Лекция 1. Общее введение в компьютерную графику

Предмет и области применения компьютерной графики. Краткая история развития компьютерной графики. Технические средства поддержки компьютерной графики: ЭЛТ, устройства ввода, видеоадаптер, графопостроители, принтеры, сканеры. Программные средства поддержки компьютерной графики: драйверы устройств, библиотеки графических программ, специализированные графические системы и пакеты программ

Предмет и область применения компьютерной графики

Компьютерная графика - это область информатики, которая охватывает все стороны формирования изображений с помощью компьютера. Появившись в 1950-х годах, она поначалу давала возможность выводить лишь несколько десятков отрезков на экране. В наши дни средства компьютерной графики позволяют создавать реалистические изображения, не уступающие фотографическим снимкам. Создано разнообразное аппаратное и программное обеспечение для получения изображений самого различного вида и назначения - от простых чертежей до реалистических образов естественных объектов. Компьютерная графика используется практически во всех научных и инженерных дисциплинах для наглядности восприятия и передачи информации. Применение ее для подготовки демонстрационных слайдов уже считается нормой. Трехмерные изображения используются в медицине (компьютерная томография), картографии, полиграфии, геофизике, ядерной физике и других областях. Телевидение и другие отрасли индустрии развлечений используют анимационные средства компьютерной графики (компьютерные игры, фильмы). Общепринятой практикой считается также использование компьютерного моделирования при обучении пилотов и представителей других профессий (тренажеры). Знание основ компьютерной графики сейчас необходимо и инженеру, и ученому.

Конечным результатом применения средств компьютерной графики является изображение, которое может использоваться для различных целей. Поскольку наибольшее количество информации человек получает с помощью зрения, уже в древние времена появились схемы и карты, используемые при строительстве, в географии и в астрономии.

Современная компьютерная графика - это достаточно сложная, основательно проработанная и разнообразная научно-техническая дисциплина. Некоторые ее разделы, такие как геометрические преобразования, способы описания кривых и поверхностей, к настоящему времени уже исследованы достаточно полно. Ряд областей продолжает активно развиваться: методы растрового сканирования, удаление невидимых линий и поверхностей, моделирование цвета и освещенности, текстурирование, создание эффекта прозрачности и полупрозрачности и др.

Сфера применения компьютерной графики включает четыре основных области.

1. Отображение информации

Проблема представления накопленной информации (например, данных о климатических изменениях за продолжительный период, о динамике популяций животного мира, об экологическом состоянии различных регионов и т.п.) лучше всего может быть решена посредством графического отображения.

Ни одна из областей современной науки не обходится без графического представления информации. Помимо визуализации результатов экспериментов и анализа данных натурных наблюдений существует обширная область математического моделирования процессов и явлений, которая просто немыслима без графического вывода. Например, описать процессы, протекающие в атмосфере или океане, без соответствующих наглядных картин течений или полей температуры практически невозможно. В геологии в результате обработки трехмерных натурных данных можно получить геометрию пластов, залегающих на большой глубине.

В медицине в настоящее время широко используются методы диагностики, использующие компьютерную визуализацию внутренних органов человека. Томография (в частности, ультразвуковое исследование) позволяет получить трехмерную информацию, которая затем подвергается математической обработке и выводится на экран. Помимо этого применяется и двумерная графика: энцефалограммы, миограммы, выводимые на экран компьютера или графопостроитель.

2. Проектирование

В строительстве и технике чертежи давно представляют собой основу проектирования новых сооружений или изделий. Процесс проектирования с необходимостью является итеративным, т.е. конструктор перебирает множество вариантов с целью выбора оптимального по каким-либо параметрам. Не последнюю роль в этом играют требования заказчика, который не всегда четко представляет себе конечную цель и технические возможности. Построение предварительных макетов - достаточно долгое и дорогое дело. Сегодня существуют развитые программные средства автоматизации проектно-конструкторских работ (САПР), позволяющие быстро создавать чертежи объектов, выполнять прочностные расчеты и т.п. Они дают возможность не только изобразить проекции изделия, но и рассмотреть его в объемном виде с различных сторон. Такие средства также чрезвычайно полезны для дизайнеров интерьера, ландшафта.

