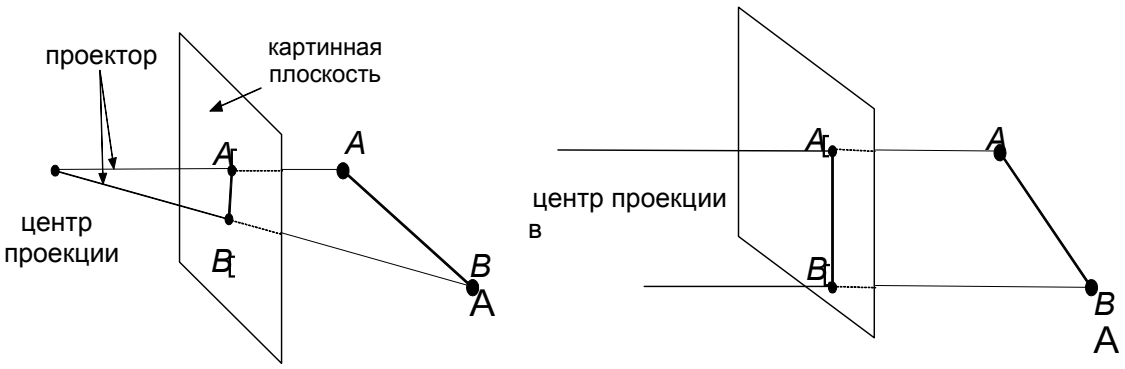


Рис. 6.7. Центральная и параллельная проекции

Определенный таким образом класс проекций существует под названием ***плоских геометрических проекций***, так как проецирование производится на плоскость, а не на искривленную поверхность и в качестве проекторов используются прямые, а не кривые линии.

Многие картографические проекции являются либо не плоскими, либо не геометрическими.

Плоские геометрические проекции в дальнейшем будем называть просто проекциями.

Проекции делятся на два основных класса (рис. 6.7): ▪ параллельные (аксонометрические); ▪ центральные (перспективные).

Полная классификация проекций приведена на рис. 6.8.

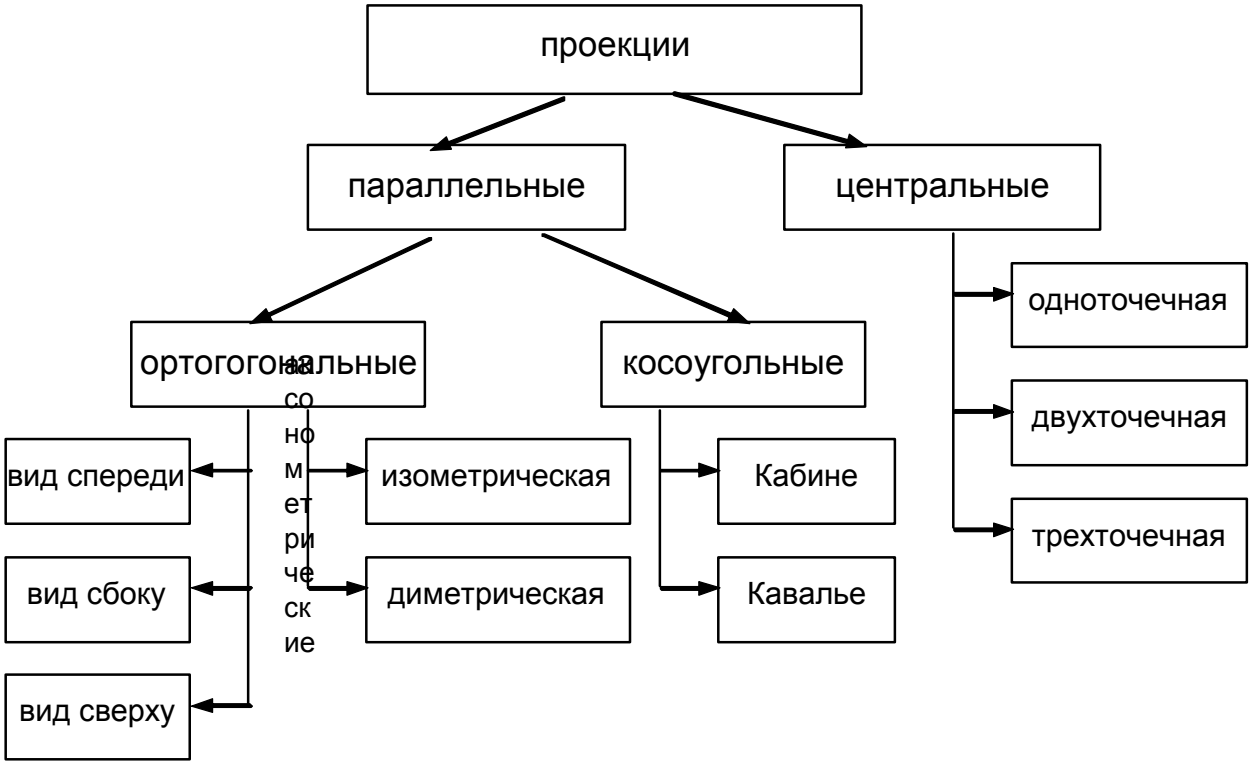


Рис. 6.8. Классификация проекций

Параллельные проекции делятся на два типа в зависимости от соотношения между направлением проецирования и нормалью к проекционной плоскости (рис. 6.9):

1) ***ортографические*** – направления совпадают, т. е. направление проецирования является нормалью к проекционной плоскости;

2) ***косоугольные*** – направление проецирования и нормаль к проекционной плоскости не совпадают.

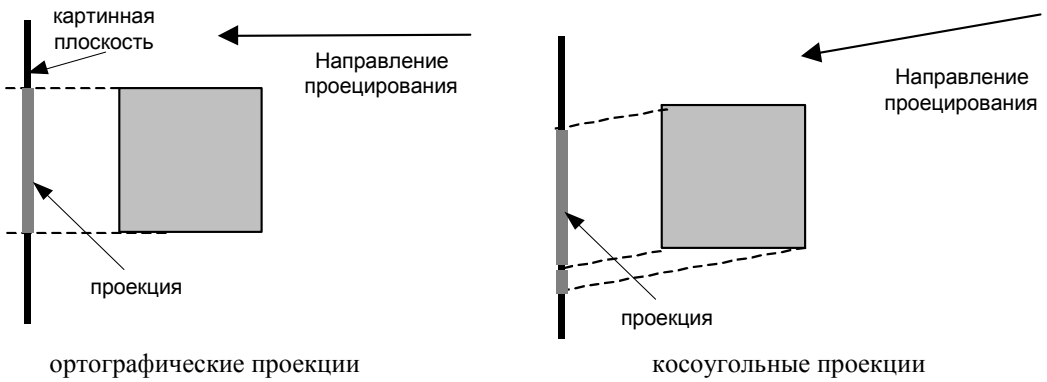


Рис. 6.9. Ортографические и косоугольные проекции

Наиболее широко используемыми видами ортографических проекций является вид спереди, вид сверху(план) и вид сбоку, в которых картинная плоскость перпендикулярна главным координатным осям. Если проекционные плоскости не перпендикулярны главным координатным осям, то такие проекции называются ***аксонометрическими***.

При аксонометрическом проецировании сохраняется параллельность прямых, а углы изменяются; расстояние можно измерить вдоль каждой из главных координатных осей (в общем случае с различными масштабными коэффициентами).

Изометрическая проекция – нормаль к проекционной плоскости, (а следовательно и направление проецирования) составляет равные углы с каждой из главных координатных осей. Если нормаль к проекционной плоскости имеет координаты (*a*,*b*,*c*), то потребуем, чтобы $|a| = |b| = |c|$, или $\square a = \square b = \square c$, т. е. имеется 8 направлений (по одному в каждом из октантов), которые удовлетворяют этому условию. Однако существует лишь 4 различных изометрических проекции (если не рассматривать удаление скрытых линий), так как векторы *(a, a, a)* и *(-a,a,-a)* определяют нормали к одной и той же проекционной плоскости.

Изометрическая проекция (рис. 6.10) обладает следующим свойством: все три главные координатные оси одинаково укорачиваются. Поэтому можно проводить измерения вдоль направления осей с одним и тем же масштабом. Кроме того, главные координатные оси проецируются так, что их проекции составляют равные углы друг с другом (120°).

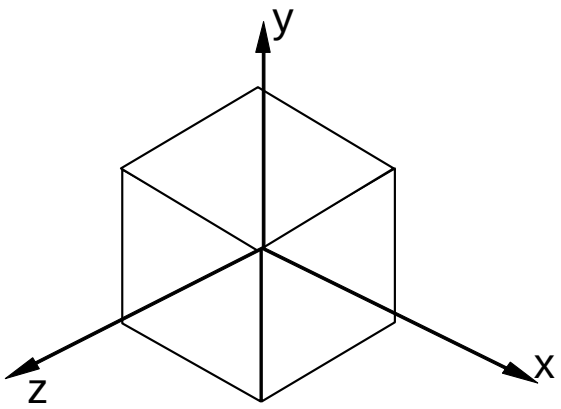


Рис. 6.10. Изометрическая проекция единичного куба

Косоугольные (наклонные) проекции сочетают в себе свойства ортографических проекций (видов спереди, сверху и сбоку) со свойствами аксонометрии. В этом случае проекционная плоскость перпендикулярна главной координатной оси, поэтому сторона объекта, параллельная этой плоскости, проецируется так, что можно измерить углы и расстояния. Проецирование других сторон объекта также допускает проведение линейных измерений (но не угловых) вдоль главных осей. Отметим, что нормаль к проекционной плоскости и направление проецирования не совпадают.

Двумя важными видами косоугольных проекций являются проекции:

- ***Кавалье*** (cavalier) – горизонтальная косоугольная изометрия (военная перспектива);
- ***Кабине*** (cabinet) – фронтальная косоугольная диметрия.

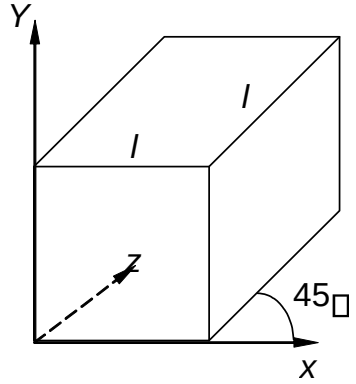


Рис. 6.11. Проекция Кавалье В ***проекции Кавалье*** (рис. 6.11) направление проецирования составляет с плоскостью угол 45□. В результате проекция отрезка, перпендикулярного проекционной плоскости, имеет ту же длину, что и сам отрезок, т. е. укорачивание отсутствует.

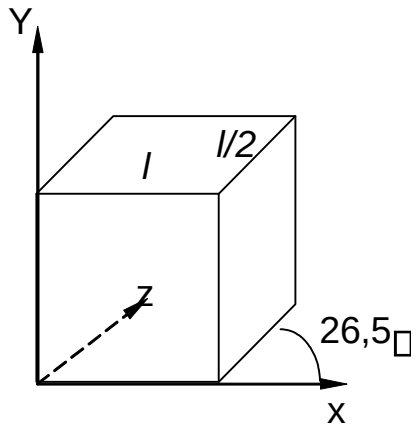


Рис. 6.12. Проекция Кабине

Проекция Кабине (рис. 6.12) имеет направление проецирования, которое составляет с проекционной плоскостью угол = arctg(½) (≈26,5°). При этом отрезки, перпендикулярные проекционной плоскости, после проецирования составляют ½ их действительной длины. Проекции Кабине являются более реалистическими, чем проекции Кавалье, так как укорачивание с коэффициентом ½ больше согласуется с нашим визуальным опытом.

Центральная проекция любой совокупности параллельных прямых, которые не параллельны проекционной плоскости, будет сходиться в точке схода. Точек схода бесконечно много. Если совокупность прямых параллельна одной из главных координатных осей, то их точка схода называется ***главной точкой схода***. Имеются только три такие точки, соответствующие пересечениям главных координатных осей с проекционной плоскостью. Центральные проекции классифицируются в зависимости от числа главных точек схода, которыми они обладают, а следовательно и от числа координатных осей, которые пересекают проекционную плоскость.

1. ***Одноточечная проекция*** (рис. 6.13).

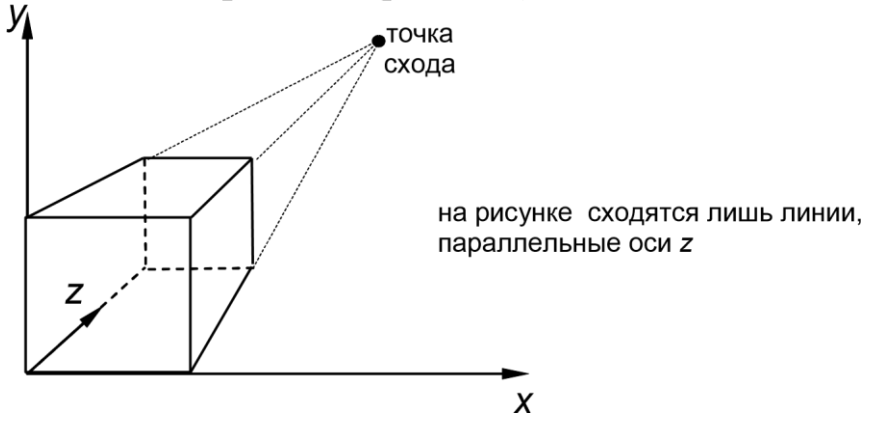


Рис. 6.13. Одноточечная перспектива

2. ***Двухточечная проекция*** (рис. 6.14) широко применяется в архитектурном, инженерном и промышленном проектировании.

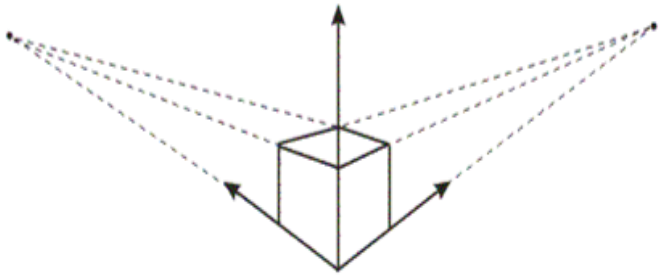


Рис. 6.14. Двухточечная перспектива

3. ***Трехточечные центральные проекции*** почти совсем не используются, во-первых, потому, что их трудно конструировать, а вовторых, из-за того, что они добавляют мало нового с точки зрения реалистичности по сравнению с двухточечной проекцией.

6.4. Математическое описание плоских геометрических проекций

Каждую из проекций можно описать матрицей 4□4. Этот способ оказывается удобным, поскольку появляется возможность объединить матрицу проецирования с матрицей преобразования.

Центральная (перспективная) проекция получается путем перспективного преобразования и проецирования на некоторую двухмерную плоскость «наблюдения». Перспективная проекция на плоскость $Z = 0$ обеспечивается преобразованием

$$[X\ Y\ Z\ H] = [x\ y\ z\ 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{bmatrix} = [x\ y\ 0\ (rz+1)].$$

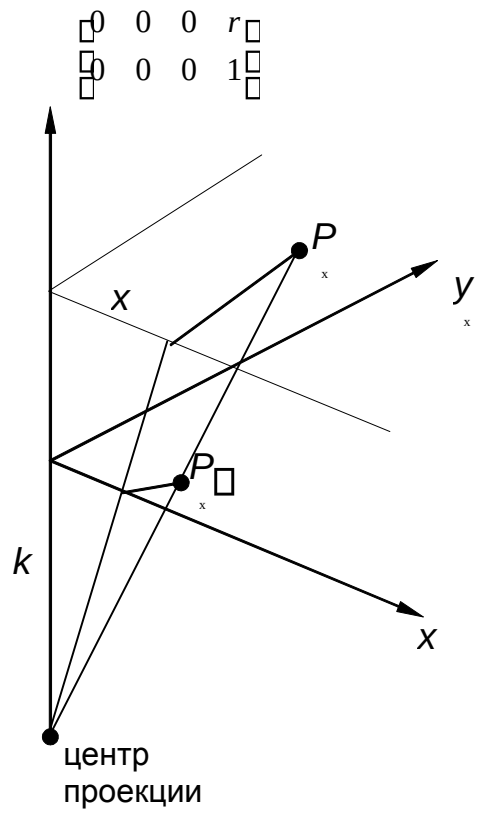


Рис. 6.15. Вычисление одноточечной перспективы

или $x^* = \frac{X}{H} = \frac{x}{rz+1}$;

$y^* = \frac{Y}{H} = \frac{y}{rz+1}$;

$z^* = \frac{Z}{H} = \frac{0}{rz+1}$,

где $r = -\frac{1}{k}$.

Центр проекции находится в точке с координатами *(*0,0,-*k)* (рис. 6.15), плоскость проецирования *Z = 0*. Соотношения между *x, y* и *x*, y** остаются теми же самыми. Рассматривая подобные треугольники, получим, что

$\frac{x^*}{-k} = \frac{x}{z-k}$, или $x^* = \frac{x}{z/k+1}$;

y аналогично $y^* = \frac{y}{z/k+1}$.

Координаты *x*, y** являются преобразованными координатами. В перспективном проектировании преобразованное пространство не является евклидовым, так как ортогональность осей не сохраняется. При $k = \infty$ получим аксонометрическое преобразование.

Аффинное преобразование есть комбинация линейных преобразований, сопровождаемых переносом.

Последний столбец в обобщенной матрице 4×4 должен быть равен:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

в этом случае $H = 1$.

Перспективному преобразованию может предшествовать произвольная последовательность аффинных преобразований. Таким образом, чтобы получить перспективные изображения из произвольной точки наблюдения вначале используют аффинные преобразования, позволяющие сформировать систему координат с осью *Z* вдоль желаемой линии визирования. Затем применяется перспективное преобразование.

Аналогично перспективное преобразование, когда картинная плоскость перпендикулярна оси *Z* и совпадает с плоскостью $Z = 1/r$.

Центр проекции находится в центре координат:

$$[X\ Y\ Z\ H] = [x\ y\ z\ 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x\ y\ z\ (rz+1)]$$ — одноточечная перспектива (точка схода *Z*);

$$\begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ — точка схода *X*.

Двухточечная (угловая) перспектива. Для получения двухточечной перспективы в общей матрице преобразования устанавливают коэффициенты p и q:

$$(x', y', z', 1) = (x, y, z, 1)\begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x, y, 0, (px+qu+1)];$$

$$(x', y', z', 1) = \left[\frac{px}{xqy}, \frac{px}{yqy}, 1, 0, 1\right].$$

Такое преобразование приводит к двум точкам схода. Одна расположена на оси X в точке $(-\frac{1}{p}, 0, 0, 1)$, другая на оси Y в точке $(0, -\frac{1}{q}, 0, 1)$.

Рассмотрим это преобразование на получение проекции единичного куба (рис. 6.16).

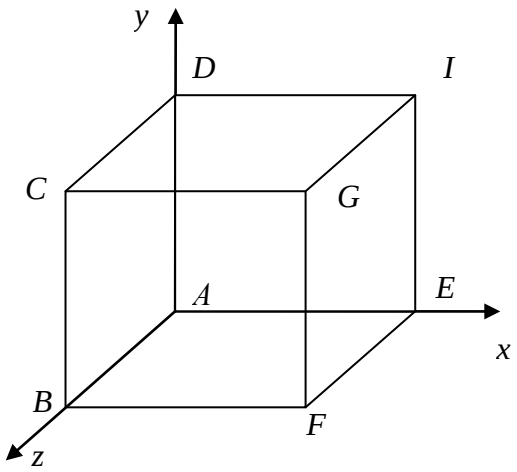


Рис. 6.16. Единичный куб для получения двухточечной проекции

В результате получаем проекцию вида, представленного на рис. 6.17.

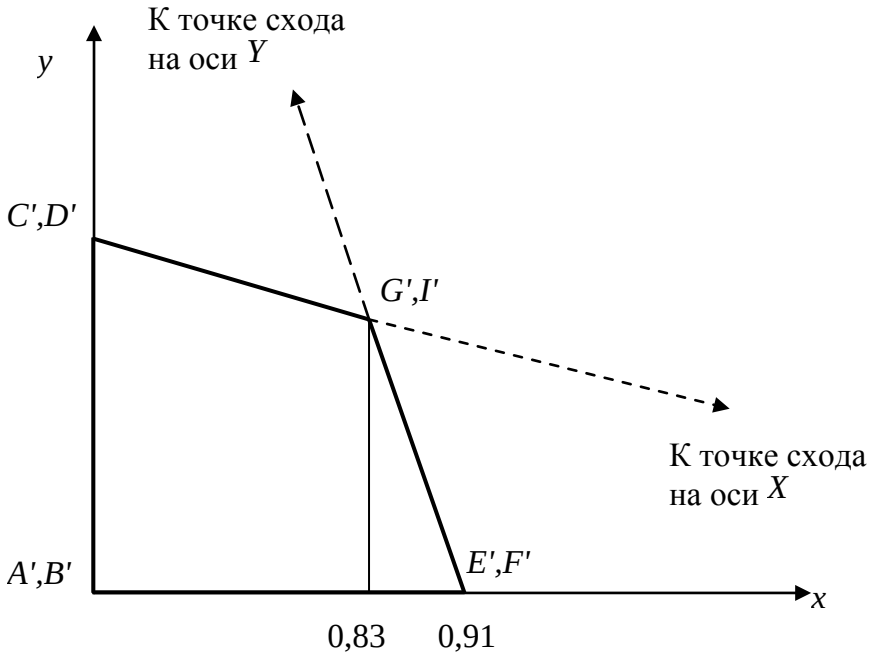


Рис. 6.17. Двухточечная проекция единичного куба

□ 0 1 0 *q* □

□ □ =[*x y z* (*px*+*qy*+*rz*+1)] — трехточечная (косая) □0 0 1 *r* □

□ □

□0 0 0 1□ перспектива.

Для того чтобы создать **диметрическую проекцию**, необходимо выполнить следующее условие:

$$\sin^2\varphi=\sin^2\theta/(1-\sin^2\theta).$$

Одним способом выбора sinθ является сокращение оси *Z* в фиксированное число раз. При этом единичный вектор на оси *Z*, равный

[0 0 1 1], преобразовывается к виду

[*X Y Z H*] = [sinφ -cosφ□sinθ cosφ□cosθ 1] или *x** = sinφ; y*= -cosφ sinθ.

Таким образом, для диметрической проекции получаем φ = 20,705□: θ = 22,208□.

Для образования **изометрической проекции** нужно в одинаковое число раз сократить все три оси. Для этого необходимо, чтобы выполнялось условие $\sin^2\varphi=\sin^2\theta/(1-\sin^2\theta)$ и $\sin^2\varphi=(1-2\sin^2\theta)/(1-\sin^2\theta)$. Таким образом, φ = 35,26439□; θ = 45□.

Рассмотрим теперь **косоугольную проекцию** (рис. 6.18), матрица может быть записана исходя из значений □ и *l*.

Проекцией точки *P*(0,0,1) является точка *P*□(*l* cos□, *l* sin□, 0), принадлежащая плоскости *xy*. Направление проецирования совпадает с отрезком *PP*□, проходящим через две эти точки. Это направление есть *P*□-*P* = (*l* cos□, *l* sin□, -1). Направление проецирования составляет угол □ с плоскостью *xy*.