3. Моделирование

Под моделированием в данном случае понимается имитация различного рода ситуаций, возникающих, например, при полете самолета или космического аппарата, движении автомобиля и т.п. В английском языке это лучше всего передается термином simulation. Но моделирование используется не только при создании различного рода тренажеров. В телевизионной рекламе, в научно-популярных и других фильмах теперь синтезируются движущиеся объекты, визуально мало уступающие тем, которые могут быть получены с помощью кинокамеры. Кроме того, компьютерная графика предоставила киноиндустрии возможности создания спецэффектов, которые в прежние годы были попросту невозможны. В последние годы широко распространилась еще одна сфера применения компьютерной графики - создание виртуальной реальности.

4. Графический пользовательский интерфейс

На раннем этапе использования дисплеев как одного из устройств компьютерного вывода информации диалог "человек-компьютер" в основном осуществлялся в алфавитно-цифровом виде. Теперь же практически все системы программирования применяют графический интерфейс. Особенно впечатляюще выглядят разработки в области сети Internet. Существует множество различных программ-браузеров, реализующих в том или ином виде средства общения в сети, без которых доступ к ней трудно себе представить. Эти программы работают в различных операционных средах, но реализуют, по существу, одни и те же функции, включающие окна, баннеры, анимацию и т.д.

В современной компьютерной графике можно выделить следующие основные направления: изобразительная компьютерная графика, обработка и анализ

изображений, анализ сцен (перцептивная компьютерная графика), компьютерная графика для научных абстракций (когнитивная компьютерная графика, т.е. графика, способствующая познанию).

Изобразительная компьютерная графика своим предметом имеет синтезированные изображения. Основные виды задач, которые она решает, сводятся к следующим:

- построение модели объекта и формирование изображения;
- преобразование модели и изображения;
- идентификация объекта и получение требуемой информации.

Обработка и анализ изображений касаются в основном дискретного (цифрового) представления фотографий и других изображений. Средства компьютерной графики здесь используются для:

- повышения качества изображения;
- оценки изображения - определения формы, местоположения, размеров и других параметров требуемых объектов;
- распознавания образов - выделения и классификации свойств объектов (при обработке аэрокосмических снимков, вводе чертежей, в системах навигации, обнаружения и наведения).

Анализ сцен связан с исследованием абстрактных моделей графических объектов и взаимосвязей между ними. Объекты могут быть как синтезированными, так и выделенными на фотоснимках. К таким задачам относятся, например, моделирование "машинного зрения" (роботы), анализ рентгеновских снимков с выделением и отслеживанием интересующего объекта (внутреннего органа), разработка систем видеонаблюдения.

Когнитивная компьютерная графика - только формирующееся новое направление, пока еще недостаточно четко очерченное. Это - компьютерная графика для научных абстракций, способствующая рождению нового научного знания. Технической основой для нее являются мощные ЭВМ и высокопроизводительные средства визуализации.

Одним из наиболее ранних примеров использования когнитивной компьютерной графики является работа Ч.Страуса "Неожиданное применение ЭВМ в чистой математике" (ТИИЭР, т. 62, № 4, 1974, с.96-99). В ней показано, как для анализа сложных алгебраических кривых используется "n-мерная" доска на основе графического терминала. Пользуясь устройствами ввода, математик может легко получать геометрические изображения результатов направленного изменения параметров исследуемой зависимости. Он может также легко управлять текущими значениями параметров, "углубляя тем самым свое понимание роли вариаций этих параметров". В результате получено "несколько новых теорем и определены направления дальнейших исследований".

В настоящем курсе предполагается рассмотреть следующие вопросы:

- представление изображения в компьютерной графике;
- способы подготовки изображения к визуализации;
- методы вывода изображения на экран;
- методы работы с изображением;
- методы вычислительной геометрии.