Теперь рассмотрим проекцию точки *x, y, z* и определим ее косоугольную проекцию (x_p y_p) на плоскости *xy:*

$$x_p = x + z(l\cos\square);\ y_p = y + z(l\sin\square).$$

Таким образом, матрица 4□4, которая выполняет эти действия и, следовательно, описывает косоугольную проекцию, имеет вид

$$M_{\text{кос}}=\square\begin{matrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{matrix}\square .$$

$$\begin{pmatrix} 1 & 0 & l\cos\alpha & 0 \\ 0 & 1 & l\sin\alpha & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

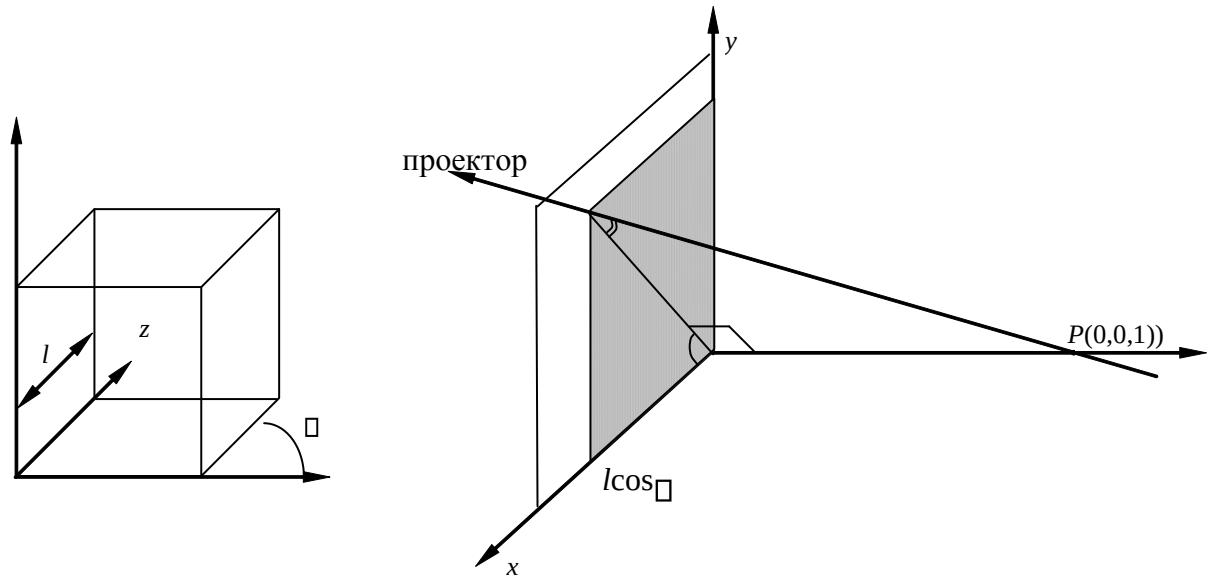


Рис. 6.18. Вычисление косоугольных проекций

Применение матрицы $M_{кос}$ приводит к сдвигу и последующему проецированию объекта: плоскости с постоянной координатой $z = z_1$ переносятся в направлении x на $z_1\ l\cos\alpha$ и в направлении y на $z_1\ l\sin\alpha$ и затем проецируется на плоскость $z = 0$. Сдвиг сохраняет параллельность прямых, а также углы и расстояния в плоскостях, параллельных оси z.

Для проекции Кавалье $l = 1$, поэтому угол $\alpha = 45°$. Для проекции Кабине l=½, а α = arctg(2) = 63,4°. В случае ортографической проекции l = 0 и $\alpha = 90°$, поэтому матрица ортографического проецирования является частным случаем косоугольной проекции.

6.5. Изображение трехмерных объектов

Процесс вывода трехмерной графической информации более сложный, чем соответствующий двумерный процесс. В двумерном случае просто задается окно в двумерном мировом координатном пространстве и поля вывода на двумерной видовой поверхности. В общем случае объекты, описанные в мировых координатах, отсекаются по границе видимого объема, а после этого преобразуются в поле вывода для дисплея. Сложность, характерная для трехмерного случая, возникает потому, что видовая поверхность не имеет третьего измерения.

Несоответствие между пространственными объектами и плоскими изображениями устраняется путем введения проекций, которые отображают трехмерные объекты на двумерной проекционной ***картинной плоскости*** (КП).

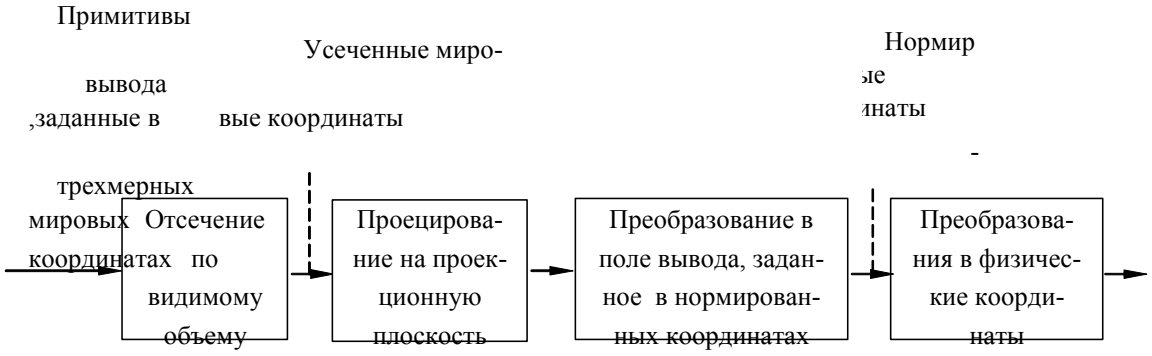


Рис. 6.19. Процесс вывода трехмерной графической информации

Процесс получения изображения из трехмерной модели называется ***визуализацией*** или ***рендерингом***. Кроме этапов, представленных на рис. 6.19, процесс рендеринга включает удаление невидимых граней и поверхностей, вычисление теней, полупрозрачных объектов, растеризацию объектов и закраску с учетом источников освещения, текстур и материалов.

6.5.1. Видимый объем

В процессе вывода трехмерной графической информации (рис. 6.19) мы задаем ***видимый объем*** (ВО) в мировом пространстве, проекцию на КП и поле вывода на видовой поверхности. В общем случае объекты, определенные в трехмерном мировом пространстве, отсекаются по границам трехмерного видимого объема и после этого проецируются. То, что попадает в пределы окна, которое само является проекцией видимого объема на картинную плоскость, затем преобразуется (отображается) в поле вывода и отображается на графическом устройстве.

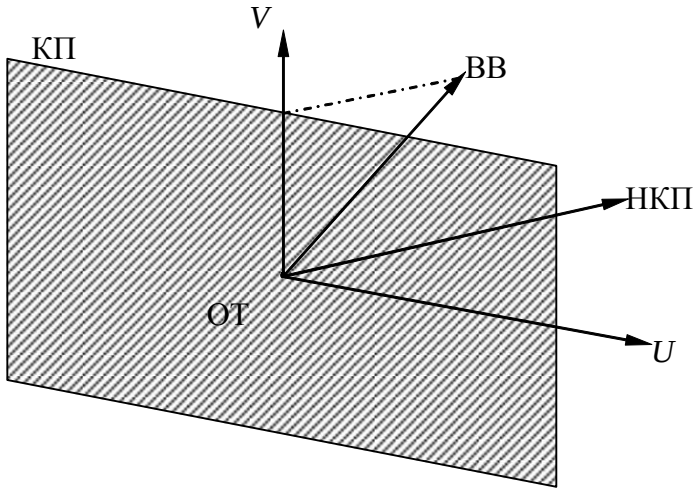


Рис. 6.20. Картинная плоскость и определяющие ее параметры

Картинная плоскость определяется (рис. 6.20) некоторой точкой на плоскости, которую будем называть ***опорной точкой*** (ОТ) и ***нормалью к картинной плоскости*** (НКП). КП может произвольным образом располагаться относительно проецируемых объектов, заданных в мировых координатах. Она может пересекать их, проходить впереди или позади объектов.

Для того чтобы задать окно, нам необходима система координат на КП, которую назовем системой координат *UV*. Началом ее служит ОТ. Направление оси *V* на КП определяет ***вектор вертикали*** (ВВ): проекция ВВ на КП совпадает с осью *V*.

ОТ и два направления вектора НКП и ВВ определяются в правосторонней мировой системе координат. Имея на КП систему *UV*, можем задать минимальное и максимальное значения *U* и *V*, определяющие окно (рис. 6.21).

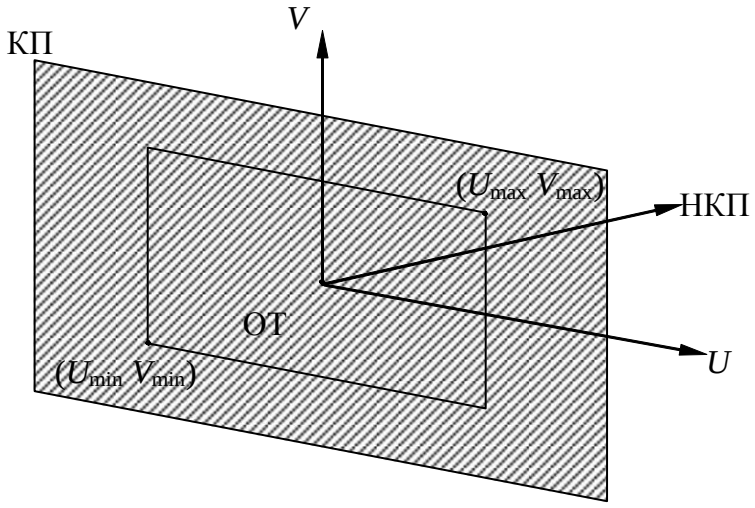


Рис. 6.21. Окно вывода на картинной плоскости

Отметим, что окно не обязательно должно быть симметрично относительно ОТ.

Видимый объем частично определяется окном и ограничивает ту часть мирового пространства, которая будет спроецирована.

В случае центральной проекции ВО определяется также центром проекции (рис. 6.22). Этот параметр задается в мировых координатах относительно ОТ. ВО представляет собой неограниченную в одну сторону пирамиду, вершина которой находится в центре проекции, а боковые стороны проходят через окно.

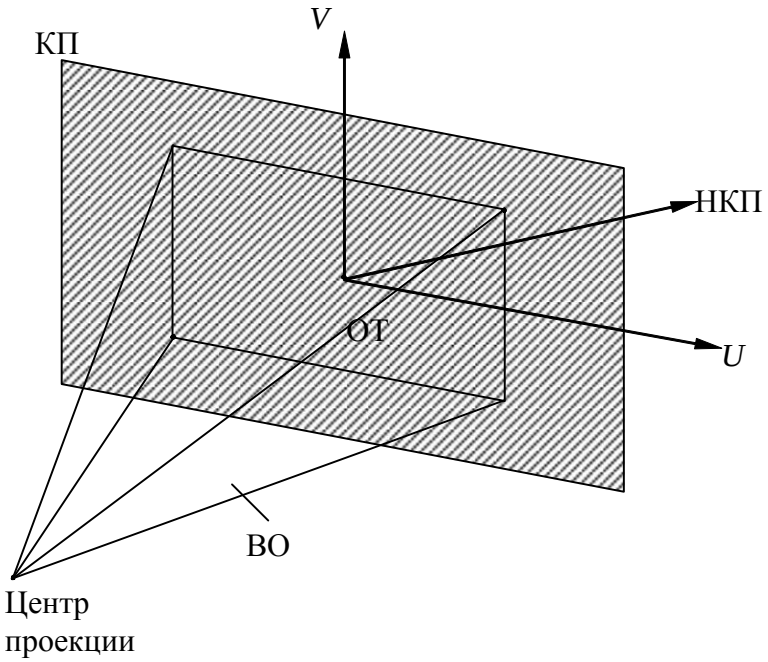


Рис. 6.22. Видимый объем для центральной проекции

Точки, лежащие позади центра проекции, не включаются в ВО и, следовательно, не будут проецироваться.

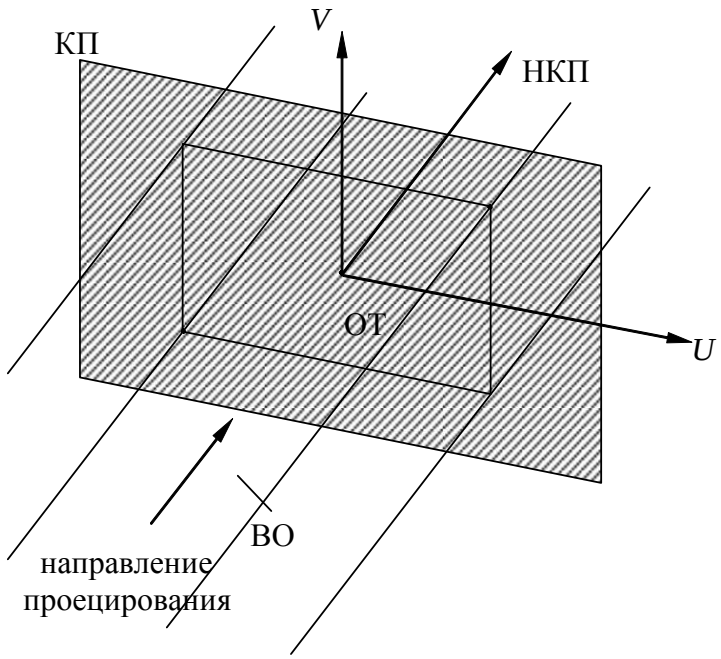


Рис. 6.23. Видимый объем параллельной проекции

В случае параллельных проекций ВО определяется также направлением проецирования (рис. 6.23). Он представляет собой неограниченный параллелепипед, стороны которого параллельны направлению проецирования.

В общем случае направление проецирования может не совпадать с НКП.

В случае ортографических параллельных проекций (но не косоугольных) боковые стороны ВО перпендикулярны КП.

В некоторых случаях может потребоваться сделать ВО конечным (рис. 3.24–3.26). Для этого задаются ПСП (передняя секущая плоскость) и ЗСП (задняя секущая плоскость).

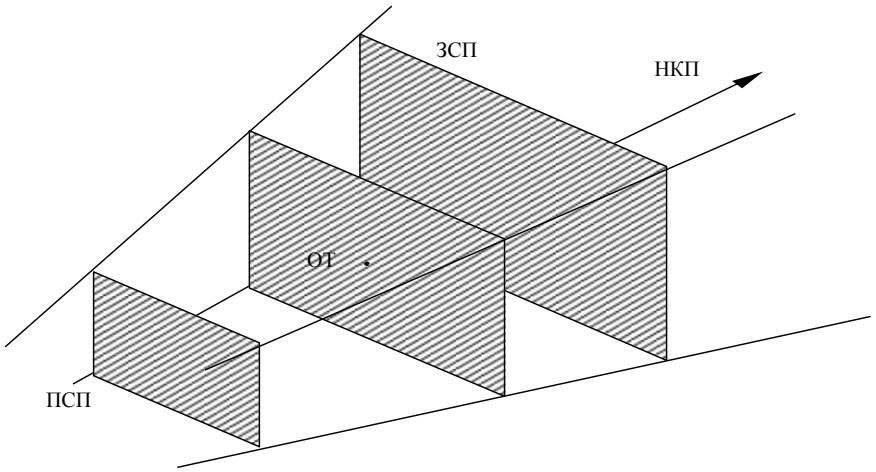


Рис. 6.24. Усеченный ВО для центральной проекции

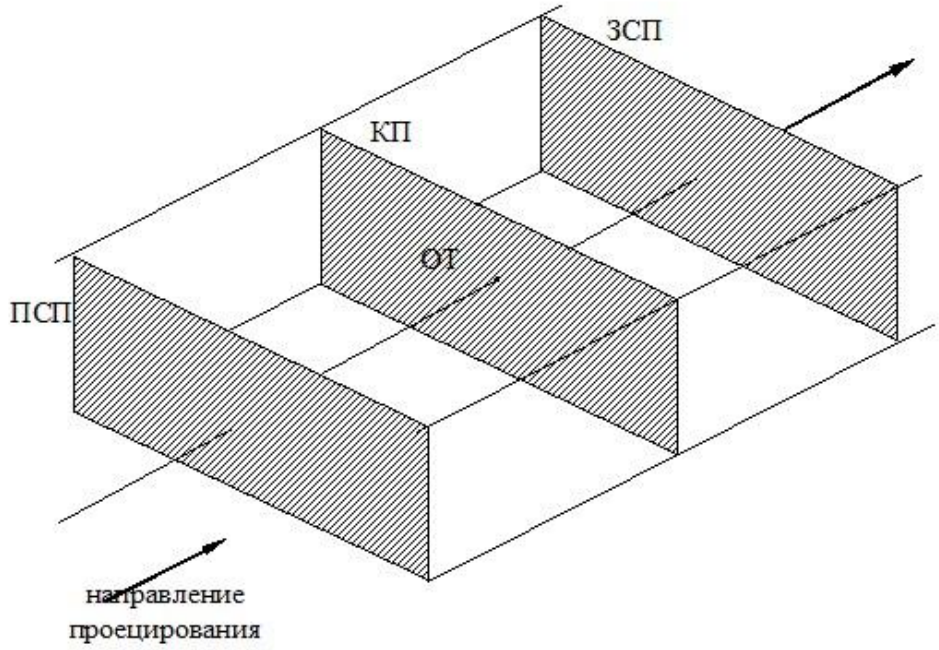


Рис. 6.25. Усеченный ВО для ортографической параллельной проекции

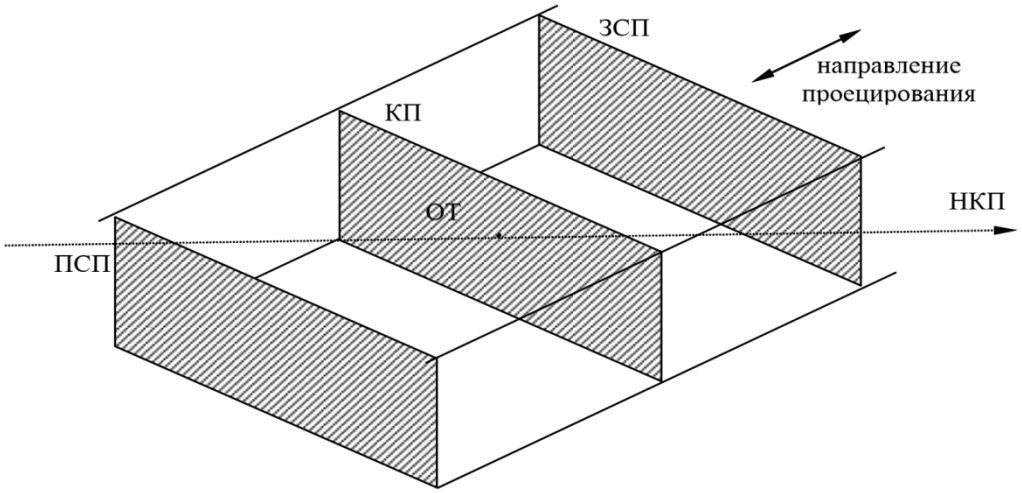


Рис. 6.26. Усеченный ВО для косоугольной параллельной проекции

Нормаль НКП направлена относительно направления проецирования и также является нормалью к ПСП и ЗСП.

6.5.2. Преобразование видимого объема

В случае центральной перспективы, для решения задачи отсечения по ВО требуются значительные вычисления. Решение заключается в преобразовании ВО к виду, в котором вычисления проводились бы значительно проще.

В общем, идея заключается в том, чтобы свести преобразование центральной перспективы математически к виду параллельной проекции.

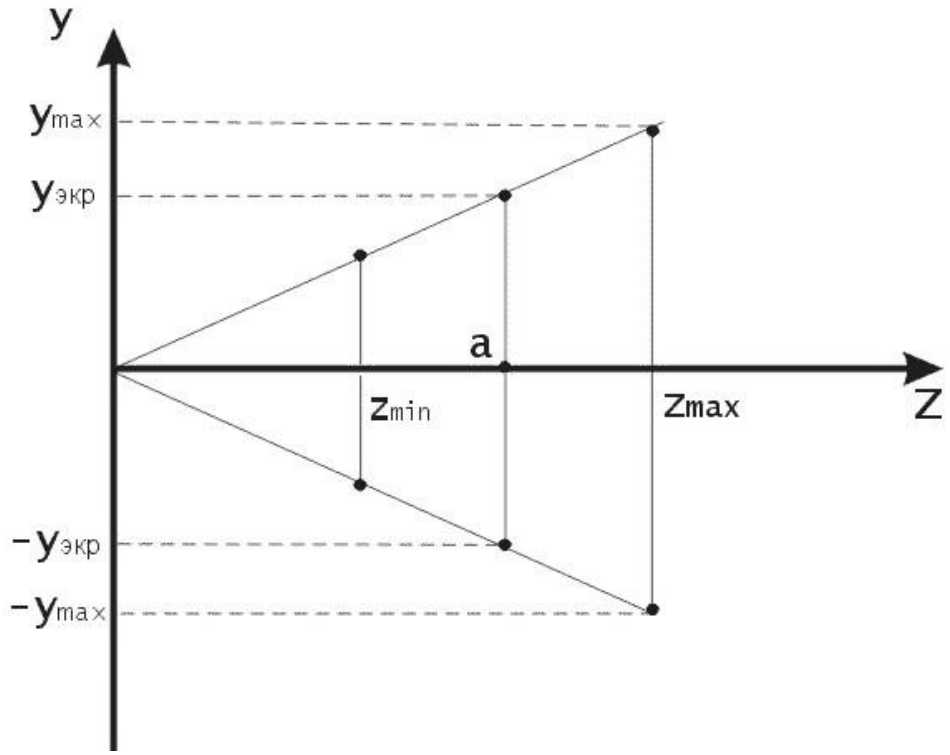
Будем решать задачу в два этапа. В начале приведем видимый объем к нормированному виду. При этом значение Z_{max}=1, а границы по осям *x* и *y* лежат в диапазоне [-1,1].

Нормирующим преобразованием в этом случае будет операция масштабирования, которая для произвольной точки *X* выражается в виде:

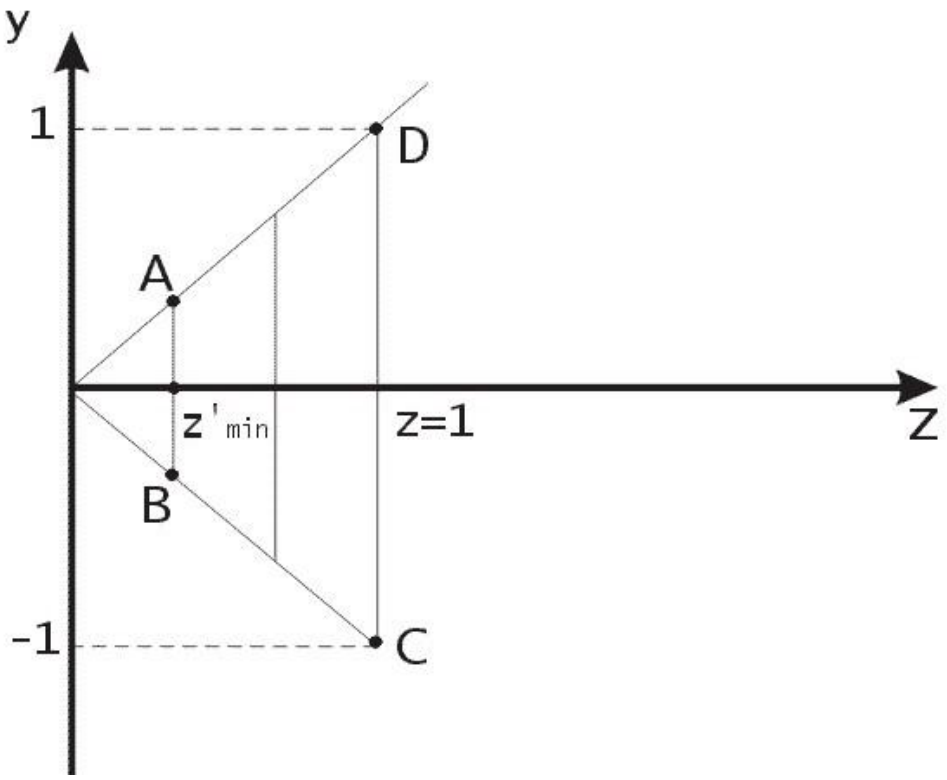
$$X = X S\left(\frac{1}{x_{max}}, \frac{1}{y_{max}}, \frac{1}{z_{max}}\right)$$

Графическая иллюстрация к нормализации ВО приведена на рис.