Краткая история

В этом кратком историческом очерке мы будем упоминать алгоритмы и методы, с которыми читатель сможет познакомиться в данном курсе. Эти предварительные

упоминания не должны смущать при первом прочтении. По завершении курса можно вернуться к этому разделу и пройти его заново.

Компьютерная графика в начальный период своего возникновения была далеко не столь эффектной, какой она стала в настоящие дни. В те годы компьютеры находились на ранней стадии развития и были способны воспроизводить только самые простые контуры (линии). Идея компьютерной графики не сразу была подхвачена, но ее возможности быстро росли, и постепенно она стала занимать одну из важнейших позиций в информационных технологиях.

Первой официально признанной попыткой использования дисплея для вывода изображения из ЭВМ явилось создание в Массачусетском технологическом университете машины Whirlwind-I в 1950 г. Таким образом, возникновение компьютерной графики можно отнести к 1950-м годам. Сам же термин "компьютерная графика" придумал в 1960 г. сотрудник компании Boeing У. Феттер.

Первое реальное применение компьютерной графики связывают с именем Дж. Уитни. Он занимался кинопроизводством в 50-60-х годах и впервые использовал компьютер для создания титров к кинофильму.

Следующим шагом в своем развитии компьютерная графика обязана Айвэну Сазерленду, который в 1961 г., еще будучи студентом, создал программу рисования, названную им Sketchpad (альбом для рисования). Программа использовала световое перо для рисования простейших фигур на экране. Полученные картинки можно было сохранять и восстанавливать. В этой программе был расширен круг основных графических примитивов, в частности, помимо линий и точек был введен прямоугольник, который задавался своими размерами и расположением.

Первоначально компьютерная графика была векторной, т.е. изображение формировалось из тонких линий. Эта особенность была связана с технической реализацией компьютерных дисплеев. В дальнейшем более широкое применение получила растровая графика, основанная на представлении изображения на экране в виде матрицы однородных элементов (пикселей).

В том же 1961 г. студент Стив Рассел создал первую компьютерную видеоигру Spacewar ("Звездная война"), а научный сотрудник Bell Labs Эдвард Зэджек создал анимацию "Simulation of a two-giro gravity control system".

В связи с успехами в области компьютерной графики крупные корпорации начали проявлять к ней интерес, что в свою очередь стимулировало прогресс в области ее технической поддержки.

Университет штата Юта становится центром исследований в области компьютерной графики благодаря Д.Эвансу и А.Сазерленду, которые в это время были самыми заметными фигурами в этой области. Позднее их круг стал быстро расширяться. Учеником Сазерленда стал Э.Кэтмул, будущий создатель алгоритма удаления невидимых поверхностей с использованием Z-буфера (1978). Здесь же работали Дж.Варнок, автор алгоритма удаления невидимых граней на основе разбиения области (1969) и основатель Adobe System (1982), Дж.Кларк, будущий основатель компании Silicon Graphics (1982). Все эти исследователи очень сильно продвинули алгоритмическую сторону компьютерной графики.

В том же 1971 г. Гольдштейн и Нагель впервые реализовали метод трассировки лучей с использованием логических операций для формирования трехмерных изображений.

В 1970-е годы произошел резкий скачок в развитии вычислительной техники благодаря изобретению микропроцессора, в результате чего началась миниатюризация компьютеров и быстрый рост их производительности. И в это же время начинает

интенсивно развиваться индустрия компьютерных игр. Одновременно компьютерная графика начинает широко использоваться на телевидении и в киноиндустрии. Дж.Лукас создает отделение компьютерной графики на Lucasfilm.

В 1977 г. появляется новый журнал "Computer Graphics World".

В середине 1970-х годов графика продолжает развиваться в сторону все большей реалистичности изображений. Э.Кэтмул в 1974 г. создает первые алгоритмы текстурирования криволинейных поверхностей. В 1975 г. появляется упомянутый ранее метод закрашивания Фонга. В 1977 г. Дж.Блин предлагает алгоритмы реалистического изображения шероховатых поверхностей (микрорельефов); Ф.Кроу разрабатывает методы устранения ступенчатого эффекта при изображении контуров (антиэлайзинг). Дж.Брезенхем создает эффективные алгоритмы построения растровых образов отрезков, окружностей и эллипсов. Уровень развития вычислительной техники к этому времени уже позволил использовать "жадные" алгоритмы, требующие больших объемов памяти, и в 1978 г. Кэтмул предлагает метод Z-буфера, в котором используется область памяти для хранения информации о "глубине" каждого пикселя экранного изображения. В этом же году Сайрус и Бэк развивают алгоритмы клиппирования (отсечения) линий. А в 1979 г. Кэй и Гринберг впервые реализуют изображение полупрозрачной поверхности.

В 1980 г. Т.Уиттед разрабатывает общие принципы трассировки лучей, включающие отражение, преломление, затенение и методы антиэлайзинга. В 1984 г. группой исследователей (Горэл, Торрэнс, Гринберг и др.) была предложена модель излучательности, одновременно развиваются методы прямоугольного клиппирования областей.

В 1980-е годы появляется целый ряд компаний, занимающихся прикладными разработками в области компьютерной графики. В 1982 г. Дж.Кларк создает Silicon Graphics, тогда же возникает Ray Tracing Corporation, Adobe System, в 1986 г. компания Pixar отпочковывается от Lucasfilm.

В эти годы компьютерная графика уже прочно внедряется в киноиндустрию, развиваются приложения к инженерным дисциплинам. В 1990-е годы в связи с возникновением сети Internet у компьютерной графики появляется еще одна сфера приложения.

Здесь перечислены далеко не все серьезные шаги на пути развития графики, но более подробное знакомство с ее историей требует достаточно хорошего представления о теории и алгоритмах этой дисциплины, поэтому мы ограничиваемся лишь кратким обзором. Нетрудно заметить, что приоритет в развитии данного направления в информационных технологиях достаточно прочно удерживают американские исследователи. Но и в отечественной науке тоже были свои разработки, среди которых можно назвать ряд технических реализаций дисплеев, выполненных в разные годы:

1968, ВЦ АН СССР, машина БЭСМ-6, вероятно, первый отечественный растровый дисплей с видеопамятью на магнитном барабане;

1972, Институт автоматики и электрометрии (ИАиЭ), векторный дисплей "Символ";

1973, ИАиЭ, векторный дисплей "Дельта";

1977, ИАиЭ, векторный дисплей ЭПГ-400;

1982, Киев, НИИ периферийного оборудования, векторный дисплей СМ-7316, 4096 символов, разрешение 2048?2048;

1979-1984, Институт прикладной физики, серия растровых цветных полутоновых дисплеев "Гамма". Последние дисплеи данной серии имели таблицу цветности, поддерживали окна, плавное масштабирование.

Таким образом, в процессе развития компьютерной графики можно выделить несколько этапов.

В 1960-1970-е годы она формировалась как научная дисциплина. В это время разрабатывались основные методы и алгоритмы: отсечение, растровая развертка графических примитивов, закраска узорами, реалистическое изображение пространственных сцен (удаление невидимых линий и граней, трассировка лучей, излучающие поверхности), моделирование освещенности.

В 1980-е графика развивается более как прикладная дисциплина. Разрабатываются методы ее применения в самых различных областях человеческой деятельности.

В 1990-е годы методы компьютерной графики становятся основным средством организации диалога "человек-компьютер" и остаются таковыми по настоящее время.

Технические средства поддержки компьютерной графики

Развитие компьютерной графики во многом обусловлено развитием технических средств ее поддержки. Прежде всего это устройства вывода, каковыми являются дисплеи. В настоящее время существует несколько типов дисплеев, использующих электронно-лучевую трубку, а также дисплеи на жидкокристаллических индикаторах и другие их виды. Нас интересуют главным образом функциональные возможности дисплеев, поэтому мы не будем касаться их внутреннего устройства и электронных схем.