6.27.



а)



б)

Рис. 6.27. Нормализация ВО

Следующим этапом является переход от нормированного объема к каноническому виду, который позволит более быстро проводить не только отсечение по ВО, но и эффективно применять в дальнейшем алгоритмы удаления скрытых линий и поверхностей. Переход к каноническому виду производится с помощью матрицы преобразования M_p:

$$M_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1 - z_{min}} & 1 \\ 0 & 0 & -\frac{z_{min}}{1 - z_{min}} & 0 \end{pmatrix}$$

Графически иллюстрация, показывающая ВО в каноническом виде, приведена на рис. 6.28.

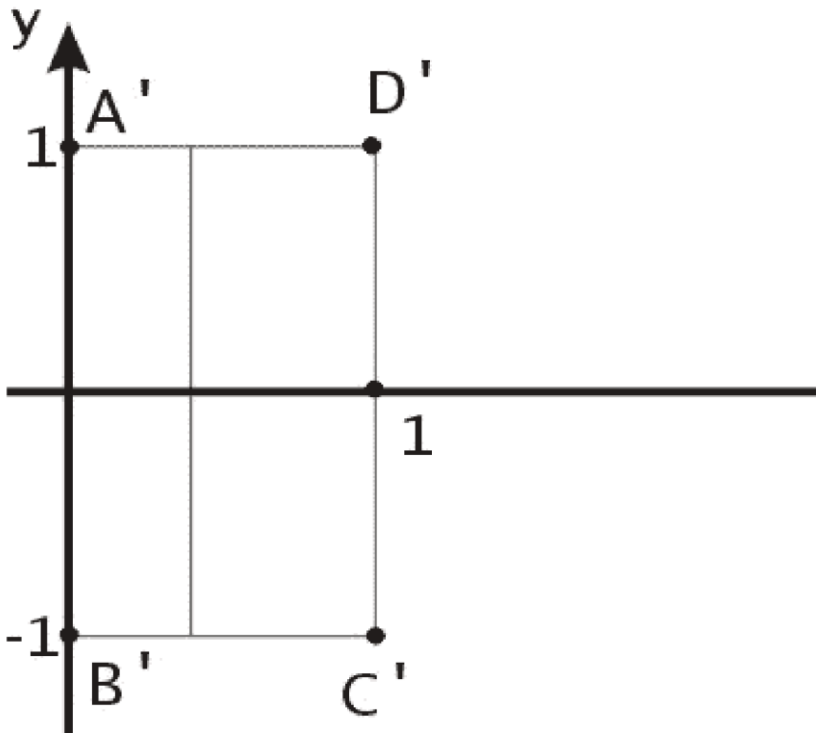


Рис. 6.28. Канонический вид ВО

7. Представление пространственных форм

Во многих приложениях машинной графики возникает потребность в представлении трехмерных форм: при проектировании самолетов, при восстановлении трехмерных тел по изображениям их поперечных сечений, построенных с помощью машинной томографии, при автоматической сборке и во многих других. Нам уже известно, как изображаются пространственные объекты, когда их удается представить в виде последовательности отрезков прямых, заданных в мировых координатах. Совокупность отрезков не является адекватным описанием объекта, поскольку отрезки сами по себе не определяют поверхностей. В то же время информация о поверхностях необходима для проведения вычислений, связанных со стиранием скрытых частей изображения, для определения объемов и т. д. Таким образом, мы приходим к выводу, что для описания трехмерных форм необходимы поверхности – примитивы более высокого уровня, чем отрезки.

Мы остановим внимание на двух широко распространенных трехмерных представлениях поверхностей в пространстве: полигональных сетках и параметрических бикубических кусках. ***Полигональной сеткой*** является

совокупность связанных между собой плоских многоугольников. Наружную форму большинства зданий можно легко и естественно описать с помощью полигональной сетки (так же, как мебель и комнаты). Полигональные сетки применяются также для представления объектов, ограниченных криволинейными поверхностями. Однако недостатком этого метода является его приблизительность. Видимые ошибки в таком представлении можно сделать сколь угодно малыми, используя все большее число многоугольников для улучшения кусочно-линейной аппроксимации объекта, но это приведет к дополнительным затратам памяти и вычислительного времени для алгоритмов, работающих с таким представлением.

При втором способе описания трехмерных форм поверхность может быть разбита на куски, каждый из которых будет описан параметрическим бикубическим уравнением. Поэтому такие поверхности называют ***параметрические бикубические поверхности***.

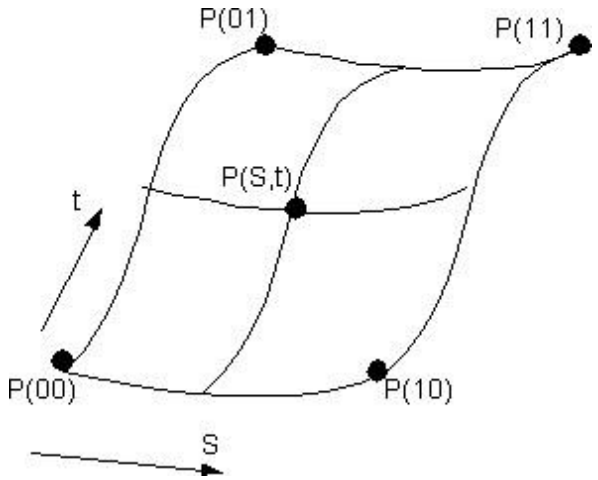


Рис. 7.1. Параметрический бикубический кусок

Параметрические бикубические куски (рис. 7.1), из которых состоят эти поверхности, описываются координатами точек с помощью трех уравнений (по одному для *x*, *y* и *z*). Каждое из уравнений имеет две переменные (два параметра), причем показатели степени при них не выше третьей (отсюда название бикубический). Уравнение для *X* будет выглядеть следующим образом:

$$X(S,t) = a_{11} S^3 t^3 + a_{12} S^3 t^2 + a_{13} S^3 t + a_{14} S^3 +$$

$$+ a_{21} S^2 t^3 + a_{22} S^2 t^2 + a_{23} S^2 t + a_{24} S^2 +$$

$$+ a_{31} S t^3 + a_{32} S t^2 + a_{33} S t + a_{34} S +$$

$$a\,t_{41}{}^{3} + a\,t_{42}{}^{2} + a\,t_{43} + a_{44}$$

В матричной форме уравнение для X можно описать как:

$X(S,t)=S\cdot C_x\cdot T^T$, где $S=[S^3\ \ S^2\ \ S\ \ 1]\quad T=[t^3\ \ t^2\ \ t\ \ 1]$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} C_x$$

Аналогично записываются уравнения для *Y* и *Z*.

Границами кусков являются параметрические кубические кривые. Для представления поверхности с заданной точностью требуется значительно меньшее число бикубических кусков, чем при аппроксимации полигональной сеткой. Однако алгоритмы для работы с бикубическими объектами существенно сложнее алгоритмов, имеющих дело с многоугольниками.

При использовании обоих методов трехмерное тело представляется в виде замкнутой поверхности.

7.1. Полигональные сетки

Полигональная сетка представляет собой совокупность ребер, вершин и многоугольников. Вершины соединяются ребрами, а многоугольники рассматриваются как последовательности ребер или вершин. Сетку можно представить несколькими различными способами, каждый из них имеет свои достоинства и недостатки. Для оценки оптимальности представления используют следующие критерии:

- Объем требуемой памяти;
- Простота идентификации ребер, инцидентных вершине;
- Простота идентификации многоугольников, которым принадлежит данное ребро;
- Простота процедуры поиска вершин, образующих ребро;
- Легкость определения всех ребер, образующих многоугольник;
- Простота получения изображения полигональной сетки;
- Простота обнаружения ошибок в представлении (например, отсутствие ребра или вершины или многоугольника).

Рассмотрим подробнее три способа описания полигональных сеток.

7.1.1. Явное задание многоугольников

Каждый многоугольник можно представить в виде списка координат его вершин:

$$P = ((x_1, y_1, z_1), (x_2, y_2, z_2), \ldots , (x_n, y_n, z_n))$$

Вершины запоминаются в том порядке, в котором они встречаются при обходе вокруг многоугольника. При этом все последовательные вершины многоугольника (а также первая и последняя) соединяются ребрами. Для каждого отдельного многоугольника данный способ записи является эффективным, однако для полигональной сетки дает потери памяти вследствие дублирования информации о координатах общих вершин.

Полигональная сетка изображается путем вычерчивания ребер каждого многоугольника, однако это приводит к тому, что общие ребра рисуются дважды – по одному разу для каждого из многоугольников.

7.1.2. Задание многоугольников с помощью указателей в список вершин

При использовании этого представления каждый узел полигональной сетки запоминается лишь один раз в списке вершин $V =((x_1, y_1, z_1), \ldots , (x_n, y_n, z_n))$. Многоугольник определяется списком указателей (или индексов) в списке вершин. Многоугольник, составленный из вершин 3, 5, 7 и 10 этого списка, представляется как P

$= (3, 5, 7, 10)$.

Такое представление имеет ряд преимуществ по сравнению с явным заданием многоугольников. Поскольку каждая вершина многоугольника запоминается только один раз, удается сэкономить значительный объем памяти. Кроме того, координаты вершины можно легко изменять. Однако все еще не просто отыскивать многоугольники с общими ребрами. Ребра при изображении всей полигональной фигуры по-прежнему рисуются дважды. Эти две проблемы можно решить, если описывать ребра в явном виде.

7.1.3. Явное задание ребер

В этом представлении имеется список вершин V, однако будем рассматривать теперь многоугольник как совокупность указателей на элементы списка ребер, в котором ребра встречаются лишь один раз. Каждое ребро в списке ребер указывает на две вершины в списке вершин, определяющие это ребро, а также на один или два многоугольника, которым это ребро принадлежит. Таким образом, мы описываем многоугольник как $P =(E_1, ..., E_2)$,

а ребро — как $E = (V_1, V_2, P_1, P_2)$. Если ребро принадлежит только одному многоугольнику, то либо P_1 либо P_2 – пусто.

При явном задании ребер полигональная сетка изображается путем вычерчивания не всех многоугольников, а всех ребер. В результате удается избежать многократного рисования общих ребер. Отдельные многоугольники при этом также изображаются довольно просто.

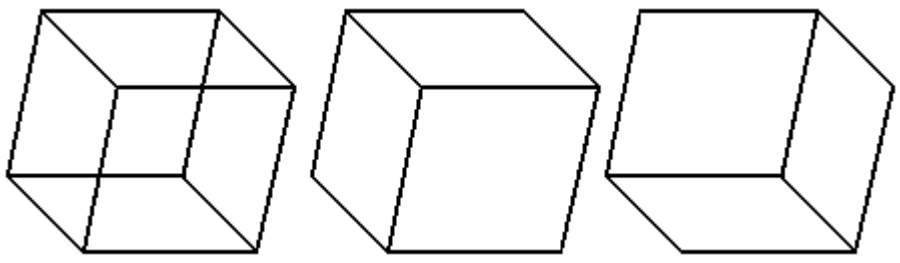
В некоторых приложениях ребра полигональных сеток являются общими для более чем двух многоугольников. Рассмотрим, например, случай в картографии, когда такие подразделения, как округа, штаты и т. д., описываются многоугольниками. Ребро (или последовательность ребер), представляющее часть границы между двумя штатами, является также границей округа в каждом штате, а возможно, и города. Таким образом, ребро может принадлежать одновременно шести многоугольникам. Если принять во внимание деление городов на районы, избирательные округа и школьные участки, то это число соответственно возрастет. Для таких приложений описания ребер могут быть расширены, чтобы включить произвольное число многоугольников: $E = (V_1, V_2, P_1, P_2, \ldots, P_n)$.

Ни в одном из этих представлений задача определения ребер, инцидентных вершине, не является простой: для ее решения необходимо перебрать все ребра. Конечно, для определения таких отношений можно непосредственно использовать дополнительную информацию.

8. Удаление невидимых линий и поверхностей

8.1. Классификация методов удаления невидимых линий и поверхностей

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства. Необходимость удаления невидимых линий, ребер, поверхностей или объемов проиллюстрирована рис.



а б в

Рис. 8.1. Необходимость удаления невидимых линий На рис. 8.1, а приведен типичный каркасный чертеж куба. Каркасный чертеж представляет трехмерный объект в виде штрихового изображения его ребер. Рис. 8.1, а можно интерпретировать двояко: как вид куба сверху слева или снизу справа. Для этого достаточно прищуриться и перефокусировать глаза. Удаление тех линий или поверхностей, которые невидимы с соответствующей точки зрения, позволяют избавиться от неоднозначности. Результаты показаны на рис. 8.1, б и рис. 8.1, в.

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Многие из них ориентированы на специализированные приложения. Наилучшего решения обшей задачи удаления невидимых линий и поверхностей не существует. Для моделирования процессов в реальном времени, например для авиатренажеров, требуются быстрые алгоритмы, которые могут порождать результаты с частотой видеогенерации 30 кадр/с. Для машинной мультипликации, например, требуются алгоритмы, которые могут генерировать сложные реалистические изображения, в которых представлены тени, прозрачность и фактура, учитывающие эффекты отражения и преломления цвета в мельчайших оттенках. Подобные алгоритмы работают медленно, и зачастую на вычисления требуется несколько минут или даже часов. Строго говоря, учет эффектов прозрачности, фактуры, отражения и т. п. не входит в задачу удаления невидимых линий или поверхностей. Естественнее считать их частью процесса визуализации изображения. Однако многие из этих эффектов встроены в алгоритмы удаления невидимых поверхностей. Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок для этих двух показателей одновременно. По мере создания все более быстрых алгоритмов можно строить все более детальные изображения. Реальные задачи, однако, всегда будут требовать учета еще большего количества деталей.

Все алгоритмы удаления невидимых линий (поверхностей) включают в себя сортировку. Порядок, в котором производится сортировка координат объектов, вообще говоря, не влияет на эффективность этих алгоритмов. Главная сортировка ведется по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения. Основная идея, положенная в основу сортировки по расстоянию, заключается в том, что чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен одним из объектов, более близких к

точке наблюдения. После определения расстояний или приоритетов по глубине остается провести сортировку по горизонтали и по вертикали, чтобы выяснить, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения. Эффективность любого алгоритма удаления невидимых линий или поверхностей в большой мере зависит от эффективности процесса сортировки. Для повышения эффективности сортировки используется также когерентность сцены, т. е. тенденция неизменяемости характеристик сцены в малом.

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают [7]. Выделяют три класса алгоритмов удаления невидимых линий или поверхностей:

- Алгоритмы, работающие в объектном пространстве.
- Алгоритмы, работающие в пространстве изображения (экрана).
- Алгоритмы, формирующие список приоритетов.

Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные лишь точностью вычислений. Полученные изображения можно свободно увеличивать во много раз. Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность.

Алгоритмы же, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана. Обычно разрешение экрана бывает довольно низким, типичный пример: 512□512 точек. Результаты, полученные в пространстве изображения, а затем увеличенные во много раз, не будут соответствовать исходной сцене. Например, могут не совпасть концы отрезков. Алгоритмы, формирующие список приоритетов, работают попеременно в обеих упомянутых системах координат.

Объем вычислений для любого алгоритма, работающего в объектном пространстве и сравнивающего каждый объект сцены со всеми остальными объектами этой сцены, растет теоретически, как квадрат числа объектов (n^2). Аналогично, объем вычислений любого алгоритма, работающего в пространстве изображения и сравнивающего каждый объект сцены с позициями всех пикселей в системе координат экрана, растет теоретически, как nN. Здесь n обозначает количество объектов (тел, плоскостей или ребер) в

сцене, а N — число пикселей. Теоретически трудоемкость алгоритмов, работающих в объектном пространстве, меньше трудоемкости алгоритмов, работающих в пространстве изображения, при $n < N$. Однако на практике это не так. Дело в том, что алгоритмы, работающие в пространстве изображения, более эффективны потому, что для них легче воспользоваться преимуществом когерентности при растровой реализации.

8.2. Алгоритм плавающего горизонта

Алгоритм плавающего горизонта можно отнести к классу алгоритмов, работающих в пространстве изображения. Алгоритм плавающего горизонта чаше всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде

$$F(x, y, z) = 0.$$

Подобные функции возникают во многих приложениях в математике, технике, естественных науках и других дисциплинах.

Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат *x*, *y* или *z*.

На рис. 8.2 приведен пример, где указанные параллельные плоскости определяются постоянными значениями *z*. Функция $F(x,y,z) = 0$ сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например к последовательности $y=f(x,z)$ или $x=g(y,z)$, где *z* постоянно на каждой из заданных параллельных плоскостей.

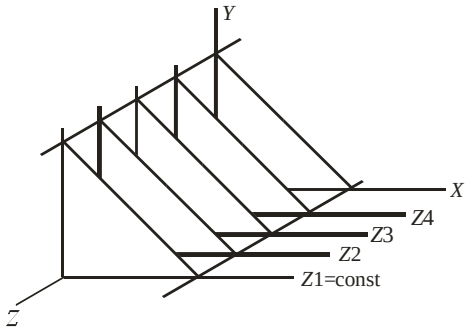


Рис. 8.2. Секущие плоскости с постоянной координатой

Итак, поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей, как показано на рис. 8.3.

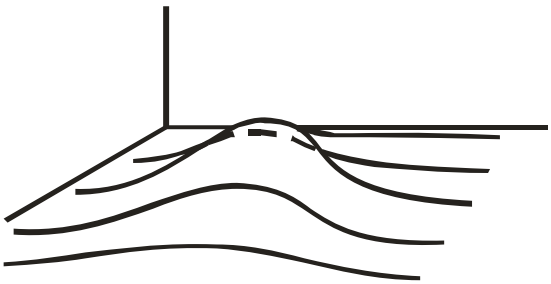


Рис. 8.3. Пространственные кривые

Здесь предполагается, что полученные кривые являются однозначными функциями независимых переменных. Если спроецировать полученные кривые на плоскость $z = 0$, как показано на рис. 8.4, то сразу становится ясна идея алгоритма удаления невидимых участков исходной поверхности.

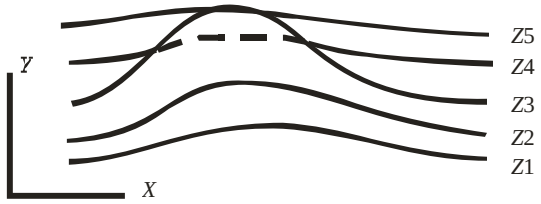


Рис. 8.4. Проекция кривых на плоскость z=0

Алгоритм сначала упорядочивает плоскости z = const по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т. е. для каждого значения координаты x в пространстве изображения определяется соответствующее значение y. Алгоритм удаления невидимой линии заключается в следующем.

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше значения y для всех предыдущих кривых при этом значении x, то текущая кривая видима в этой точке; в противном случае она невидима.

Невидимые участки показаны пунктиром на рис. 8.4. Реализация данного алгоритма достаточно проста. Для хранения максимальных значений y при каждом значении x используется массив, длина которого равна числу различимых точек (разрешению) по оси x в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения "горизонта". Поэтому по мере рисования каждой очередной кривой этот горизонт "всплывает". Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией.

Алгоритм работает очень хорошо до тех пор, пока какая-нибудь очередная кривая не окажется ниже самой первой из кривых, как показано на рис. 8.5, а.

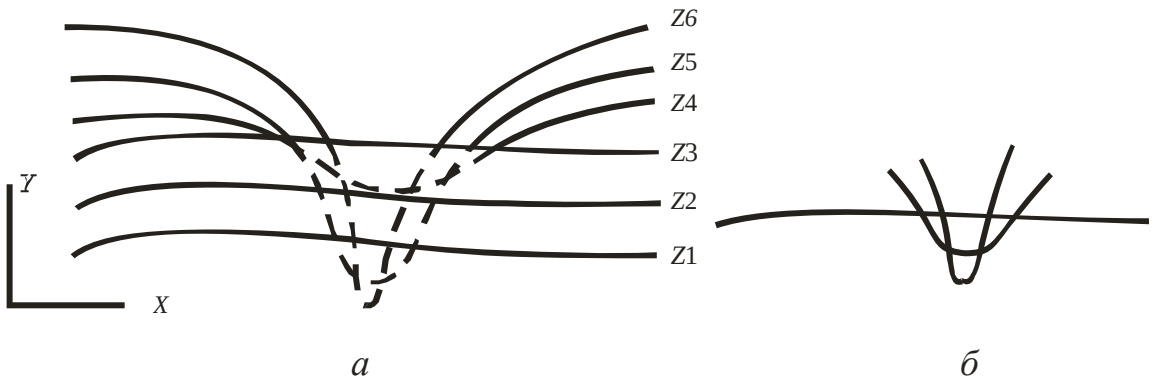


Рис. 8.5. Обработка нижней стороны поверхности

Подобные кривые, естественно, видимы и представляют собой нижнюю сторону исходной поверхности, однако алгоритм будет считать их невидимыми. Нижняя сторона поверхности делается видимой, если модифицировать этот алгоритм, включив в него нижний горизонт, который опускается вниз по ходу работы алгоритма. Это реализуется при помощи второго массива, длина которого равна числу различимых точек по оси *x* в пространстве изображения. Этот массив содержит наименьшие значения *y* для каждого значения *x*. Алгоритм теперь становится таким: если на текущей плоскости при некотором заданном значении *x* соответствующее значение *y* на кривой больше максимума или меньше минимума по *y* для всех предыдущих кривых при этом значении *x*, то текущая кривая видима. В противном случае она невидима.

Полученный результат показан на рис. 8.5, б.

В изложенном алгоритме предполагается, что значение функции, т. е. *y*, известно для каждого значения *x* в пространстве изображения. Однако если для каждого значения *x* нельзя указать (вычислить) соответствующее ему значение y, то невозможно поддерживать массивы верхнего и нижнего плавающих горизонтов. В таком случае используется линейная интерполяция значений *y* между известными значениями для того, чтобы заполнить массивы верхнего и нижнего плавающих горизонтов.

Изложенный алгоритм приводит к некоторым дефектам, когда кривая, лежащая в одной из более удаленных от точки наблюдения плоскостей, появляется слева или справа из-под множества кривых, лежащих в плоскостях, которые ближе к указанной точке наблюдения. Этот эффект

продемонстрирован на рис. 8.6, где уже обработанные плоскости *n*-1 и *n* расположены ближе к точке наблюдения. На рисунке показано, что получается при обработке плоскости *n*+1. После обработки кривых *n*-1 и n верхний горизонт для значений *x* = 0 и *x* = 1 равен начальному значению *y*; для значений *x* от 2 до 17 он равен ординатам кривой *n*; а для значений 18, 19, 20 - ординатам кривой *n*-1. Нижний горизонт для значений *x* = 0 и *x* = 1 равен начальному значению *y*; для значений *x* = 2, 3, 4 - ординатам кривой *n*; а для значений *x* от 5 до 20 - ординатам кривой *n*-1. При обработке текущей кривой (*n*+1) алгоритм объявляет ее видимой при *x* = 4. Это показано сплошной линией на рис. 8.6. Аналогичный эффект возникает и справа при *x* = 18. Такой эффект приводит к появлению зазубренных боковых ребер. Проблема с зазубренностью боковых ребер решается включением в массивы верхнего и нижнего горизонтов ординат, соответствующих штриховым линиям на рис. 8.6. Это можно выполнить эффективно, создав ложные боковые ребра.

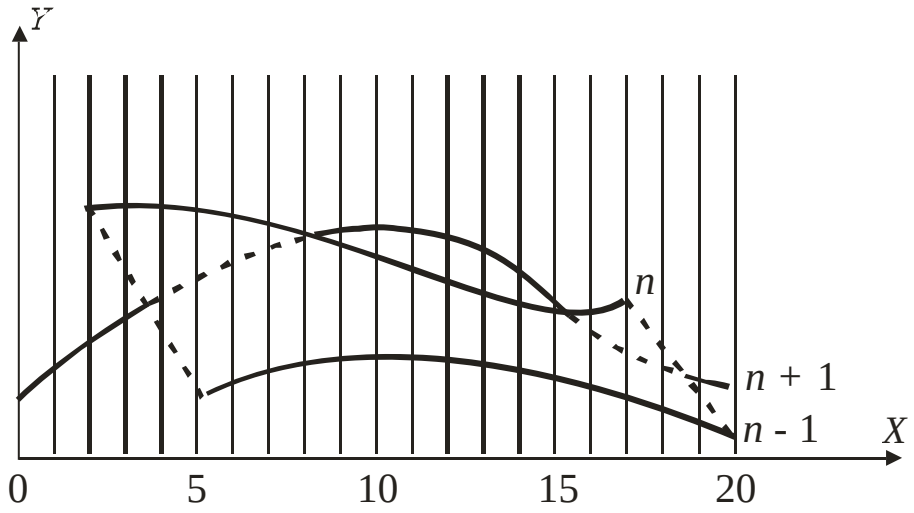


Рис. 8.6. Эффект зазубренного ребра

Если функция содержит очень острые участки (пики), то приведенный алгоритм также может дать некорректные результаты. Этот эффект вызван вычислением значений функции и оценкой ее видимости на участках, меньших, чем разрешающая способность экрана, т. е. тем, что функция задана слишком малым количеством точек. Если встречаются узкие участки, то функцию следует вычислять в большем числе точек.

На рис. 8.7 показан типичный результат работы алгоритма плавающего горизонта для функции $y = (1/5)\sin x \cos z - (3/2) \cos (7\alpha/4)\, e^{(-\alpha)}$, где $\alpha = (x - \square)^2+(z - \square)^2$, в интервале (0, 2□).

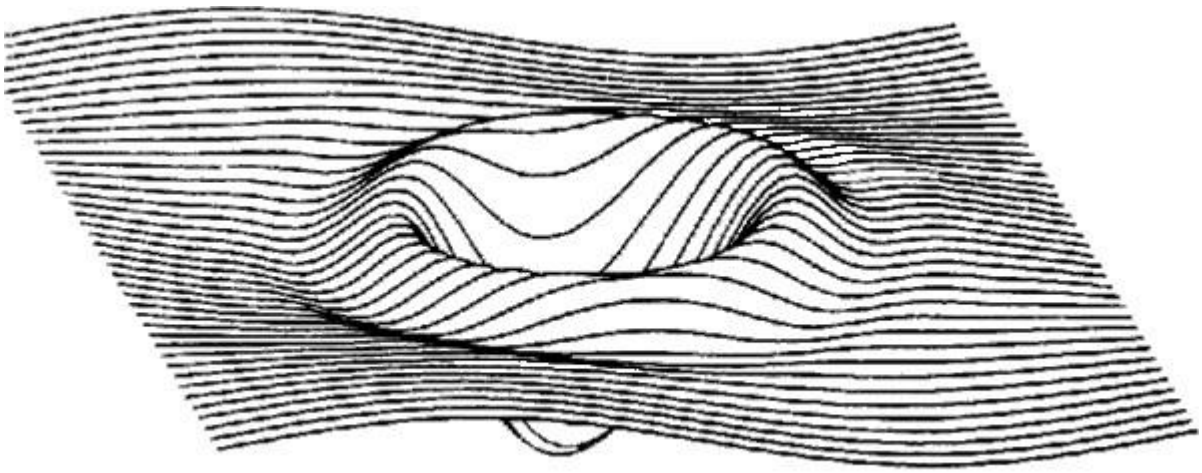


Рис. 8.7. Результат работы алгоритма плавающего горизонта

8.3. Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это математически элегантный метод, работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически, как квадрат числа объектов. Это в сочетании с ростом интереса к растровым дисплеям, работающим в пространстве изображения, привело к снижению интереса к алгоритму Робертса. Однако математические методы, используемые в этом алгоритме, просты, мощны и точны. Кроме того, этот алгоритм можно использовать для иллюстрации некоторых важных концепций. Наконец, более поздние реализации алгоритма, использующие предварительную приоритетную сортировку вдоль оси z и простые габаритные или минимаксные тесты, демонстрируют почти линейную зависимость от числа объектов.

Работа Алгоритм Робертса проходит в два этапа:

1. Определение нелицевых граней для каждого тела отдельно.
2. Определение и удаление невидимых ребер.

8.3.1. Определение нелицевых граней

Пусть F — некоторая грань многогранника. Плоскость, несущая эту грань, разделяет пространство на два подпространства. Назовем положительным то из них, в которое смотрит внешняя нормаль к грани. Если точка наблюдения – в положительном подпространстве, то грань – ***лицевая***, в

противном случае – ***нелицевая***. Если многогранник выпуклый, то удаление всех нелицевых граней полностью решает задачу визуализации с удалением невидимых граней.

Для определения, лежит ли точка в положительном подпространстве, используют проверку знака скалярного произведения (l, n), где l – вектор, направленный к наблюдателю, фактически определяет точку наблюдения; n – вектор внешней нормали грани. Если (l, n) > 0, т. е. угол между векторами острый, то грань является лицевой. Если (l, n) < 0, т. е. угол между векторами тупой, то грань является нелицевой.

В алгоритме Робертса требуется, чтобы все изображаемые тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые части. В этом алгоритме выпуклое многогранное тело с плоскими гранями должно представиться набором пересекающихся плоскостей. Уравнение произвольной плоскости в трехмерном

пространстве имеет вид

$$ax + by + cz + d = 0 \qquad (8.1.)$$

В матричной форме этот результат выглядит так:

$$[x\ y\ z\ 1][P]^T = 0,$$

где $[P]^T = [a\ \ b\ \ c\ \ d]$ представляет собой плоскость. Поэтому любое выпуклое твердое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей, т. е.

$$[V] = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{bmatrix},$$

где каждый столбец содержит коэффициенты одной плоскости.

Напомним, что любая точка пространства представима в однородных координатах вектором $[S] = [x\ y\ z\ 1]$. Более того, если точка $[S]$ лежит на плоскости, то $[S]\cdot[P]^T = 0$. Если же $[S]$ не лежит на плоскости, то знак этого скалярного произведения показывает, по какую сторону от плоскости расположена точка. В алгоритме Робертса предполагается, что точки, лежащие внутри тела, дают отрицательное скалярное произведение, т. е. нормали

направлены наружу. Чтобы проиллюстрировать эти идеи, рассмотрим следующий пример. Рассмотрим единичный куб с центром в начале координат (рис.

8.8).

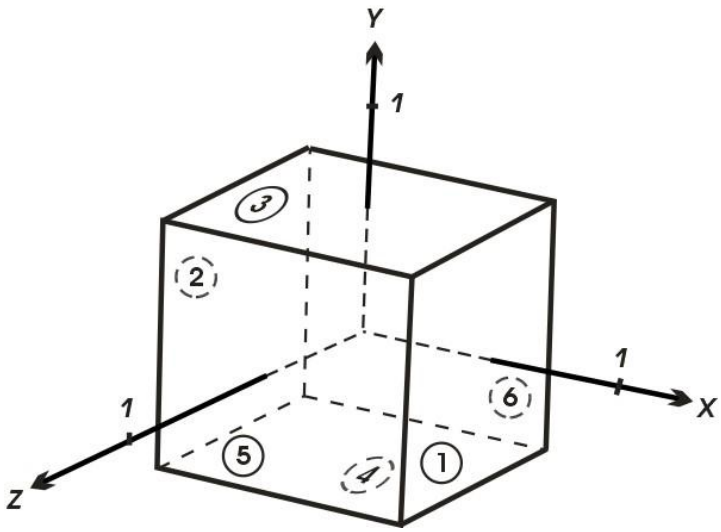


Рис. 8.8. Куб с центром в начале координат

Шесть плоскостей, описывающих данный куб, таковы: $x_1 = 1/2$, $x_2 = -1/2$, $y_3 = 1/2$, $y_4 = -1/2$, $z_5 = 1/2$, $z_6 = -1/2$. Более подробно уравнение правой плоскости можно записать как

$$x_1 + 0 \cdot y_1 + 0 \cdot z_1 - (1/2) = 0 \text{ или } 2x_1 - 1 = 0$$

Полная матрица тела такова:

$$[V] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1/2 & 1/2 & 1/2 & 1/2 & 1/2 & 1/2 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Экспериментально проверим матрицу тела с помощью точки, о которой точно известно, что она лежит внутри тела. Если знак скалярного произведения для какой-нибудь плоскости больше нуля, то соответствующее уравнение плоскости следует умножить на -1. Для проверки возьмем точку внутри куба с координатами x = 1/4, y = 1/4, z = 1/4. В однородных координатах эта точка представляется в виде вектора

$$[S] = [1/4 \;\; 1/4 \;\; 1/4 \;\; 1] = [1 \;\; 1 \;\; 1 \;\; 4].$$

Скалярное произведение этого вектора на матрицу объема равно

$$[S]\cdot[V] = [1\ \ 1\ \ 1\ \ 4]\cdot\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} =$$

$$= [-2\ \ 6\ \ -2\ \ 6\ \ -2\ \ 6].$$

Здесь результаты для второго, четвертого и шестого уравнения плоскостей (столбцов) положительны и, следовательно, составлены некорректно. Умножая эти уравнения (столбцы) на -1, получаем корректную матрицу тела для куба:

$$[V] = \begin{bmatrix} 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

В приведенном примере корректность уравнений плоскостей была проверена экспериментально. Разумеется, это не всегда возможно. Существует несколько полезных методов для более общего случая.

Хотя уравнение плоскости содержит четыре неизвестных коэффициента, его можно нормировать так, чтобы d = 1. Следовательно, трех неколлинеарных точек достаточно для определения этих коэффициентов. Подстановка координат трех неколлинеарных точек (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3) в нормированное уравнение (8.1.) дает

$$ax_1 + by_1 + cz_1 = -1;$$

$$ax_2 + by_2 + cz_3 = -1;$$

$$ax_3 + by_2 + cz_3 = -1.$$

В матричной форме это выглядит так:

□x_1 $y_1 z_1$□ □a□ □□1□

□□$x_2 y_2 z_2$□□□□□b□□ □ □□□1□□

□□$x_3 y_3 z_3$□□ □□c□□□□□1□□

или

$$[X][C] = [D] \qquad (8.2.)$$

Решение этого уравнения дает значения коэффициентов уравнения плоскости: $[C] = [X]^{-1}[D]$.

Другой способ используется, если известен вектор нормали к плоскости, т. е.

$$n = a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k},$$

где ***i, j, k*** – единичные векторы осей *x*, *y*, *z* соответственно. Тогда уравнение плоскости примет вид

$$ax + by + cz + d = 0 \qquad (8.3.)$$

Величина *d* вычисляется с помощью произвольной точки на плоскости. В частности, если компоненты этой точки на плоскости (x_1, y_1, z_1), то

$$d = -(ax_1 + by_1 + cz_1) \qquad (8.4.)$$

Перед началом работы алгоритма удаления невидимых линий или поверхностей для получения желаемого вида сцены часто применяется трехмерное видовое преобразование. Матрицы тел для объектов преобразованной сцены можно получить или преобразованием исходных матриц тел, или вычислением новых матриц тел, используя преобразованные вершины или точки.

Если [*B*] – матрица однородных координат, представляющая исходные вершины тела, а [*T*] – матрица размером 4□4 видового преобразования, то преобразованные вершины таковы:

$$[BT] = [B][T], \qquad (8.5.)$$

где [*BT*] – преобразованная матрица вершин. Использование уравнения (8.2.) позволяет получить уравнения исходных плоскостей, ограничивающих тело:

$$[B][V] = [D], \qquad (8.6.)$$

где [*V*] – матрица тела, а [*D*] в правой части – нулевая матрица. Аналогично уравнения преобразованных плоскостей задаются следующим образом:

$$[BT][VT] = [D], \qquad (8.7.)$$

где [*VT*] – преобразованная матрица тела. Приравнивая левые части уравнения (8.6.) и (8.7.), получаем

$$[BT][VT] = [B][V].$$

Подставляя уравнение (8.5.), сокращая на [*B*] и умножая слева на $[T]^{-1}$, имеем

$$[VT] = [T]^{-1}[V].$$

Итак, преобразованная матрица тела получается умножением исходной матрицы тела слева на обратную матрицу видового преобразования.

Тот факт, что плоскости имеют бесконечную протяженность и что скалярное произведение точки (вектора точки) на матрицу тела положительно, если точка лежит вне этого тела, позволяет предложить метод, в котором матрица тела используется для определения граней, которые экранируются самим этим телом.

Положительное скалярное произведение дает только такая плоскость (столбец) в матрице тела, относительно которой точка лежит снаружи, т. е. в положительном подпространстве. Проиллюстрируем это на примере уже рассмотренного единичного куба (рис. 8.9):

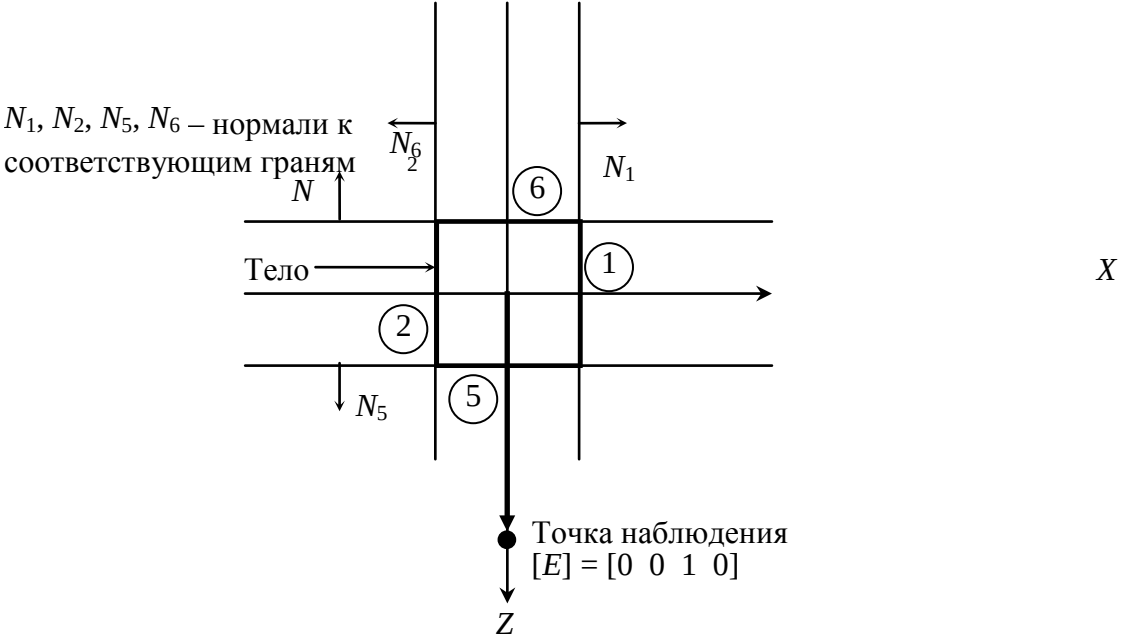


Рис. 8.9.

Точка наблюдения вне тела

Условие $[E]\cdot[V] < 0$ определяет, что плоскости — нелицевые, а их грани — задние. Заметим, что для аксонометрических проекций (точка наблюдения в бесконечности) это эквивалентно поиску положительных значений в третьей строке матрицы тела.

Найдем произведение $[E]\cdot[V]$ для нашего примера (рис. 8.9):

$$[E]\cdot[V] = [0\ \ 0\ \ 1\ \ 0] \begin{bmatrix} 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = [0\ \ 0\ \ 0\ \ 0\ \ 2\ \ -2].$$

Отрицательное число в шестом столбце показывает, что грань с этим номером нелицевая. Положительное число в пятом столбце показывает, что грань лицевая. Нулевые результаты соответствуют плоскостям, параллельным направлению взгляда.

Положительный результат вектора E и вектора нормали можно интерпретировать как острый угол между этими векторами, отрицательный результат – как тупой угол. Если угол между векторами острый, то грань является лицевой; если угол тупой, то грань – нелицевая.

Этот метод является простейшим алгоритмом удаления невидимых поверхностей для тел, представляющих собой одиночные выпуклые многогранники. Он также используется для удаления нелицевых или задних

граней из сцены перед применением одного из алгоритмов удаления невидимых линий, которые обсуждаются ниже. Этот способ часто называют ***отбрасыванием задних плоскостей***. Для выпуклых многогранников число граней при этом сокращается примерно наполовину. Метод эквивалентен вычислению нормали к поверхности для каждого отдельного многоугольника.

Данный метод определения нелицевых граней в результате формирует аксонометрическую проекцию на некую плоскость, расположенную бесконечно далеко от любой точки трехмерного пространства. Видовые преобразования, включая перспективное, производятся до определения нелицевых плоскостей. Когда видовое преобразование включает в себя перспективу, то нужно использовать полное перспективное преобразование одного трехмерного пространства в другое, а не перспективное проецирование на некоторую двумерную плоскость. Полное перспективное преобразование приводит к искажению трехмерного тела, которое затем проецируется на некую плоскость в бесконечности, когда нелицевые плоскости уже определены. Этот результат эквивалентен перспективному проецированию из некоторого центра на конечную плоскость проекции.

В литературе описаны и другие способы удаления невидимых граней. Так, в источнике [7] все нормали к граням тела направляются внутрь тела и используется вектор, направленный от наблюдателя к проекционной плоскости.

8.3.2. Удаление невидимых ребер

После первого этапа удаления нелицевых отрезков необходимо выяснить, существуют ли такие отрезки, которые экранируются другими телами в картинке или в сцене. Для этого каждый оставшийся отрезок или ребро нужно сравнить с другими телами сцены или картинки.

Возможны следующие случаи:

- Грань ребра не закрывает. Ребро остается в списке ребер.
- Грань полностью закрывает ребро. Ребро удаляется из списка рассматриваемых ребер.
- Грань частично закрывает ребро. В этом случае ребро разбивается на несколько частей, видимыми из которых являются не более двух. Само ребро удаляется из списка рассматриваемых ребер, но в список проверяемых ребер добавляются те его части, которые данной гранью не закрываются.

Для оптимизации используется приоритетная сортировка (zсортировка), а также, сравнения с прямоугольными объемлющими оболочками тел. Такой подход позволяет удалить целые группы или кластеры отрезков и тел. Например, если все тела в сцене упорядочены в некотором приоритетном

списке, использующем значения *z* ближайших вершин для представления расстояния до наблюдателя, то никакое тело из этого списка, у которого ближайшая вершина находится дальше от наблюдателя, чем самая удаленная из концевых точек ребра, не может закрывать это ребро. Более того, ни одно из оставшихся тел, прямоугольная оболочка которого расположена полностью справа, слева, над или под ребром, не может экранировать это ребро. Использование этих приемов значительно сокращает число тел, с которыми нужно сравнивать каждый отрезок или ребро. Рис. 8.10 иллюстрирует работу алгоритма.

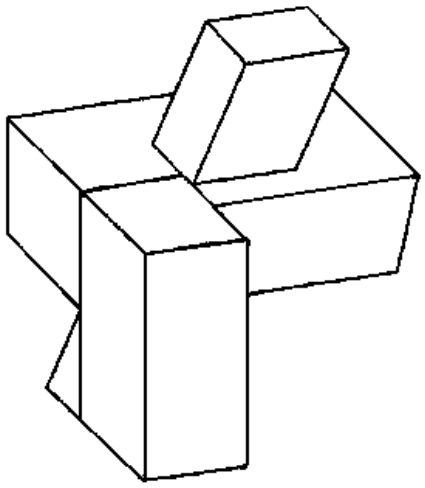


Рис. 8.10. Результат работы алгоритма Робертса

8.4. Алгоритм, использующий z–буфер

Алгоритм, использующий *z-буфер* это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Работает этот алгоритм в пространстве изображения. Идея *z*-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты *z* или глубины каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение *z* каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в *z*-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка *z*-буфера новым значением *z*. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по *x* и *y* наибольшего значения функции *z (x, y)*.

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 значений координат *z*, 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-ти 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 с 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 пиксели 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 пиксели могут заноситься в буфер кадра в некорректном порядке, что ведет к локальным ошибкам.

Формальное описание алгоритма *z*-буфера таково:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить *z*-буфер минимальным значением *z*.
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке.
4. Для каждого *Пиксель*(*x,y*) в многоугольнике вычислить его глубину *z*(*x,y*).
5. 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.

На псевдокоде алгоритм можно представить так:

```
for all objects
    for all covered pixels        compare z
```

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

Если известно уравнение плоскости, несущей каждый многоугольник, то вычисление глубины каждого пикселя на сканирующей строке можно проделать пошаговым способом. Грань при этом рисуется последовательно (строка за строкой). Для нахождения необходимых значений используется линейная интерполяция (рис.

8.11).

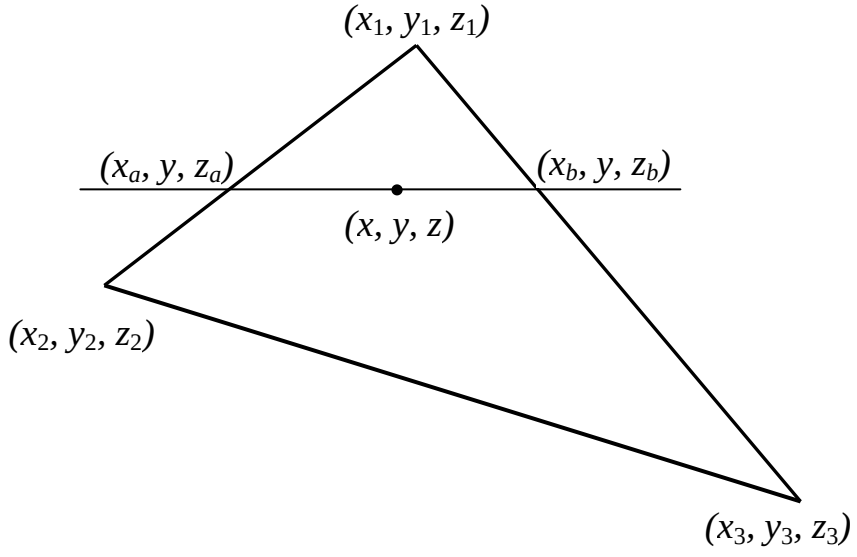


Рис. 8.11. Сканирующая строка по грани

Для рисунка y меняется от y_1 до y_2 и далее до y_3, при этом для каждой строки определяется x_a, z_a, x_b, z_b:

$$x_a = x_1 + (x_2 - x_1)\frac{y - y_1}{y_2 - y_1}; \quad y_1 \le y \le y_2$$

$$x_b = x_1 + (x_3 - x_1)\frac{y - y_1}{y_3 - y_1}; \quad y_1 \le y \le y_3$$

$$z_a = z_1 + (z_2 - z_1)\frac{y - y_1}{y_2 - y_1}; \quad y_1 \le y \le y_2$$

$$z_b = z_1 + (z_3 - z_1)\frac{y - y_1}{y_3 - y_1}; \quad y_1 \le y \le y_3$$

На сканирующей строке x меняется от x_a до x_b и для каждой точки строки определяется глубина z:

$$z = z_a + (z_b - z_a)\frac{x - x_a}{x_b - x_a}$$

Реализация алгоритма вдоль сканирующей строки позволяет совместить алгоритм z-буфера с алгоритмами растровой развертки ребер и алгоритмами закраски грани.

Проиллюстрируем работу алгоритма на примере для рис. 8.12.

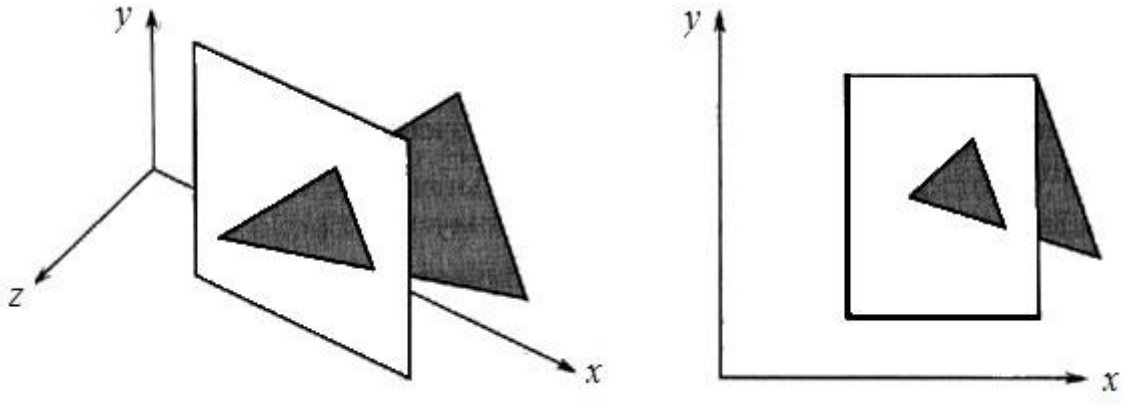


Рис. 8.12. Протыкающий треугольник

В начале в буфере кадра и в z-буфере содержатся нули. После растровой развертки прямоугольника содержимое буфера кадра будет иметь вид

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 Содержимое z-буфера таково:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

При обработке треугольника преобразование его в растровую форму и сравнение по глубине дает новое значение буфера кадра:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0
0 0 0 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 0
0 0 0 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 0
0 0 0 1 1 1 1 1 1 2 2 2 2 1 1 1 2 2 0
0 0 0 1 1 1 1 1 2 2 2 2 2 1 1 1 2 2 0
0 0 0 1 1 1 1 2 2 2 2 2 2 1 1 1 2 2 0
0 0 0 1 1 1 1 1 2 2 2 2 2 2 1 1 2 2 2
0 0 0 1 1 1 1 1 1 1 1 2 2 2 1 1 2 2 2
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 Новое содержимое z-буфера таково:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 5 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 5 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 6 0 0
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 6 0 0
0 0 0 10 10 10 10 10 10 10 11 10 10 10 10 6 5 0
0 0 0 10 10 10 10 10 10 10 12 11 10 10 10 7 5 0
0 0 0 10 10 10 10 10 10 13 12 11 11 10 10 7 6 0
0 0 0 10 10 10 10 10 14 13 12 12 11 10 10 7 6 0
0 0 0 10 10 10 10 15 14 13 12 12 11 10 10 7 6 0
0 0 0 10 10 10 10 10 14 13 12 12 11 11 10 8 7 5
0 0 0 10 10 10 10 10 10 10 12 11 11 10 10 8 7 5
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 8 7 6
0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 0 0 6
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

8.5. Методы трассировки лучей

В простейшем методе, использующим лучи, для каждого пикселя картинной плоскости определяется ближайшая к нему грань, для чего через этот пиксель выпускается луч, находятся все его пересечения с гранями и среди них выбирается ближайшая. Цвет этой грани и определяет цвет пикселя. Такой метод часто называют ***методом бросания лучей*** или ray casting.

Алгоритм на псевдокоде можно кратко записать так:

```
for all pixels
    for all objects                compare z
```

Алгоритм трассировки лучей несколько похож на алгоритм *z*буфера, однако здесь циклы по пикселям и по объектам меняются местами. В этом случае нельзя использовать когерентность пикселей, и линейную интерполяцию вдоль сканирующей строки. Однако проигрывая в скорости рендеринга, можно получить более качественные изображения использую модификацию описанного алгоритма.

В этом случае для удаления скрытых линий и поверхностей можно использовать тот факт, что в реальной природе источник света испускает луч света, который, «путешествуя» по пространству, в конечном счѐте «натыкается» на какую-либо преграду, которая прерывает распространение этого светового луча. В какой-либо точке пути с лучом света может случиться любая комбинация трѐх вещей: ***поглощение***, ***отражение*** (***рефлекция***) и ***преломление*** (***рефракция***).

Поверхность может отразить весь световой луч или только его часть в одном или нескольких направлениях. Поверхность может также поглотить часть светового луча, что приводит к потере интенсивности отраженного и/или преломлѐнного луча. Если поверхность имеет какиелибо свойства прозрачности, то она преломляет часть светового луча внутри себя и изменяет его направление распространения, поглощая некоторый (или весь) спектр луча (и, возможно, изменяя цвет). Суммарная интенсивность светового луча, которая была «потеряна» вследствие поглощения, преломления и отражения, должна быть в точности равной исходящей (начальной) интенсивности этого луча. Далее отраженные и/или преломлѐнные лучи достигают других поверхностей, где их поглощающие, отражающие и преломляющие способности снова вычисляются, основываясь на результатах вычислений входящих лучей. Таким образом, луч, пропущенный через пиксель картиной плоскости, образует сложный разделяющийся путь.

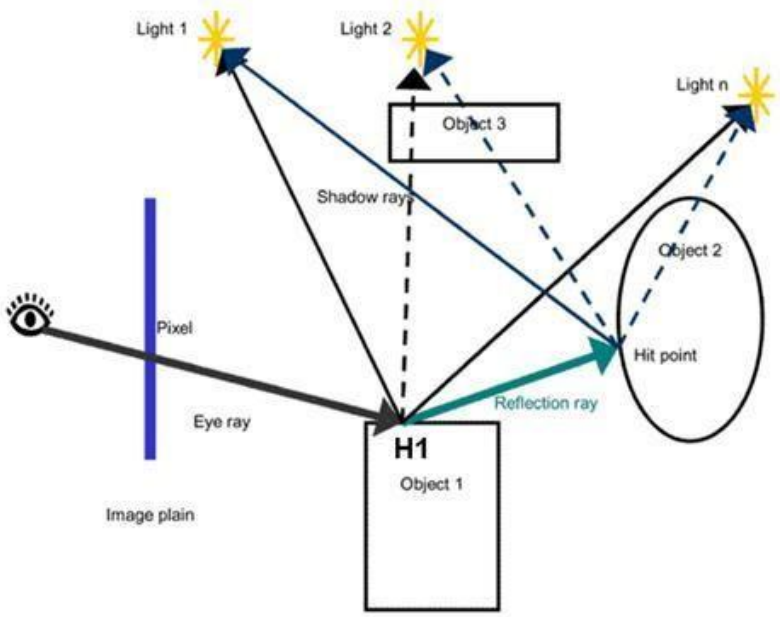


Рис. 8.13. Обратная трассировка лучей

Как отмечено в [7], если проследить за лучами света, выпущенным источником света, то можно убедиться, что весьма немногие дойдут до наблюдателя. Следовательно, этот процесс вычислительно неэффективен. Поэтому было предложено отслеживать (трассировать) лучи в *обратном* направлении, т.е. от наблюдателя к объекту и далее к источнику освещения (рис. 8.13). При этом, интенсивность в пикселе, через который проходит луч, рассчитывалась бы интегрально с учетом всех слагаемых: пересечения, отражения, преломления луча с разными объектами и характеристик источников света. Такой подход получил название ***метод обратной трассировки лучей*** (ray tracing).

8.6. Алгоритмы, использующие список приоритетов

При реализации всех обсуждавшихся алгоритмов удаления невидимых линий и поверхностей устанавливались приоритеты, т. е. глубины объектов сцены или их расстояния от точки наблюдения. Алгоритмы, использующие список приоритетов, пытаются получить преимущество посредством предварительной сортировки по глубине или приоритету. Цель такой сортировки состоит в том, чтобы получить окончательный список элементов сцены, упорядоченных по приоритету глубины, основанному на расстоянии от точки наблюдения. Если такой список окончателен, то никакие два элемента не будут взаимно перекрывать друг друга. Тогда можно записать все элементы в буфер кадра поочередно, начиная с элемента, наиболее удаленного от точки наблюдения. Более близкие к наблюдателю элементы будут затирать

информацию о более далеких элементах в буфере кадра. Поэтому задача об удалении невидимых поверхностей решается тривиально. Эффекты прозрачности можно включить в состав алгоритма путем не полной, а частичной корректировки содержимого буфера кадра с учетом атрибутов прозрачных элементов.

Для простых элементов сцены, например для многоугольников, этот метод иногда называют алгоритмом ***художника***, поскольку он аналогичен тому способу, которым художник создает картину. Сначала художник рисует фон, затем предметы, лежащие на среднем расстоянии, и, наконец, передний план. Тем самым художник решает задачу об удалении невидимых поверхностей, или задачу видимости, путем построения картины в порядке обратного приоритета.

Для простой сцены, вроде той, что показана на рис. 8.14, a, окончательный список приоритетов можно получить непосредственно. Например, эти многоугольники можно упорядочить по их максимальному или минимальному значению координаты *z*. Однако уже для сцены, показанной на рис. 8.14 b, окончательный список приоритетов по глубине невозможно получить простой сортировкой по *z*. Если *P* и *Q* с рис. рис. 8.14, b упорядочены по минимальному значению координаты *z* (z_{min}), окажется, что *P* в списке приоритетов по глубине будет стоять перед *Q*. Если их записать в буфер кадра в таком порядке, то получится, что *Q* частично экранирует *P*. Однако фактически *P* частично экранирует *Q*. Правильный порядок в списке приоритетов будет тогда, когда *P* и *Q* поменяются местами.

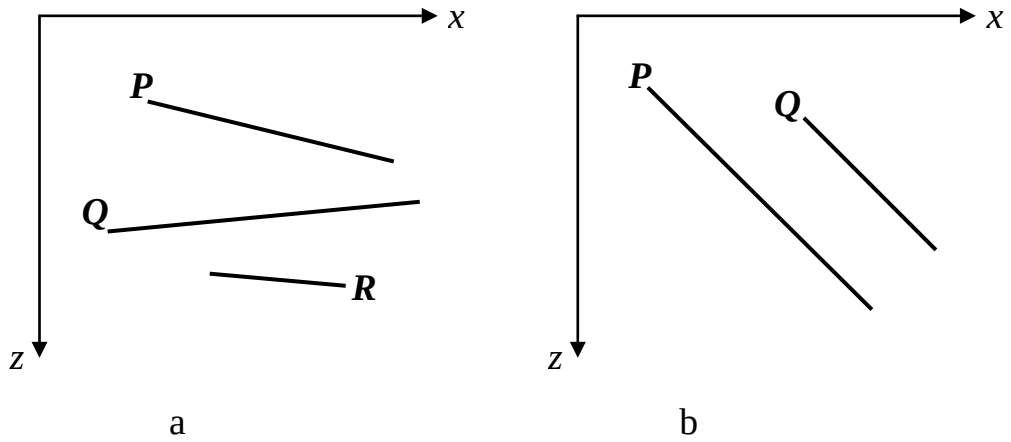
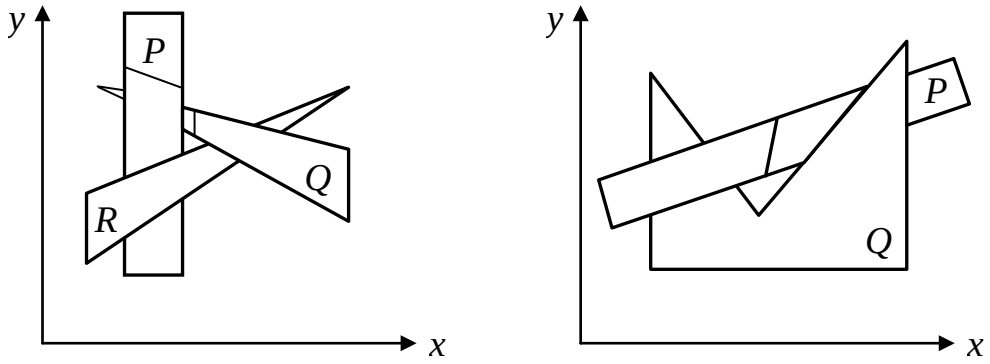


Рис. 8.14. Установление приоритетов для многоугольников

Другие трудности показаны на рис. 8.15. Здесь многоугольники циклически перекрывают друг друга. На рис. 8.15, a *P* находится впереди *Q*, который лежит впереди *R*, который, в свою очередь, находится впереди *P*. На рис. 8.15, b *P* экранирует *Q*, а *Q* экранирует *P*. Аналогичное циклическое

экранирование возникает при протыкании многоугольников; например, на рис. 5.12 показано, как треугольник протыкает прямоугольник. Там прямоугольник экранируется треугольником, и наоборот. В обоих примерах окончательный список приоритетов невозможно установить сразу. Выход из положения заключается в циклическом разрезании многоугольников по линиям, образованным пересечениями их плоскостей, до тех пор, пока не будет получен окончательный список приоритетов. Такие линии показаны пунктиром на рис. 8.15.



a b

Рис. 8.15. Циклически перекрывающие многоугольники

Ньюэл М., Ньюэл Р. и Санча предложили специальный метод сортировки для разрешения конфликтов, возникающих при создании списка приоритетов по глубине. Этот метод включен в состав алгоритма Ньюэла - Ньюэла - Санча, который излагается ниже. В алгоритме динамически вычисляется новый список приоритетов перед обработкой каждого кадра сцены. Не накладывается никаких ограничений на сложность сцены и на тип многоугольников, используемых для описания элементов сцены. Первоначальный алгоритм Ньюэла - Ньюэла - Санча был предназначен для обработки трехмерных тел. Это расширение не ограничено рамками многогранников. Оно может, кроме того, обрабатывать тела смешанных типов в рамках одной сцены.

8.7. Алгоритм Ньюэла-Ньюэла-Санча для случая многоугольников

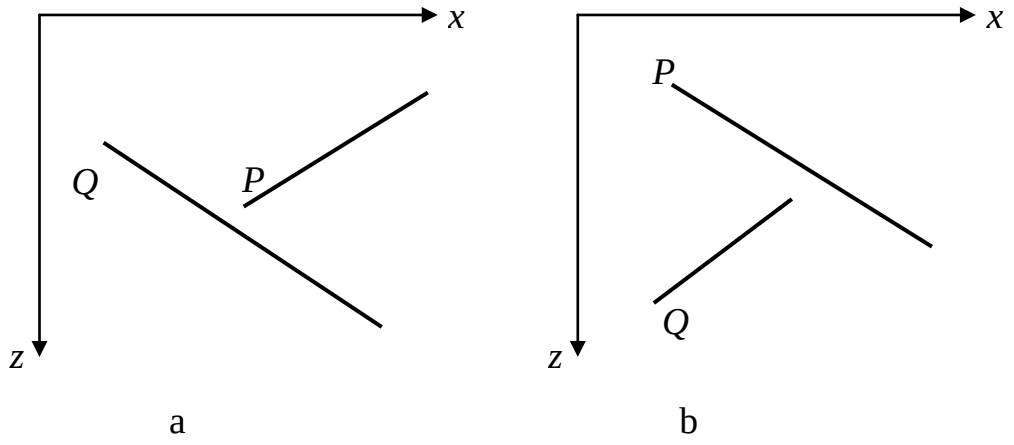
Сформировать предварительный список приоритетов по глубине, используя в качестве ключа сортировки значение z_{min} для каждого многоугольника. Первым в списке будет многоугольник с минимальным значением z_{min}. Этот многоугольник лежит дальше всех от точки наблюдения, расположенной в бесконечности на положительной полуоси z. Обозначим его

через P, а следующий в списке многоугольник - через Q. Для каждого многоугольника P из списка надо проверить его отношение с Q.

Если ближайшая вершина P ($P_{z\max}$) будет дальше от точки наблюдения, чем самая удаленная вершина Q ($Q_{z\min}$), т. е. $Q_{z\min} \geq P_{z\max}$ никакая часть P не может экранировать Q. Занести P в буфер кадра (см. рис. 8.14, а).

Если $Q_{z\min} < P_{z\max}$, то P потенциальное экранирует не только Q, но также и любой другой многоугольник типа Q из списка, для которого $Q_{z\min} < P_{z\max}$. Тем самым образуется множество $\{Q\}$. Однако P может фактически и не экранировать ни один из этих многоугольников. Если последнее верно, то P можно заносить в буфер кадра. Для ответа на этот вопрос используется серия тестов, следующих по возрастанию их вычислительной сложности. Эти тесты ниже формулируются в виде вопросов. Если ответ на любой вопрос будет положительным, то P не может экранировать $\{Q\}$. Поэтому P сразу же заносится в буфер кадра. Вот эти тесты:

- Верно ли, что прямоугольные объемлющие оболочки P и Q не перекрываются по x?
- Верно ли, что прямоугольные оболочки P и Q не перекрываются по y?
- Верно ли, что P целиком лежит по ту сторону плоскости, несущей Q, которая расположена дальше от точки наблюдения (рис. 8.16, а)?
- Верно ли, что Q целиком лежит по ту сторону плоскости, несущей P, которая ближе к точке наблюдения (рис. 8.16, b)? Верно ли, что проекции P и Q не перекрываются?



a b

Рис. 8.16. Тесты для перекрывающихся многоугольников

Каждый из этих тестов применяется к каждому элементу $\{Q\}$. Если ни один из них не дает положительного ответа и не заносит P в буфер кадра, то P может закрывать Q.

Поменять P и Q местами, пометив позицию Q в списке. Повторить тесты с новым списком. Это дает положительный результат для сцены с рис. 8.14, b.

Если сделана попытка вновь переставить Q, значит, обнаружена ситуация циклического экранирования (см. рис. 8.15). В этом случае P разрезается плоскостью, несущей Q, исходный многоугольник P удаляется из списка, а его части заносятся в список. Затем тесты повторяются для нового списка. Этот шаг предотвращает зацикливание алгоритма.

8.8. Алгоритм Варнока (Warnock)

Алгоритм Варнока является одним из примеров алгоритма, основанного на разбиении картинной плоскости на части, для каждой из которых исходная задача может быть решена достаточно просто.

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 фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения.

Кратко опишем оригинальную версию алгоритма, предложенного Варноком.

Разобьем видимую часть картинной плоскости на четыре равные части и рассмотрим, каким образом могут соотноситься между собой проекции граней получившейся части картинной плоскости. Возможны четыре различных случая:

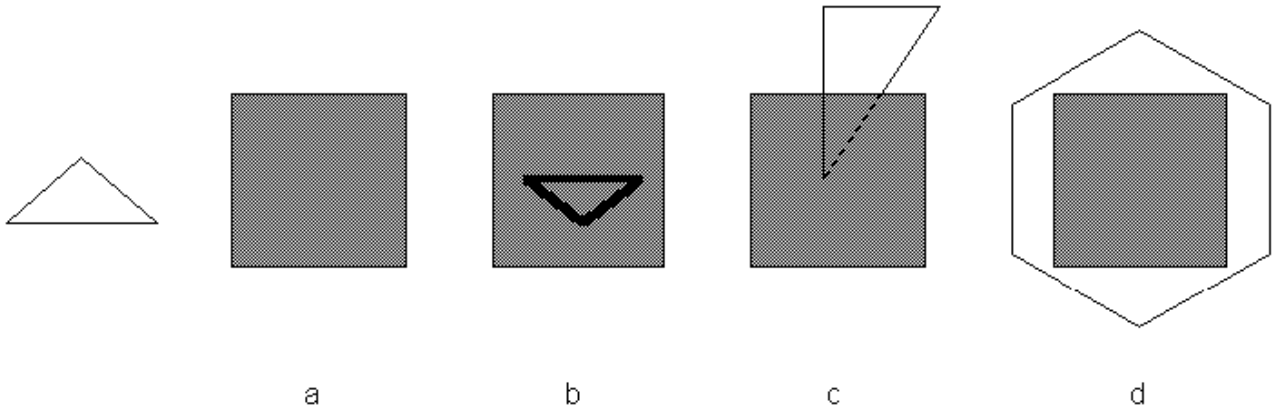


Рис. 8.17. Соотношение проекции грани с подокном

1. Проекция грани полностью накрывает область (рис. 8.17, d);
2. Проекция грани пересекает область, но не содержится в ней полностью (рис. 8.17, c);
3. Проекция грани целиком содержится внутри области (рис. 8.17, b);
4. Проекция грани не имеет общих внутренних точек с рассматриваемой областью (рис. 8.17, a).

Очевидно, что в последнем случае грань вообще никак не влияет на то, что видно в данной области.

Сравнивая область с проекциями всех граней, можно выделить случаи, когда изображение, получающееся в рассматриваемой области, определяется сразу:

- Проекция ни одной грани не попадает в область.
- Проекция только одной грани содержится в области или пересекает область. В этом случае проекции грани разбивают всю область на две части, одна из которых соответствует этой проекции.
- Существует грань, проекция которой полностью накрывает данную область, и эта грань расположена к картинной плоскости ближе, чем все остальные грани, проекции которых пересекают данную область. В данном случае область соответствует этой грани.

Если ни один из рассмотренных трех случаев не имеет места, то снова разбиваем область на четыре равные части и проверяем выполнение этих условий для каждой из частей. Те части, для которых таким образом не удалось установить видимость, разбиваем снова и т. д. Естественно, возникает вопрос о критерии, на основании которого прекращать разбиение. В качестве очевидного критерия можно взять размер области. Как только размер области станет не больше размера одного пикселя, то производить дальнейшее разбиение не имеет смысла и для данной области ближайшая к ней грань определяется явно, как в методе трассировки лучей.

Иногда, для устранения лестничного эффекта, процесс разбиения проводится до размеров, меньших, чем разрешение экрана, на один пиксель. При этом усредняются атрибуты подпикселей, чтобы определить атрибуты самих пикселей.

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. 8.18, a показан

другой общий способ разбиения и дано его сравнение с изложенным ранее жестким способом, представленным на рис. 8.18, b.

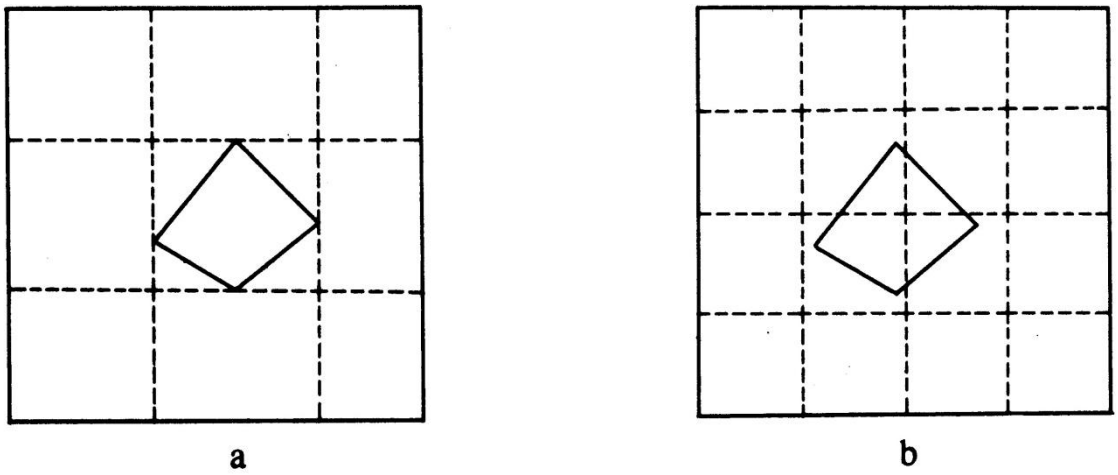


Рис. 8.18. Способы разбиения окна

Разбиение, показанное на рис. рис. 8.18, a, получается с использованием прямоугольной объемлющей оболочки многоугольника. Заметим, что оболочки при этом могут быть неквадратными. Этот способ можно рекурсивно применить к любому многоугольнику, который полностью охвачен каким-нибудь окном или оболочкой. 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. Однако с ростом сложности сцены его преимущество сходит на нет.

8.9. Алгоритм Вейлера-Азертона (Weiler-Atherton)

Разбиение картинной плоскости можно производить не только прямыми, параллельными координатным осям, но и по границам проекций граней. В результате получается точное решение задачи.

Предлагаемый метод работает с проекциями граней на картинную плоскость.

В качестве первого шага производится сортировка всех граней по глубине (front-to-back).

Затем из списка оставшихся граней берется ближайшая грань A и все остальные грани обрезаются по этой грани. Если проекция грани B пересекает проекцию грани A, то грань B разбивается на части так, что каждая часть либо содержится в грани A, либо не имеет с ней общих внутренних точек.

Таким образом, получаются два множества граней: F_{in} – грани, проекции которых содержатся в проекции грани A (сюда входит и сама грань A), и F_{out} –

грани, проекции которых не имеют общих внутренних точек с проекцией грани *A*.

Множество F_{in} обычно называют множеством граней, внутренних по отношению к *A*.

Однако во множестве F_{in} могут быть грани, лежащие к наблюдателю ближе, чем сама грань *A* (это возможно, например, при циклическом наложении граней). В этом случае каждая такая грань используется для разбиения всех граней из множества F_{in} (включая исходную грань *A*). Когда рекурсивное разбиение завершится, то все грани из первого множества выводятся из набора оставшихся граней (их уже ничто не может закрывать). Затем из набора оставшихся граней F_{out} берется очередная грань и процедура повторяется.

Рассмотрим простейший случай для двух граней *A* и *B* (рис. 8.19).

Будем считать, что грань *A* расположена ближе, чем грань *B*. Тогда на первом шаге для разбиения используется именно грань *A*. Грань *B* разбивается на две части. Часть B_1 попадает в первое множество F_{in} и, так как она лежит дальше грани *A*, удаляется.

После этого выводится грань *A* и в списке оставшихся граней остается только грань B_2. Так как кроме нее других граней не осталось, то эта грань выводится, и на этом работа завершается.

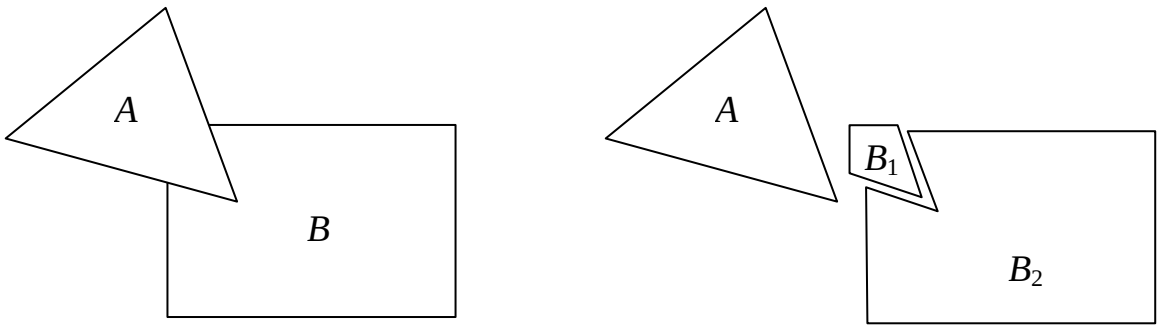


Рис. 8.19. Пример разбиения граней

9. Методы закраски

9.1. Диффузное отражение и рассеянный свет

Матовые поверхности обладают свойством ***диффузного отражения***, т. е. равномерного по всем направлениям рассеивания света. Поэтому кажется, что поверхности имеют одинаковую яркость независимо от угла обзора. Для таких поверхностей справедлив закон косинусов Ламберта, устанавливающий соответствие между количеством отраженного света и косинусом угла θ между

направлением L на точечный источник света интенсивности I_p и нормалью N к поверхности (рис. 9.1). При этом количество отраженного света не зависит от положения наблюдателя.

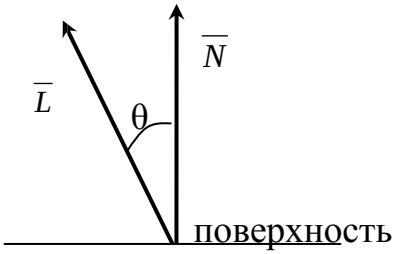


Рис. 9.1. Падающий свет и нормаль к поверхности

Освещенность рассеянным светом вычисляется по формуле

$$I_d = I_p \cdot k_d \cdot \cos\theta.$$

Значение коэффициента диффузного отражения k_d является константой в диапазоне (0, 1) и зависит от материала. Если векторы $\overline{L}$ и $\overline{N}$ нормированы, то, используя скалярное произведение, формулу освещенности можно записать так:

$$I_d = I_p \cdot k_d \cdot (\overline{L} \cdot \overline{N}).$$

Предметы, освещенные одним точечным источником света, выглядят контрастными. Этот эффект аналогичен тому, который можно наблюдать, когда предмет, помещенный в темную комнату, виден при свете направленной на него фотовспышки. В данной ситуации, в отличие, от большинства реальных визуальных сцен, отсутствует рассеянный свет, под которым здесь понимается свет постоянной яркости, созданный многочисленными отражениями от различных поверхностей. Такой свет практически всегда присутствует в реальной обстановке. Даже если предмет защищен от прямых лучей, исходящих от точечного источника света, он все равно будет виден из-за наличия рассеянного света. Учитывая это, формулу окраски можно записать так:

$$I_d = I_a \cdot k_a + I_p \cdot k_d \cdot (\overline{L} \cdot \overline{N})$$

Рассеянный свет представлен членом I_a, и k_a определяет количество рассеянного света, которое отражается от поверхности предмета.

Точечный источник света удобнее всего расположить в позиции, совпадающей с глазом наблюдателя. Тени в этом случае отсутствуют, а лучи света, падающие на поверхность, окажутся параллельными. Однако теперь, если две поверхности одного цвета параллельны друг другу и их изображения перекрываются, нормали к поверхностям совпадают и, следовательно, поверхности закрашиваются одинаково, различить их невозможно. Этот эффект можно устранить, если учесть, что энергия падающего света убывает пропорционально квадрату расстояния, которое свет проходит от источника до поверхности и обратно к глазу наблюдателя. Обозначая это расстояние за R, запишем:

$$I_d = I_a \cdot \bar{k}_a + I_p \cdot k_d \cdot (L \cdot N)/R^2.$$

Однако данным правилом на практике трудно воспользоваться. Для параллельной проекции, когда источник света находится в бесконечности, расстояние R также становится бесконечным. Даже в случае центральной проекции величина $1/R^2$ может принимать значения в широком диапазоне, поскольку точка зрения часто оказывается достаточно близкой к предмету. В результате закраска поверхностей,

которые имеют одинаковые углы θ между N и L, будет существенно различаться. Большей реалистичности можно достичь, если заменить R^2 на $r+k$, где k – некоторая константа, а r –расстояние от центра проекции до поверхности:

$$I_d = I_a \cdot k_a + I_p \cdot \bar{k}_d \cdot (L \cdot N)/(r+k).$$

Для представления диффузного отражения от цветных поверхностей уравнения записываются отдельно для основных цветов модели CMY (голубого, пурпурного и желтого). При этом константы отражения для этих цветов задаются тройкой чисел (k_{dc}, k_{dm} k_{dy}). Эти цвета используются, поскольку отражение света является субтрактивным процессом. Поэтому интенсивность для цветного изображения описывается тремя уравнениями:

$$I_{dc} = I_{ac} \cdot k_{ac} + I_{pc} \cdot k_{dc} \cdot (L \cdot N)/(r+k) \quad \text{(для голубой компоненты);}$$

$$I_{dm} = I_{am} \cdot k_{am} + I_{pm} \cdot k_{dm} \cdot (\bar{L} \cdot \bar{N})/(r+k)$$ (для пурпурной компоненты);

$$I_{dy} = I_{ay} \cdot k_{ay} + I_{py} \cdot k_{dy} \cdot (\bar{L} \cdot \bar{N})/(r+k)$$ (для желтой компоненты).

9.2. Зеркальное отражение

Зеркальное отражение можно получить от любой блестящей поверхности. Осветите ярким светом яблоко – световой блик на яблоке возникает в результате зеркального отражения, а свет, отраженный от остальной части, появится в результате диффузного отражения. Отметим также, что в том месте, где находится световой блик, яблоко кажется не красным, а скорее белым, т. е. окрашенным в цвет падающего цвета.

Если мы изменим положение головы, то заметим, что световой блик тоже сместится. Это объясняется тем, что блестящие поверхности отражают свет неодинаково по всем направлениям. От идеального зеркала свет отражается только в том направлении, для которого углы падения и отражения совпадают. Это означает, что наблюдатель сможет увидеть зеркально отраженный свет только в том случае, если угол α (рис. 6.2.) равен нулю.

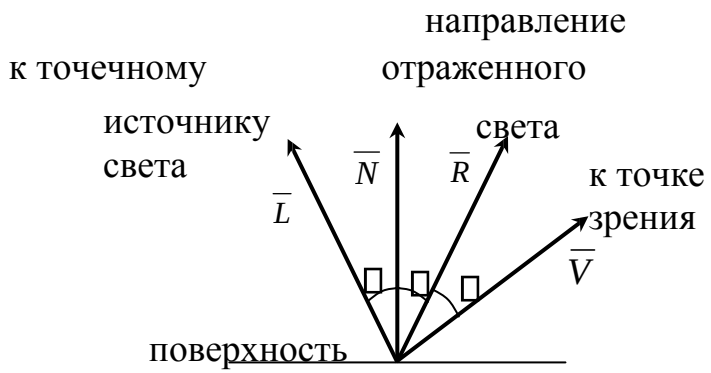


Рис. 9.2. Зеркальное отражение

Для неидеальных отражающих поверхностей, таких, как яблоко, интенсивность отраженного света резко падает с ростом α. В модели предложенной Фонгом, быстрое убывание интенсивности описывается функцией $\cos^n \alpha$, где n обычно лежит в диапазоне 1–200, в зависимости от вида поверхности. Для идеального отражателя n бесконечно велико. В основе такой модели лежит эмпирическое наблюдение, а не фундаментальное понимание процесса зеркального отражения.

Количество падающего света, которое зеркально отражается в случае реальных материалов, зависит от угла падения . Обозначим зеркально отражаемую долю света через $W(\theta)$, тогда

$$I_d = I_a \cdot k_a + \sum [k_d \cdot \cos\theta + W(\theta) \cdot \cos^n r \cdot k].$$

Если векторы направления отраженного света и направления к точке зрения R и V нормированы, то $\cos\theta = (R \cdot V)$. Часто в качестве $W(\theta)$ служит константа k_s, которая выбирается таким образом, чтобы получающиеся результаты были приемлемы с эстетической точки зрения. В этом случае уравнение с учетом зеркального отражения можно записать так:

$$I_d = I_a \cdot k_a + \sum [k_d \cdot (L \cdot N) + k_s \cdot (R \cdot V)^n r \cdot k].$$

Для цветного изображения описываются три уравнения: для голубого, пурпурного и желтого цветов:

$$I_{dc} = I_{ac} \cdot k_{ac} + \sum [k_{dc} \cdot (L \cdot N) + k_s \cdot (R \cdot V)^n r \cdot k];$$

$$I_{dm} = I_{am} \cdot k_{am} + \sum [k_{dm} \cdot (L \cdot N) + k_s \cdot (R \cdot V)^n r \cdot k];$$

$$I_{dy} = I_{ay} \cdot k_{ay} + \sum [k_{dy} \cdot (L \cdot N) + k_s \cdot (R \cdot V)^n r \cdot k].$$

Если источник света расположен в бесконечности, для заданного многоугольника произведение $(L \cdot N)$ является константой, а $(R \cdot V)$ меняет значение в многоугольнике.

Кроме эмпирической модели Фонга, для зеркального отражения разработана модель Торрэнса-Спэрроу, которая представляет собой теоретическую обоснованную модель отражающей поверхности. В этой модели предполагается, что поверхность является совокупностью микроскопических граней, каждая из которых – идеальный отражатель. Ориентация любой грани задается функцией распределения вероятностей Гаусса.

9.3. Однотонная закраска полигональной сетки

Существует три основных способа закраски объектов, заданных полигональными сетками. В порядке возрастания сложности ими являются:

1) однотонная закраска;

2) метод Гуро (основан на интерполяции значений интенсивности);

3) метод Фонга (основан на интерполяции векторов нормали).

В каждом из этих случаев может быть использована любая из моделей закраски, описанная выше.

При однотонной закраске вычисляют один уровень интенсивности, который используется для закраски всего многоугольника. При этом предполагается, что:

1. Источник света расположен в бесконечности, поэтому произведение $(\bar{L} \cdot \bar{N})$ постоянно на всей полигональной грани.

2. Наблюдатель находится в бесконечности, поэтому произведение $(\bar{N} \cdot \bar{V})$ постоянно на всей полигональной грани.

3. Многоугольник представляет реальную моделируемую поверхность, а не является аппроксимацией криволинейной поверхности. Если какое-либо из первых двух предположений оказывается неприемлемым, можно воспользоваться усредненными значениями $\bar{L}$ и $\bar{V}$, вычисленными, например, в центре многоугольника.

Последнее предположение в большинстве случаев не выполняется, но оказывает существенно большее влияние на получаемое изображение, чем два других. Влияние состоит в том, что каждая из видимых полигональных граней аппроксимированной поверхности хорошо отличима от других, поскольку интенсивность каждой из этих граней отличается от интенсивности соседних граней. Различие в окраске соседних граней хорошо заметно вследствие эффекта полос Маха.

9.4. Метод Гуро

Метод закраски, который основан на интерполяции интенсивности и известен как метод Гуро (по имени его разработчика), позволяет устранить дискретность изменения интенсивности. Процесс закраски по методу Гуро осуществляется в четыре этапа:

1. Вычисляются нормали ко всем полигонам.

2. Определяются нормали в вершинах путем усреднения нормалей по всем полигональным граням, которым принадлежит вершина (рис. 9.3).

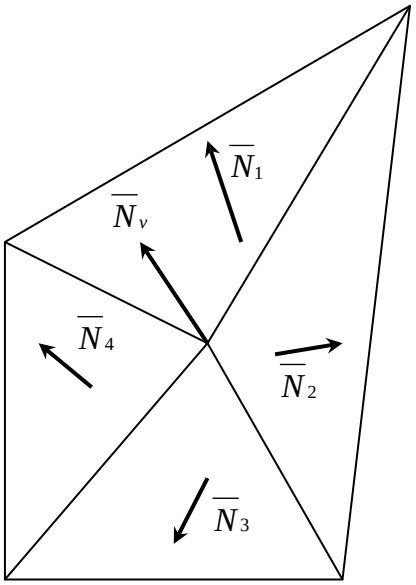


Рис. 9.3. Нормали к вершинам: $\overline{N}_v=(\overline{N}_1+\overline{N}_2+\overline{N}_3+\overline{N}_4)/4$

3. Используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах.

4. Каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки (рис. 9.4).

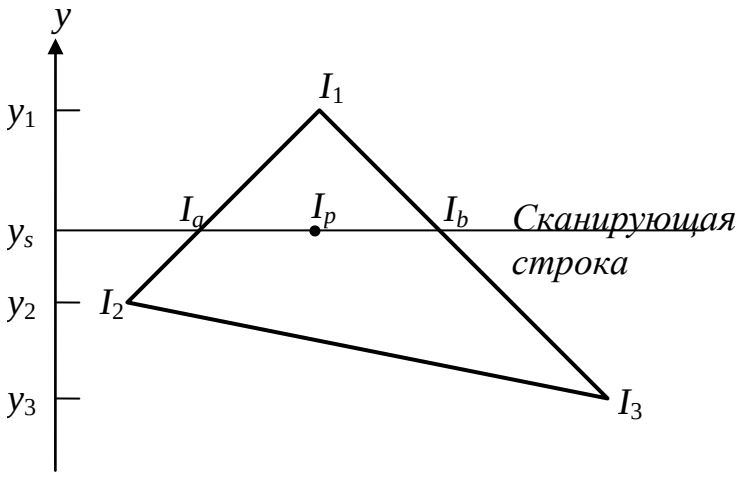


Рис. 9.4. Интерполяция интенсивностей

Интерполяция вдоль ребер легко объединяется с алгоритмом удаления скрытых поверхностей, построенным на принципе построчного сканирования. Для всех ребер запоминается начальная интенсивность, а также изменение

интенсивности при каждом единичном шаге по координате *y*, Заполнение видимого интервала на сканирующей строке производится путем интерполяции между значениями интенсивности на двух ребрах, ограничивающих интервал (рис 6.4).

$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}; \quad y_1 \neq y_2$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}; \quad y_1 \neq y_3$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}, \quad x_b \neq x_a$$

Для цветных объектов отдельно интерполируется каждая из компонент цвета.

9.5. Метод Фонга

В методе закраски, разработанном Фонгом, используется

интерполяция вектора нормали $\bar{N}$ к поверхности вдоль видимого интервала на сканирующей строке внутри многоугольника, а не интерполяция интенсивности. Интерполяция выполняется между начальной и конечной нормалями, которые сами тоже являются результатами интерполяции вдоль ребер многоугольника между нормалями в вершинах. Нормали в вершинах, в свою очередь, вычисляются так же, как в методе закраски, построенном на основе интерполяции интенсивности.

В каждом пикселе вдоль сканирующей строки новое значение интенсивности вычисляется с помощью любой модели закраски. Заметные улучшения по сравнению с интерполяцией интенсивности наблюдаются в случае использования модели с учетом зеркального отражения, так как при этом более точно воспроизводятся световые блики. Однако даже если зеркальное отражение не используется, интерполяция векторов нормали приводит к более качественным результатам, чем интерполяция интенсивности, поскольку аппроксимация нормали в этом случае осуществляется в каждой точке. При этом значительно возрастают вычислительные затраты.

Чтобы закрасить куски бикубической поверхности, для каждого пикселя, исходя из уравнений поверхности, вычисляется нормаль к поверхности. Этот процесс тоже достаточно дорогой. Затем с помощью любой модели закраски определяется значение интенсивности. Однако прежде чем применить метод закраски к плоским или бикубическим поверхностям, необходимо иметь информацию о том, какие источники света (если они имеются) в

действительности освещают точку. Поэтому мы должны рассматривать также и тени.

9.6. Тени

Алгоритмы затенения в случае точечных источников света идентичны алгоритмам удаления скрытых поверхностей. В алгоритме удаления скрытых поверхностей определяются поверхности, которые можно увидеть из точки зрения, а в алгоритме затенения выделяются поверхности, которые можно «увидеть» из источника света.

Поверхности, видимые как из точки зрения, так и из источника света, не лежат в тени. Те же поверхности, которые видимы из точки зрения, но невидимы из источника света, находятся в тени. Эти рассуждения можно легко распространить на случай нескольких источников света. Отметим, однако, что, используя такой простой подход, нельзя смоделировать тени от распределенных источников света. При наличии таких источников потребуется вычислять как тени, так и полутени.

Поскольку алгоритмы затенения и удаления скрытых поверхностей одинаковы, представляется возможным обрабатывать описание объекта, используя лишь один из этих алгоритмов, последовательно применяя его к точке зрения и к каждому из точечных источников света. Совокупность полученных результатов позволяет определить, какие части объекта видимы наблюдателю и какие видны из одного или нескольких источников света. На основании этой информации осуществляется закраска сцены. Если правильно организовать вычислительный процесс, определение теней можно проводить лишь один раз для серии сцен, которые состоят из одних и тех же объектов, рассматриваемых с различных точек зрения. Источники света предполагаются неподвижными относительно объектов. Все это оказывается возможным потому, что тени не зависят от положения точки зрения.

9.7. Поверхности, пропускающие свет

Поверхности могут обладать не только свойствами зеркального и диффузного отражения, но и свойствами направленного и диффузного пропускания. Направленное пропускание света происходит сквозь прозрачные вещества (например, стекло или отшлифованный люциит). Через них обычно хорошо видны предметы, даже несмотря на то, что лучи света, как правило, преломляются, т. е. отклоняются от первоначального направления. Диффузное пропускание света происходит сквозь просвечивающие материалы (например, замерзшее стекло), в которых поверхностные или внутренние неоднородности приводят к беспорядочному перемешиванию световых лучей. Поэтому когда

предмет рассматривается через просвечивающее вещество, его очертания размыты.

Изложим кратко идею только одного метода (подход Уиттеда), учитывающего пропускающий свет. Данный подход основан на использовании алгоритмов трассировки лучей. Его основная идея заключается в трассировании световых лучей и определении, какие из этих лучей попадают в точку зрения. К сожалению, из каждой точки источника света исходит бесконечное число лучей, причем большинство из них никогда не достигает точки зрения. Поэтому трассирование начинается из точки зрения и лучи отслеживаются в обратном направлении через каждый пиксель к их источнику. Луч света, падающий на поверхность, в общем случае разделяется на три части: диффузно отраженный свет, зеркально отраженный свет и пропущенный (и, следовательно, преломленный) свет. Аналогично луч света, исходящий от поверхности объекта, в общем случае является суммой составляющих от трех источников. Это означает, что каждый раз, когда луч исходит от объекта, возможно появление трех новых лучей, которые должны быть оттрассированы. К сожалению, диффузное отражение приводит к появлению бесконечного числа лучей, поэтому трассируются только лучи, появляющиеся в результате зеркального отражения и преломления. Для моделирования рассеянного и

диффузного света используется уравнение

$$I_d = I_a \cdot k_a + I_p \cdot k_d \cdot (\overline{L} \cdot \overline{N})$$

9.8. Детализация поверхностей

Существуют два способа детализации поверхности: цветом и фактурой. В результате применения к гладкой поверхности детализации цветом форма поверхности не изменяется, если же производится детализация фактурой – поверхность становится шероховатой.

9.8.1. Детализация цветом

Детализацию цветом на глубоком уровне легко осуществить путем введения многоугольников детализации поверхности, чтобы выделить особенности (такие, как двери, окна и надписи) на основном многоугольнике (например, на стене здания). Многоугольники детализации поверхности лежат в одной плоскости с основными многоугольниками и так помечены в структуре данных, чтобы алгоритм удаления скрытых поверхностей мог присвоить им более высокие приоритеты, чем основным многоугольникам.

По мере того как детализация цветом становится более тонкой и сложной, непосредственное моделирование при помощи многоугольников становится менее практичным.

9.8.2. Детализация фактурой

Идея детализации фактурой состоит в отображении массива узора, представляющего собой оцифрованное изображение, на плоскую или криволинейную поверхность. Значения из массива узора используются для масштабирования диффузной компоненты интенсивности.

Один пиксель на экране может покрывать несколько элементов массива узора. Чтобы избежать проблем, связанных с лестничным эффектом, необходимо учитывать все затрагивающие пиксель элементы. Для этого определяются четыре точки в массиве узора, которые соответствуют четырем углам пикселя. Эти точки в массиве узора образуют четырехугольник. Значения попадающих в него элементов взвешиваются с учетом доли каждого элемента, а затем суммируются.

Отображение при такой детализации проводится в два этапа:

1. Фиксированное отображение рисунка на поверхность объекта.
2. Видовое преобразование объекта на экран.

Отображение массива узора влияет на расцветку поверхности, однако поверхность продолжает казаться геометрически гладкой. Существует два способа нанесения на поверхность ***деталей фактуры***. В первом из них непосредственное геометрическое моделирование фактуры не производится, и тем не менее получается хороший визуальный эффект. Для этого вносится возмущение в нормаль к поверхности до ее использования в модели закраски. Эти возмущения моделируют небольшие неровности на поверхности.

Второй способ основывается на использовании фрактальных поверхностей, т. е. класса нерегулярных форм, задаваемых вероятностным образом и хорошо описывающих многие реальные формы, такие, как рельефы местности, береговые линии, сети рек, хлопья снега и ветви деревьев. Например, реалистичное изображение горы создается путем аппроксимации горы при помощи полигональной сетки. Каждый полигон, который необязательно является плоским, затем некоторое число раз рекурсивно подразделяется, чтобы создать неровный, с зазубринами, рельеф местности. Разбиение проводится с применением случайной функции. Таким образом, из начальной аппроксимации получается множество многоугольников. Далее проводится удаление скрытых поверхностей и применяется соответствующая модель закраски.

10. Библиотека OpenGL

На данный момент в Windows существует два стандарта для работы с трѐхмерной графикой: OpenGL, являющийся стандартом дефакто для всех графических рабочих станций, и Direct3D – стандарт, предложенный фирмой Microsoft. Далее будет рассмотрен только стандарт OpenGL

Существенным достоинством OpenGL является его широкая распространенность – он является стандартом в мире графических рабочих станций типа Sun, Silicon Graphics и др. В основу стандарта была положена библиотека IRIS GL, разработанная фирмой Silicon Graphics Inc.

OpenGL представляет собой программный интерфейс к графическому оборудованию (хотя существуют и чисто программные реализации OpenGL). Интерфейс насчитывает около 120 различных команд, которые программист использует для задания объектов и операций над ними (необходимых для написания интерактивных трѐхмерны приложений).

OpenGL был разработан как эффективный, аппаратно-независимый интерфейс, пригодный для реализации на различных аппаратных платформах. Поэтому OpenGL не включает в себя никаких специальных команд для работы с окнами или ввода информации от пользователя. OpenGL позволяет:

1. Создавать объекты из геометрических примитивов (точки, линии, грани и битовые изображения).
2. Располагать объекты в трѐхмерном пространстве и выбирать способ и параметры проецирования.
3. Вычислять цвет всех объектов. Цвет может быть как явно задан, так и вычисляться с учѐтом источников света, параметров освещения, текстур.
4. Переводить математическое описание объектов и связанной с ними информации о цвете в изображение на экране.

При этом OpenGL может осуществлять дополнительные операции, такие, как удаление невидимых фрагментов изображения.

Команды OpenGL реализованы как модель клиент-сервер. Приложение выступает в роли клиента: оно вырабатывает команды, а сервер OpenGL интерпретирует и выполняет их. Сам сервер может находиться как на том же компьютере, на котором находится и клиент, так и на другом.

10.1. Особенности использования OpenGL в Windows

OpenGL представляет собой универсальную графическую библиотеку, которая может быть реализована в любой оконной среде. Поставляется в составе операционной системы Windows, начиная с версии OSR2 в виде двух DLL-файлов – opengl32.dll и glu32.dll. Первая из этих библиотек и есть

собственно набор функций OpenGL, вторая содержит дополнительный набор функций, упрощающих кодирование, но построенных и выполняемых с подключением opengl32.dll, и является надстройкой.

То, что эти библиотеки поставляются в составе операционной системы, значительно упрощает распространение разработанных приложений. То, что OpenGL распространяется в виде динамических библиотек, упрощает доступ к его функциям.

Для работы с OpenGL в Windows используется понятие контекста воспроизведения (rendering context), который связывает OpenGL с оконной системой Windows. Если обычный контекст устройства (device context) содержит информацию, относящуюся к графическим компонентам GDI, то контекст воспроизведения содержит информацию, относящуюся к OpenGL.

Таким образом, чтобы начать работать с командами OpenGL, приложение должно создать, как минимум, один контекст воспроизведения и сделать его текущим.

Перед созданием контекста воспроизведения необходимо установить формат пикселей. Для установки формата пикселей

используется функция Windows GDI `int ChoosePixelFormat(HDC, const PIXELFORMATDESCRIPTOR)`, выбирающая наиболее подходящий формат исходя из информации, переданной в полях структуры `PIXELFORMATDESCRIPTOR`.

После того как найден подходящий формат пикселей, следует установить его в контексте устройства при помощи функции `BOOL SetPixelFormat(HDC hDC, inl pixelFormat, const PIXELFORMATDESCRIPTOR)`.

Для работы с контекстом воспроизведения в Windows существуют

функции `HGLRC wglCreateContext(HDC hDC)` и `BOOL wglMakeCurrent(HDC hDC, HGLRC hGLRC)`.

Первая из них создаèт новый контекст воспроизведения OpenGL, который подходит для рисования на устройстве, задаваемом контекстом `hDC`. Вторая функция устанавливает текущий контекст воспроизведения.

По окончании работы с OpenGL созданный контекст воспроизведения необходимо удалить. Для этого существует функция

`BOOL wglDeleteContext(HGLRC hGLRC)`.

Текущий контекст воспроизведения можно узнать при помощи функции `HGLRC wglGetCurrentContext()`.

При помощи OpenGL можно создавать анимации. При этом для изображения используется режим работы с двумя буферами, когда содержимое одного из них показывается, а в другом осуществляется построение. После окончания построения специальная команда меняет буферы местами (по аналогии с двухстраничным режимом работы). Для использования двойной буферизации необходимо установить флаг `PFD_DOUBLE_BUFFER` при задании формата пикселей и применить команду `SwapBuffers`, меняющую буферы местами (по умолчанию вывод происходит в невидимый буфер).

10.2. Основные типы данных

Все команды (процедуры и функции) OpenGL начинаются с префикса **gl**, а все константы – с префикса **GL_**. Кроме того, в имена функций и процедур OpenGL входят суффиксы, несущие информацию о числе передаваемых параметров и о их типе. В табл. 1 приводятся вводимые OpenGL типы данных, стандартные типы языка C и суффиксы, которым они соответствуют.

Таблица 1 Типы данных OpenGL

Суффикс	Описание	Тип в C	Тип в OpenGL
b	8-битовое целое	`char`	`GLbyte`
s	16-битовое целое	`short`	`GLshort`
i	32-битовое целое	`long`	`GLint GLsizei`
f	32-битовое вещественное число	`float`	`GLfloat, GLclampf`
d	64-битовое вещественное число	`double`	`GLdouble, GLclampd`
ub	8-битовое беззнаковое целое	`unsigned char`	`GLubyte, GLboolean`
us	16-битовое беззнаковое целое	`unsigned short`	`GLushort`
ui	32-битовое беззнаковое целое	`unsigned long`	`GLuint, GLenum, GLbitfield`
		`void`	`GLvoid`

Некоторые команды OpenGL оканчиваются на букву *v*. Это говорит о том, что команда получает указатель на массив значений, а не сами эти значения в виде отдельных параметров. Многие команды имеют как векторные, так и не векторные версии. Например, конструкции

`glColor3f(1.0, 1.0, 1.0);` и
`GLfloat color[] = {1.0, 1.0, 1.0}; glColor3fv(color);`
эквивалентны.

OpenGL можно рассматривать как автомат, находящийся в одном из нескольких состояний. Внутри OpenGL содержится целый ряд переменных, например, текущий цвет или текущий режим закрашивания. Если установить текущий цвет, то все последующие объекты будут этого цвета до тех пор, пока текущий цвет не будет изменѐн.

По умолчанию каждая системная переменная имеет своѐ значение, и в любой момент значение каждой из этих переменных можно узнать. Обычно для этого используется одна из следующих функций: `glGetBooleanv()`, `glGetDoublev()`, `glGetFloatv()` и `glGetIntegerv()`. Для определения значений некоторых переменных служат специальные функции.

OpenGL предоставляет пользователю достаточно мощный, но низкоуровневый набор команд, и все операции высокого уровня должны выполняться в терминах этих команд. Обычно для облегчения работы вместе с OpenGL поставляется библиотека дополнительных команд, каждая из которых начинается с префикса **glu**. Далее будет рассмотрена часть из этих команд.

10.3. Рисование геометрических объектов

10.3.1. Работа с буферами и задание цвета объектов

OpenGL содержит внутри себя несколько различных буферов. Среди них фрейм буфер (где строится изображение), *z*-буфер, служащий для удаления невидимых поверхностей, буфер трафарета и аккумулирующий буфер.

Для очистки внутренних буферов служит процедура `glClear(GLbitfield mask)`, очищающая буферы, заданные переменной `mask`. Параметр mask является комбинацией следующих констант:

`GL_COLOR_BUFFER_BIT` – очистить буфер изображения (фреймбуфер);

`GL_DEPTH_BUFFER_BIT` – очистить *z*-буфер;

`GL_ACCUM_BUFFER_BIT` – очистить аккумулирующий буфер;

`GL_STENCIL_BUFFER_BIT` – очистить буфер трафарета.

Цвет, которым очищается буфер изображения, задаѐтся процедурой `glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)`. Значение, записываемое в *z*-буфер при очистке, задаѐтся процедурой `glClearDepth(GLfloat depth)`. Значение, записываемое в буфер трафарета при очистке, задаѐтся процедурой

`glClearStencil(GLint s)`. Цвет, записываемый в аккумулирующий буфер при очистке, задаѐтся процедурой `glClearAccum(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha)`.

Сама команда `glClear` очищает одновременно все заданные буферы, заполняя их соответствующими значениями. Для задания цвета объекта служит процедура

```
glColor{3 4}{b s i f d ub us ui}[v](TYPE red, …).
```

Цифра 3 или 4 указывает на количество требуемых аргументов, а буква, следующая за цифрой, показывает тип аргументов. Например, в процедуру `glColor3i` будут переданы три параметра целого типа.

Если значение параметра не задано, то оно автоматически полагается равным единице. Версии процедуры `glColor`, где параметры являются переменными с плавающей точкой, автоматически обрезают переданные значения в отрезок [0, 1].

Процедура `glFlush()` вызывает немедленное рисование ранее переданных команд. При этом ожидания завершения всех ранее переданных команд не происходит. С другой стороны, команда `glFinish()` ожидает, пока не будут завершены все ранее переданные команды.

Если нужно включить удаление невидимых поверхностей методом *z*-буфера, то *z*-буфер необходимо очистить и передать команду `glEnable(GL_DEPTH_TEST)`. Команду `glEnable()` можно выполнить только один раз при инициализации системных переменных OpenGL. Очистку *z*-буфера необходимо производить перед началом построения очередного кадра изображения.

10.3.2. Задание графических примитивов

Все геометрические примитивы в OpenGL задаются в терминах вершин. Каждая вершина задаѐтся набором чисел, определяющих еѐ координаты в пространстве.

OpenGL работает с однородными координатами (*x*, *y*, *z*, *w*). Если координата *z* не задана, то она считается равной нулю. Если координата *w* не задана, то она считается равной единице.

Под линией в OpenGL подразумевается отрезок, заданный своими начальной и конечной вершинами.

Под гранью (многоугольником) в OpenGL подразумевается замкнутый выпуклый многоугольник с несамопересекающейся границей.

Все геометрические объекты в OpenGL задаются посредством вершин, а сами вершины задаются процедурой

```
glVertex{2 3 4}{s i f d}[v](TYPE x, …),
```

где реальное количество параметров определяется первым суффиксом (2, 3 или 4), а суффикс *v* означает, что в качестве единственного аргумента выступает массив, содержащий необходимое количество координат. Например:

```
glVertex2s(1, 2); glVertex3f(2.3, 1.5, 0.2);
GLdouble vect[] = {1.0, 2.0, 3.0, 4.0};
glVertex4dv(vect);
```

Для задания геометрических примитивов необходимо как-то выделить набор вершин, определяющих этот объект. Для этого служат процедуры `glBegin()` и `glEnd()`. Процедура `glBegin(GLenum mode)` обозначает начало списка вершин, описывающих геометрический примитив. Тип примитива задаётся параметром `mode`, который принимает одно из следующих значений:

`GL_POINTS` – набор отдельных точек;

`GL_LINES` – пары вершин, задающих отдельные точки;

`GL_LINE_STRIP` – незамкнутая ломаная;

`GL_LINE_LOOP` – замкнутая ломаная;

`GL_POLYGON` – простой выпуклый многоугольник;

`GL_TRIANGLES` – тройки вершин, интерпретируемые как вершины отдельных треугольников;

`GL_TRIANGLE_STRIP` – связанная полоса треугольников;

`GL_TRIANGLE_FAN` – веер треугольников;

`GL_QUADS` – четвёрки вершин, задающие выпуклые четырёхугольники;

`GL_QUAD_STRIP` – полоса четырёхугольников.

Процедура `glEnd()` отмечает конец списка вершин.

Между командами `glBegin()` и `glEnd()` могут находится команды задания различных атрибутов вершин: `glVertex()`, `glColor()`, `glNormal()`, `glCallList()`, `glCallLists()`, `glTexCoord()`, `glEdgeFlag()`, `glMaterial()`. Между командами `glBegin()` и `glEnd()`

все остальные команды OpenGL недопустимы и приводят к возникновению ошибок.

Рассмотрим в качестве примера задание окружности:

```
 glBegin(GL_LINE_LOOP); for (int i = 0; I < N;
i++)
    {
    float angle = 2 * M_PI * i / N;
glVertex2f(cos(angle), sin(angle));
    } glEnd();
```

Хотя многие команды могут находиться между `glBegin()` и `glEnd()`, вершины генерируются при вызове `glVertex()`. В момент вызова `glVertex()` OpenGL присваивает создаваемой вершине текущий цвет, координаты текстуры, вектор нормали и т. д. Изначально вектор нормали полагается равным (0, 0, 1), цвет полагается равным (1, 1, 1, 1), координаты текстуры полагаются равными нулю.

10.3.3. Рисование точек, линий и многоугольников

Для задания размеров точки служит процедура `glPointSize(GLfloat size)`, которая устанавливает размер точки в пикселях, по умолчанию он равен единице.

Для задания ширины линии в пикселях служит процедура `gflineWidth(GLfloatwidth)`. Шаблон, которым будет рисоваться линия, можно задать при помощи процедуры `glLineStipple(Glint factor, GLushort pattern)`. Шаблон задается переменной `pattern` и растягивается в `factor` раз. Использование шаблонов

линии необходимо разрешить при помощи команды `glEnable(GL_UNE_STIPPLE)`. Запретить использование шаблонов линий можно командой `glDisable(GL_LINE_ST1PPLE)`.

Многоугольники рисуются как заполненные области пикселей внутри границы, хотя их можно рисовать либо только как граничную линию, либо просто как набор граничных вершин.

Многоугольник имеет две стороны (лицевую и нелицевую) и может быть отрисован по-разному, в зависимости от того, какая сторона обращена к наблюдателю. По умолчанию обе стороны рисуются одинаково. Для задания того, как именно следует рисовать переднюю и заднюю стороны многоугольника, служит процедура `glPolygonMode(GLenum face, GLenum mode)`. Параметр `face` может принимать значения

GL_FRONT_AND_BACK (обе стороны), GL_FRONT (лицевая сторона) или GL_BACK (нелицевая сторона). Параметр mode может принимать значения GL_POINT, GL_LINE или GL_FILL, обозначая, что многоугольник должен рисоваться как набор граничных точек, граничная ломаная линия или заполненная область, например:

```
    glPolygonMode(GL_FRONT, GL_FILL);
glPolygonMode(GL_BACK, GLJJNE).
```

По умолчанию вершины многоугольника, которые появляются на экране в направлении против часовой стрелки, называются ***лицевыми***. Это можно изменить при помощи процедуры glFrontFace(GLenum mode). По умолчанию параметр mode равняется GL_CCW, что соответствует направлению обхода против часовой стрелки. Если задать этот параметр равным GL_CW, то лицевыми будут считаться многоугольники с направлением обхода вершин по часовой стрелке.

При помощи процедуры glCullFace(GLenum mode) вывод лицевых или нелицевых граней многоугольников можно запретить. Параметр mode принимает одно из значений GL_FRONT (оставить только лицевые грани), GL_BACK (оставить нелицевые) или GL_FRONT_AND_BACK (оставить все грани). Для отсечения граней необходимо разрешить отсечение при помощи команды glEnable(GL_CULL_FACE).

Шаблон для заполнения грани можно задать при помощи процедуры gIPolygonStipple(const GLubyte * mask), где mask задает массив битов размером 32 на 32.

Для разрешения использования шаблонов при выводе многоугольников служит команда glEnable(GL_POLYGON_STIPPLE).

Свой вектор нормали для каждой вершины можно задать при помощи одной из следующих процедур:

```
    glNormal3{b s i d f}(TYPE nx, TYPE ny, TYPE nz);
glNormal3{b s I d f}v(const TYPE * v).
```

В версиях с суффиксами *b, s* или *i* значения аргументов масштабируются в отрезок [-1, 1].

В качестве примера приведѐм процедуру, строящую прямоугольный параллелепипед с рѐбрами, параллельными координатным осям, по диапазонам изменения *x*, *y* и *z*.

```
    #include <windows.h> #include <gl\gl.h>
    drawBox(GLfloat x1, GLfloat x2, GLfloat y1,
    GLfloat y2, GLfloat z1, GLfloat z2) {
    glBegin ( GL_POLYGON ); glNormal3f ( 0.0, 0.0,
1.0 ); glVertex3f ( x1, y1, z2 ); glVertex3f ( x2,
y1, z2 ); glVertex3f ( x2, y2, z2 ); glVertex3f ( x1,
y2, z2 ); glEnd (); glBegin ( GL_POLYGON );
glNormal3f ( 0.0, 0.0, -1.0 ); glVertex3f ( x2, y1,
z1 ); glVertex3f ( x1, y1, z1 ); glVertex3f ( x1, y2,
z1 ); glVertex3f ( x2, y2, z1 ); glEnd (); glBegin (
GL_POLYGON ); glNormal3f ( -1.0, 0.0, 0.0 );
glVertex3f ( x1, y1, z1 ); glVertex3f ( x1, y1, z2 );
glVertex3f ( x1, y2, z2 ); glVertex3f ( x1, y2, z1 );
glEnd ();
    glBegin ( GL_POLYGON ); glNormal3f ( 1.0, 0.0,
0.0 ); glVertex3f ( x2, y1, z2 ); glVertex3f ( x2,
y1, z1 ); glVertex3f ( x2, y2, z1 ); glVertex3f ( x2,
y2, z2 ); glEnd (); glBegin ( GL_POLYGON );
glNormal3f ( 0.0, 1.0, 0.0 ); glVertex3f ( x1, y2, z2
); glVertex3f ( x2, y2, z2 ); glVertex3f ( x2, y2, z1
); glVertex3f ( x1, y2, z1 ); glEnd (); glBegin (
GL_POLYGON ); glNormal3f ( 0.0, -1.0, 0.0 );
glVertex3f ( x2, y1, z2 ); glVertex3f ( x1, y1, z2 );
glVertex3f ( x1, y1, z1 ); glVertex3f ( x2, y1, z1 );
glEnd ();
    }
```

10.4. Преобразование объектов в пространстве

10.4.1. Преобразования в пространстве

В процессе построения изображения координаты вершин подвергаются определенным преобразованиям. Подобным преобразованиям подвергаются заданные векторы нормали.

Изначально камера находится в начале координат и направлена вдоль отрицательного направления оси *Oz*.

В OpenGL существуют две матрицы, последовательно применяющиеся в преобразовании координат. Одна из них – ***матрица моделирования*** (modelview matrix), а другая – ***матрица проецирования*** (projection matrix). Первая служит для задания положения объекта и его ориентации, вторая отвечает за выбранный способ проецирования. OpenGL поддерживает два типа проецирования – параллельное и перспективное.

Существует набор различных процедур, умножающих текущую матрицу (моделирования или проецирования) на матрицу выбранного геометрического преобразования.

Текущая матрица задается при помощи процедуры `gIMatrixMode(GLenum mode)`. Параметр mode может принимать значения `GL_MODELVIEW`, `GL_TEXTURE` или `GL_PROJECTION`, позволяя выбирать в качестве текущей матрицы матрицу моделирования (видовую матрицу), матрицу проецирования или матрицу преобразования текстуры.

Процедура `gILoadldenity()` устанавливает единичную текущую матрицу.

Обычно задание соответствующей матрицы начинается с установки единичной и последовательного применения матриц геометрических преобразований.

Преобразование переноса задается процедурой `glTranslate{f d}(TYPE x, TYPE y, TYPE z)`, обеспечивающей перенос объекта на величину (*x*, *y*, *z*).

Преобразование поворота задаѐтся процедурой `glRotate{f d}(TYPE angle, TYPE x, TYPE y, TYPE z)`, обеспечивающей поворот на угол angle в направлении против часовой стрелки вокруг прямой с направляющим вектором (*x*, *y*, *z*).

Преобразование масштабирования задаѐтся процедурой `glScale{f d}(TYPE x, TYPE y, TYPE z)`.

Если указано несколько преобразований, то текущая матрица в результате будет последовательно умножена на соответствующие матрицы.

10.4.2. Получение проекций

Видимым объемом при перспективном преобразовании в OpenGL является усеченная пирамида.

Для задания перспективного преобразования в OpenGL служит процедура `gIFrustrum(GLdouble teft, GLdoubte right, GLdouble bottom, GLdouble top, GLdouble near,`

`GLdoubte far)`.

Параметры определяют плоскости, по которым проводится отсечение. Величины near и far должны быть неотрицательными.

Иногда для задания перспективного преобразования удобнее воспользоваться следующей процедурой из библиотеки утилит `gluPerspective(GLdouble fovy, GLdouble aspect,`

`GLdouble zNear, GLdouble zFar)`.

Эта процедура создает матрицу для задания симметричного поля зрения и умножает текущую матрицу на неѐ. Здесь `fovy` – угол зрения камеры в плоскости *Oxz*, лежащей в диапазоне [0, 180]. Параметр `aspect` – отношение ширины области к еѐ высоте, `zNear` и `zFar` – расстояния вдоль отрицательного направления оси *Oz*, определяющие ближнюю и дальнюю плоскости отсечения.

Существует ещѐ одна удобная функция для задания перспективного проецирования `gluLookAt (GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GIdouble centerX, GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY, GLdouble up2).`

Вектор (`eyeX, eyeY, eyeZ`) задаѐт положение наблюдателя, вектор (`cenlerX, centerY, centerZ`) – направление на центр сцены, а вектор (`upX, upY, upZ`) – направление вверх.

В случае параллельного проецирования видимым объемом является прямоугольный параллелепипед. Для задания параллельного проецирования служит процедура `glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far).`

Параметры `left` и `right` определяют координаты левой и правой вертикальных плоскостей отсечения, а `bottom` и `top` – нижней и верхней горизонтальных плоскостей.

Следующим шагом в задании проецирования (после выбора параллельного или перспективного преобразования) является задание области в окне, в которую будет помещено получаемое изображение. Для этого служит процедура `gIViewport(GLint x, GLint y,`

`GLsizei width, GLsizei height).`

Здесь (`x, y`) задаѐт нижний левый угол прямоугольной области в окне, а `width` и `height` являются еѐ шириной и высотой.

OpenGL содержит стек матриц для каждого из трѐх типов преобразований. При этом текущую матрицу можно поместить в стек или извлечь матрицу из стека и сделать еѐ текущей.

Для помещения текущей матрицы в стек служит процедура `gIPushMatrix()`, для извлечения матрицы из стека – процедура `glPopMatrix().`

10.5. Задание моделей закрашивания

Линия или заполненная грань могут быть нарисованы одним цветом (плоское закрашивание, GL_FLAT) или путѐм интерполяции цветов в вершинах (закрашивание Гуро, GL_SMOOTH).

Для задания режима закрашивания служит процедура glShadeModel(GLenum mode), где параметр mode принимает значение GL_SMOOTH или GL_FLAT.

10.6. Освещение

OpenGL использует модель освещѐнности, в которой свет приходит из нескольких источников, каждый из которых может быть включѐн или выключен. Кроме того, существует еще общее фоновое (ambient) освещение.

Для правильного освещения объектов необходимо для каждой грани задать материал, обладающий определенными свойствами. Материал может испускать свой собственный свет, рассеивать падающий свет во всех направлениях (диффузное отражение) или, подобно зеркалу, отражать свет в определенных направлениях.

Пользователь может определить до восьми источников света и их свойства, такие, как цвет, положение и направление. Для задания этих

свойств служит процедура gllight{i f}[v](GLenum light, GLenum pname, TYPE param), которая задаѐт параметры для источника света light, принимающего значения GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT7. Параметр pname определяет характеристику источника света, которая задается последним параметром.

Для использования источников света расчѐт освещенности следует разрешить командой glEnable(GL_LGHTING), а применение соответствующего источника света разрешить (включить) командой glEnable, например: glEnable(GL_LIGHT0).

Источник света можно рассматривать как имеющий вполне определенные координаты и светящий во всех направлениях или как направленный источник, находящийся в бесконечно удаленной точке и светящий в заданном направлении (x, y, z).

Если параметр w в команде GL_POSITION равен нулю, то соответствующий источник света – направленный и светит в направлении (x, y, z). Если же w отлично от нуля, то это позиционный источник света, находящийся в точке с координатами (x/w, y/w, z/w).

Заданием параметров GL_SPOT_CUTOFF и GL_SPOT_DIRECTION можно создавать источники света, которые будут иметь коническую

направленность. По умолчанию значение параметра GL_SPOT_CUTOFF равно 180°, т. е. источник светит во всех направлениях с равной интенсивностью. Параметр GL_SPOT_CUTOFF определяет максимальный угол от направления источника, в котором распространяется свет от него. Он может принимать значение 180° (не конический источник) или от 0 до 90°.

Интенсивность источника с расстоянием, вообще говоря, убывает (параметры этого убывания задаются при помощи параметров (GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION и GL_QUADRATIC_ATTENUATION). Только собственное свечение материала и глобальная фоновая освещенность с расстоянием не ослабевают.

Глобальное фоновое освещение можно задать при помощи команды gtLightModel{i f} [v] (GL_LIGHT_MODEL_AMBIENT ambientColor).

Местонахождение наблюдателя оказывает влияние на блики на объектах. По умолчанию при расчѐтах освещѐнности считается, что наблюдатель находится в бесконечно удалѐнной точке, т. е. направление на наблюдателя постоянно для любой вершины. Можно включить более реалистическое освещение, когда направление на наблюдателя будет вычисляться для каждой вершины отдельно. Для этого служит команда glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE).

Для задания освещения как лицевых, так и нелицевых граней (для нелицевых граней вектор нормали переворачивается) служит следующая команда glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE), причѐм существует возможность отдельного задания свойств материала для каждой из сторон.

Свойства материала, из которого сделан объект, задаются при помощи процедуры glMaterial{i f}[v](GLenum face, GLenum pname, TYPE param).

Параметр face указывает, для какой из сторон грани задается свойство, и принимает одно из следующих значений: GL_BACK, GL_FRONT_AND_BACK, GL_FRONT.

Параметр pname указывает, какое именно свойство материала задается.

Расчѐт освещѐнности в OpenGL не учитывает затенения одних объектов другими.

10.7. Полупрозрачность. Использование α-канала

До сих пор не рассматривался α-канал (в RGBA-представлении цвета) и значение соответствующей компоненты во всех примерах всегда равнялось единице. Задавая значения, отличные от единицы, можно смешивать цвет выводимого пикселя с цветом пикселя, уже находящегося в соответствующем месте на экране, создавая тем самым эффект прозрачности.

При этом наиболее естественно думать об этом, считая что RGBкомпоненты задают цвет фрагмента, α-значение – его непрозрачность (степень поглощения фрагментом проходящего через него света). Так, если у стекла установить значение, равное 0.2, то в результате вывода цвет получившегося фрагмента будет на 20 % состоять из собственного цвета стекла и на 80 % – из цвета фрагмента под ним.

Для использования α-канала необходимо сначала разрешить режим прозрачности и смешения цветов командой `gEnable(GL_BLEND)`.

В процессе смешения цветов цветовые компоненты выводимого фрагмента $R_sG_sB_sA_s$ смешиваются с цветовыми компонентами уже выведенного фрагмента $R_dG_dB_dA_d$ по формуле

($R_sS_r+R_dD_r$, $G_sS_g+G_dD_g$, $B_sS_b+B_dD_b$, $A_sS_a+A_dD_a$), где (S_r, S_g, S_b, S_a) и (D_r, D_g, D_b, D_a) – коэффициенты смешения.

Для задания связи этих коэффициентов с α-значениями используется функция `glBlendFunc(GLenum sfactor, GLenum dfactor)`.

Здесь параметр `sfactor` задаèт то, как нужно вычислять коэффииенты (S_r, S_g, S_b, S_a), а параметр `dfactor` – коэффициенты (D_r, D_g, D_b, D_a).

10.8. Наложение текстуры

Текстурирование позволяет наложить изображение на многоугольник и вывести этот многоугольник с наложенной на него текстурой, соответствующим образом преобразованной. OpenGL поддерживает одно- и двумерные текстуры и различные способы наложения (применения) текстуры.

Для использования текстуры надо сначала разрешить одно- или двумерное текстурирование при помощи команд `glEnable(GL_TEXTURE1D)` или `glEnable(GL_TEXTURE_2D)`.

Для задания двумерной текстуры служит процедура `glTexlmage2D(GLenum target, GLint level, GLint component, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels)`.

Параметр `target` зарезервирован для будущего использования и в текущей версии OpenGL должен быть равен `GL_TEXTURE_2D`. Параметр

`level` используется в том случае, если задается несколько разрешений данной текстуры. При ровно одном разрешении он должен быть равным нулю.

Следующий параметр `component` – целое число от одного до четырех, показывающее, какие из RGBA-компонент выбраны для использования. Значение 1 выбирает компоненту *R*, значение 2 выбирает *R* и *A* компоненты, 3 соответствует *R, G* и *B*, а 4 соответствует компонентам RGBA.

Параметры `width` и `height` задают размеры текстуры, `border` задает размер границы (бортика), обычно равный нулю. Как параметр `width`, так и параметр `height`, должны иметь вид $2^n + 2b$, где n – целое число, а b – значение параметра `border`. Максимальный размер текстуры зависит от реализации OpenGL, но он не менее 64 на 64.

При текстурировании OpenGL поддерживает использование пирамидального фильтрования (mip-mappping). Для этого необходимо иметь текстуры всех промежуточных размеров, являющихся степенями двух, вплоть до 1□1, и для каждого такого разрешения вызвать `glTexImage2D` с соответствующими параметрами `level`, `width`, `height` и `image`. Кроме того, необходимо задать способ фильтрования, который будет применяться при выводе текстуры.

Под фильтрованием здесь подразумевается способ, которым для каждого пикселя будет выбираться подходящий элемент текстуры (тексель). При текстурировании возможна ситуация, когда одному пикселю соответствует небольшой фрагмент текселя (увеличение) или же, наоборот, когда одному пикселю соответствует целая группа текселей (уменьшение).

Способ выбора соответствующего текселя, как для увеличения, так и для уменьшения (сжатия) текстуры необходимо задать отдельно. Для этого используется процедура `glTexParameteri(GL_TEXTURE_2D, GLenum p1, GLenum p2)`, где параметр `p1` показывает, задается ли фильтр для сжатия или для растяжения текстуры, принимая значение

`GL_TEXTURE_MIN_FLITER` или `GL_TEXTURE_MAG_FILTER`.

Параметр `p2` задает способ фильтрования.

При использовании пирамидального фильтрования помимо выбора текселя на одном слое текстуры появляется возможность либо выбрать один соответствующий слой, либо проинтерполировать результаты выбора между двумя соседними слоями. Для правильного применения текстуры каждой вершине следует задать соответствующие ей координаты текстуры при помощи процедуры `glTexCoord{1 2 3 4}{s i f d}[v](TYPE coord, ...)`.

Этот вызов задаèт значения индексов текстуры для последующей команды gIVertex.

Если размер грани больше, чем размер текстуры, то для циклического повторения текстуры служат команды gITexParameteri (GL_TEXTURE_2D, GL_TEXTURE_S_WRAP, GL_REPEAT), gITexParameteri(GL_TEXTURE_2D,

GL_TEXTURE_T_WRAP, GL_REPEAT).

Координаты текстуры обычно подвергаются преобразованию при помощи матрицы текстурирования. По умолчанию она совпадает с единичной матрицей, но пользователь сам имеет возможность задать преобразования текстуры, например следующим образом: gIMatrixMode(GL_TEXTURE); gIRotatef(...};

gIMatrixMode(GL_MODELV1EW);

При выводе текстуры OpenGL может использовать линейную интерполяцию (аффинное текстурирование) или же точно учитывать перспективное искажение. Для задания точного текстурирования служит команда glHint(GL_PERSPECTIVE_CORRECTION_HINT,

GL_NICEST).

Если качество не играет большой роли, а нужна высокая скорость рендеринга, то в качестве последнего аргумента следует использовать константу GL_FASTEST.

Описанный выше способ работ с текстурами используется в OpenGL версии 1.0. В более новых версиях OpenGL, начиная с версии 1.1, введены дополнительные функции, повышающие удобство работы с текстурами. В OpenGL 1.0 процедуру glTexImage2D необходимо вызывать всякий раз, когда нужно сменить текущую текстуру. Это достаточно медленный способ. В OpenGL 1.1 имеется возможность присваивать имена текстурам и затем сменять текущую текстуру только указанием имени новой текстуры, без повторной еè загрузки в память процедурой glTexImage2D.

Имя текстуры представляет собой уникальное значение типа GLuint. Перед использованием текстуры необходимо присвоить ей имя. Имена текстур можно сгенерировать при помощи процедуры glGenTextures(GLsizei n, GLuint *textures).

Параметр n определяет количество текстур, для которых необходимо сгенерировать имена. Параметр textures является указателем на массив переменных типа GLuint, состоящим из *n* элементов. После вызова процедуры

каждый элемент массива будет содержать уникальное имя текстуры, которое затем может быть использовано при работе с текстурами.

Для выбора текущей (активной) текстуры используется функция `glBindTexture(GLenum target, GLuint texture)`.

Параметр `target` определяет тип текстуры (одномерная – `GL_TEXTURE_1D` или двумерная – `GL_TEXTURE_2D`). На практике более часто используются двумерные текстуры, которые представляют собой обычные двумерные изображения. Параметр `texture` определяет имя текстуры, которую необходимо сделать активной.

После того, как установлена активная текстура, можно вызвать процедуру `glTexImage2D` и задать параметры текстуры, а также сами еѐ тексели. После вызова процедуры `glTexImage2D` текстура готова к применению.

Для того чтобы наложить текстуру на объект или многоугольник достаточно установить активную текстуру (процедура `glBindTexture`) и определить текстурные координаты при помощи процедуры `glTexCoord`.

Достоинство использования функций OpenGL 1.1 для работы с текстурами заключается не только в более высоком быстродействии по сравнению с использованием процедуры `glTexImage2D`, но и в повышенном удобстве работы с текстурами. Создав массив текстурных имѐн можно работать одновременно с несколькими текстурами, вызывая лишь функцию `glBindTexture` по мере необходимости.

11. Аппаратные средства машинной графики

Математическое и программное обеспечение компьютерной графики нельзя рассматривать в отрыве от аппаратных средств, применяемых на различных этапах работы с изображениями. Все эти средства принято делить на три большие группы:

- устройства ввода (сканеры, дигитайзеры/графические планшеты, цифровые фото- и видеокамеры);
- устройства вывода (мониторы, принтеры, плоттеры, цифровые проекторы);
- устройства обработки (графические ускорители, кодеры MPEG и др.).

Поскольку детальная информация о принципах действия, параметрах и применении вышеперечисленных устройств дается в соответствующих разделах других курсов, например «Периферийные устройства ЭВМ»,

подробнее остановимся только на аппаратных средствах первой группы, на рынке которых наблюдается в настоящий момент бум новых средств и технологий.

11.1. Устройства ввода

Существуют различные технические средства, осуществляющие процесс преобразования изображений в цифровую форму, например: сканеры, дигитайзеры (графические планшеты), цифровые фото- и видеокамеры. В каждом конкретном случае важно правильно выбрать нужное устройство, руководствуясь его техническими характеристиками, для получения оцифрованного изображения с требуемой детальностью и цветовой гаммой.

11.2. Сканеры

Сканером называется устройство, позволяющее вводить в компьютер образы изображений, представленных в виде текста, рисунков, слайдов, фотографий или другой графической информации. Традиционно сканеры служили для решения специализированных задач: ввода и запоминания изображений в настольных издательских системах, организации хранения текстовых документов в юридических фирмах и т. п. С появлением почти у каждого собственных страниц Web и повсеместным распространением цветных струйных принтеров сканеры быстро превращаются в универсальные настольные средства подобно принтерам и модемам.

Принцип действия и виды сканеров

Принцип действия практически всех типов сканеров един. Он основан на том, что направленным лучом освещаются отдельные точки исходного изображения (оригинала) и отраженный в результате луч воспринимается фоточувствительным приемником, где информация о «цвете» точки интерпретируется как конкретное численное значение, которое через определенный интерфейс передается в компьютер.

Как правило, светочувствительные элементы объединяют в матрицу, для того, чтобы сканировать одновременно целый участок оригинала.

По механизму перемещения матрицы светочувствительных элементов относительно оригинала выделяют следующие типы сканеров:

1) ***Планшетный сканер*** (Flatbed Scanner) – сканер, в котором оригинал кладется на стекло и сканируется при помощи подвижной линейной матрицы (рис. 11.1). Размеры матрицы и системы фокусировки подобраны так, чтобы вести сканирование листа по всей ширине.

Рис. 11.1. Планшетный сканер 2) ***Ручной сканер*** (Handheld Scanner) – портативный сканер, в котором сканирование осуществляется путем ручного перемещения сканера по оригиналу. По принципу действия такой сканер аналогичен планшетному. Ширина области сканирования не более 15 см.

3) ***Барабанный сканер*** (Drum Scanner) – сканер, в котором оригинал закрепляется на вращающемся барабане. При этом сканируется точечная область изображения, а сканирующая головка движется вдоль барабана на очень маленьком расстоянии от оригинала.

Основные характеристики

При выборе конкретной модели сканера необходимо учитывать ряд характеристик, связанных с техническими возможностями модели.

Разрешение (Resolution) – число точек или растровых ячеек, из которых формируется изображение, на единицу длины или площади. Чем больше разрешение устройства, тем более мелкие детали могут быть воспроизведены.

Аппаратное/оптическое разрешение сканера (Hardware/optical Resolution) – это одна из основных характеристик сканера, напрямую связанная с плотностью размещения чувствительных элементов на матрице сканера. Измеряется в количестве пикселов на квадратный дюйм изображения – PPI (Pixel Per Inch). Пример: 300□300 ppi.

Интерполированное разрешение (Interpolated Resolution) – разрешение изображения, полученного при помощи математической обработки исходного изображения. С улучшением качества имеет мало общего. Часто служит рекламной уловкой для неподготовленных пользователей. Пример: 600□1200 (9600) ppi (цифра 600 – максимальное оптическое разрешение, 1200 – разрешение "двойного шага", 9600 – максимальное интерполированное разрешение).

Глубина цвета (color depth) – количество разрядов каждого пиксела в цифровом изображении, в том числе выдаваемом сканером. Описывает максимальное количество цветов, воспроизводимое сканером в виде степени числа 2. Одному разряду соответствует чернобелое изображение, 8-ми – серое полутоновое, 16-ти – цветное, 24цветное изображение – наиболее близкое к

человеческому восприятию (модель RGB), 36bit и больше – полноцветное изображение с высокой достоверностью цветопередачи, предназначенное для профессиональной работы, чаще всего в издательском деле.

Фирмы-производители

На мировом рынке представлено достаточно большое число фирмпроизводителей сканеров. Наиболее популярные модели производят Hewlett-Packard, Agfa, Canon, Mustek.

11.3. Дигитайзеры

Дигитайзер (графический планшет) — это устройство, предназначенное для оцифровки изображений, применяемое для создания на компьютере рисунков и набросков (рис. 11.2). Художник создает изображение на экране, но его рука водит пером по планшету. Как правило, планшет используют профессиональные художники для более точной обработки (создания) изображений.

Рис. 11.2. Графический планшет

Кроме того, дигитайзер можно использовать просто как аналог манипулятора «мышь».

Принцип действия

Дигитайзер, или планшет, как его часто называют, состоит из двух основных элементов: основания и курсора, двигающегося по его поверхности. Принцип действия дигитайзера основан на фиксации местоположения курсора с помощью встроенной в планшет сетки. При нажатии на кнопку курсора его местоположение на поверхности планшета фиксируется, а его координаты передаются в компьютер. Сетка состоит из проволочных или печатных проводников с довольно большим расстоянием между соседними проводниками (от трех до шести мм).

Основные характеристики

Механизм регистрации позволяет получить шаг считывания информации намного меньше шага сетки (до 100 линий на мм). Шаг считывания информации называется ***разрешением дигитайзера.***

Шаг считывания регистрирующей сетки является физическим пределом разрешения дигитайзера. Мы говорим о пределе разрешения, потому что следует различать разрешение как характеристику прибора и ***программно-задаваемое разрешение***, что есть переменная величина в настройке дигитайзера.

Следует отметить, что в работе планшетов возможны помехи со стороны излучающих устройств, в частности мониторов. Независимо от принципа регистрации существует погрешность в определении координат курсора, именуемая ***точностью дигитайзера***. Эта величина зависит от типа дигитайзера и от конструкции его составляющих. Точность существующих планшетов колеблется в пределах от 0.005 дюйма до 0.03 дюйма.

Важной характеристикой дигитайзера является регистрируемое ***число степеней нажатия*** электронного пера. В существующих моделях эта величина может изменяться в пределах от одного до 256-ти. Программа-обработчик использует эту величину, устанавливая в зависимости от нее, например, толщину проводимой линии (чем сильнее нажим, тем толще линия).

Фирмы-производители

Наиболее популярны модели следующих фирм: CalComp, NUMONICS, WACOM.

11.4. Цифровые фотокамеры

Цифровая камера получает изображения, обрабатывает их и хранит в цифровом формате. Вместо пленки она использует встроенную или сменную полупроводниковую память, чтобы хранить снимки. Она обладает теми же основными свойствами, что и нормальная фотокамера, и, помимо этого, может соединяться с компьютером, телевизором или принтером. Поскольку обработка кадра происходит непосредственно в камере, пользователь может сразу проверить правильность полученного изображения, напечатать его или послать по электронной почте.

Рис. 11.3. Цифровая фотокамера Достоинства цифровых фотокамер:

- Изображение обрабатывается сразу же. Большинство цифровых камер имеют маленький цветной экран, на котором можно немедленно увидеть снимок, который был сделан. Это позволяет отказаться от неудачного кадра и записать на его место другой.
- Изображения хранятся в электронной памяти, циклы записистирания информации в которую могут повторяться практически бесконечно. Пропадает необходимость каждый раз покупать пленку, реактивы для ее проявки.
- Упростился процесс ввода фотографий в компьютер. Теперь не нужно сканировать изготовленные обычным образом фотографии. Вы просто подключаете цифровую камеру с помощью кабеля или PC-карты к ПК и переписываете нужные снимки на жесткий диск.
- Цифровая камера позволяет проводить множество манипуляций с фотографиями.

Недостатки цифровых фотокамер:

- Низкое разрешение. Приемлемым для качественной печати разрешением (свыше 300 dpi) обладают на сегодняшний день только профессиональные цифровые камеры со стоимостью, делающей их недоступными для массового пользователя.
- Высокая, по сравнению с обычными фотокамерами такого же класса, цена.
- Действительно, качественная печать цифровых фотографий требует чаще всего специального оборудования и имеет высокую себестоимость за счет дорогих расходных материалов.

Принцип действия

Принцип действия цифровой фотокамеры аналогичен принципу действия видеокамеры и состоит в следующем. Пучок лучей света от объекта съемки,

проходя через линзу (или систему линз) объектива и диафрагму, попадает на матрицу CCD (Charged Coupled Device).

Матрица CCD или, как ее еще называют, ПЗС (преобразователь светсигнал) представляет собой прямоугольную матрицу из светочувствительных элементов. Луч света, попадая на чувствительный элемент, преобразуется в аналоговый электрический сигнал. Аналоговые сигналы от CCD преобразуются в цифровые, обрабатываются и записываются в память. Преобразование сигналов в цифровую форму производится с помощью аналого-цифрового преобразователя ADC.

Кроме CCD, ADC и памяти в электрическую схему цифровой фотокамеры входят процессор DSP, который формирует изображение из цифровых потоков, и конвертор JPEG, сжимающий изображения для увеличения количества хранимых кадров.

Сменная память используется в цифровых камерах для увеличения количества сохраняемых кадров и, чаще всего, представляет собой Flash-карту памяти.

Фирмы-производители

В настоящее время на рынке работают десятки известнейших фирм-производителей как традиционного фотооборудования и материалов (Kodak, Konica, Nikon, Fuji, Agfa, Olympus и др.), так и компьютерной периферии и прочего электронного оборудования

(Hewlett-Packard, Seiko Epson, Sony, Ricoh, Mustek, UMAX, LG Electronics, Minolta и др.).

Литература

1. Божко А. Н. Компьютерная графика : [учебное пособие для вузов] / А. Н. Божко, Д. М. Жук, В. Б. Маничев. – М: Изд-во МГТУ им. Н. Э. Баумана, 2007. – 392 с.
2. Гринченко В.Т., Мацыпура В.Т., Снарский А.А. Введение в нелинейную динамику. Хаос и фракталы. - 2-е изд. Издательство: ЛКИ, 2007 г. — 264 с.
3. Дегтярев В. М. Компьютерная геометрия и графика. – М: Академия 2010 г. 192 с.
4. Краснов М. В. OpenGL. Графика в проектах Delphi. — СПб.: БХВПетербург, 2001. — 352 с.
5. М. Домасев, С. Гнатюк. Цвет. Управление цветом, цветовые расчеты и измерения. – СПб: Питер 2009 г. 224 с.
6. Никулин Е. А. Компьютерная геометрия и алгоритмы машинной графики. — СПб: БХВ-Петербург, 2003. — 560 с.
7. Роджерс Д. Алгоритмические основы машинной графики: Пер. с англ. - М.: Мир, 1989. – 512 с.
8. Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. – М.: Мир, 2001. – 604 с.
9. Тихомиров Ю. Программирование трехмерной графики. – СПб: BHV – Санкт-Петербург, 1998. – 256 с.
10. Фоли Дж., вэн Дэм А. Основы интерактивной машинной графики: В 2-х кн., Кн. 1. / Пер. с англ. – М.: Мир, 1985 – 368 с.
11. Фоли Дж., вэн Дэм А. Основы интерактивной машинной графики: В 2-х кн., Кн. 2. / Пер. с англ. – М.: Мир, 1985 – 368 с.
12. Шикин Е. В., Боресков А. В. Компьютерная графика. Полигональные модели. – М.: ДИАЛОГ-МИФИ, 2000. – 464 с.

Содержание

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «ГРАФИЧЕСКИЕ ТЕХНОЛОГИИ И ФОРМАТЫ ГРАФИЧЕСКИХ ДАННЫХ»

для обучающихся по направлению 09.03.02 Информационные системы и технологии всех форм обучения

Составители:
Кузовкин Алексей Викторович
Суворов Александр Петрович
Золототрубова Юлия Сергеевна

Компьютерный набор А.В. Кузовкина

Подписано к изданию_____.
Уч.-изд. л. ____.

ФГБОУ ВО «Воронежский государственный технический университет»
396026 Воронеж, Московский просп., 14

Участок оперативной полиграфии издательства ВГТУ
396026 Воронеж, Московский просп., 14