Возникновение компьютерной графики, как уже говорилось ранее, можно отнести к 50-м годам. Дисплейная графика на первом этапе своего развития использовала электронно-лучевые трубки (ЭЛТ) **с произвольным сканированием луча** для вывода в виде изображения информации из ЭВМ. С эксперимента в Массачусетском технологическом институте начался этап развития векторных дисплеев (дисплеев с произвольным сканированием луча).

Самым простым из устройств на ЭЛТ является **дисплей на запоминающей трубке** с прямым копированием изображения. Запоминающая трубка обладает свойством длительного времени послесвечения: изображение остается видимым в течение длительного времени (до одного часа). При выводе изображения интенсивность электронного луча увеличивают до уровня, при котором происходит запоминание следа луча на люминофоре. Сложность изображения практически не ограничена. Стирание происходит путем подачи на всю трубку специального напряжения, при котором свечение исчезает, и эта процедура занимает приблизительно 0,5 с. Поэтому изображения, полученные на экране, нельзя стереть частично, а стало быть, динамические изображения или анимация на таком дисплее невозможны. Дисплей на запоминающей трубке является векторным, или дисплеем с произвольным сканированием, т.е. он позволяет провести отрезок из одной адресуемой точки в любую другую. Его достаточно легко программировать, но уровень интерактивности у него ниже, чем у ряда дисплеев других типов ввиду низкой скорости и плохих характеристик стирания.

Следующий тип - это **векторные дисплеи с регенерацией изображения**. При перемещении луча по экрану в точке, на которую попал луч, возбуждается свечение люминофора экрана. Это свечение достаточно быстро прекращается при перемещении луча в другую позицию (обычное время послесвечения - менее 0,1 с). Поэтому, для

того чтобы изображение было постоянно видимым, приходится его "перерисовывать" (регенерировать изображение) 50 или 25 раз в секунду. Необходимость регенерации изображения требует сохранения его описания в специально выделенной памяти, называемой памятью регенерации. Само описание изображения называется дисплейным файлом. Понятно, что такой дисплей требует достаточно быстрого процессора для обработки дисплейного файла и управления перемещением луча по экрану.

Обычно серийные векторные дисплеи успевали 50 раз в секунду строить только около 3000–4000 отрезков. При большем числе отрезков изображение начинает мерцать, так как отрезки, построенные в начале очередного цикла, полностью гаснут к тому моменту, когда будут строиться последние.

Другим недостатком векторных дисплеев является малое число градаций по яркости (обычно от двух до четырех). Были разработаны, но не нашли широкого применения двух- и трехцветные ЭЛТ, также обеспечивавшие несколько градаций яркости.

В векторных дисплеях легко стереть любой элемент изображения - достаточно при очередном цикле построения удалить стираемый элемент из дисплейного файла.

Текстовый диалог поддерживается с помощью алфавитно-цифровой клавиатуры. Косвенный графический диалог, как и во всех остальных дисплеях, осуществляется перемещением перекрестия (курсора) по экрану с помощью тех или иных средств управления перекрестием - координатных колес, управляющего рычага (джойстика), трекбола (шаровой рукоятки), планшета и т.д. Отличительной чертой векторных дисплеев является возможность непосредственного графического диалога, заключающаяся в простом указании с помощью светового пера объектов на экране (линий, символов и т.д.).

Векторные дисплеи обычно подключаются к ЭВМ высокоскоростными каналами связи. Первые серийные векторные дисплеи за рубежом появились в конце 1960-х годов.

Прогресс в технологии микроэлектроники привел к тому, что с середины 1970-х годов преимущественное распространение получили дисплеи с растровым сканированием луча. Растровое устройство можно рассматривать как матрицу дискретных точек (пикселей), каждая из которых может быть подсвечена. Таким образом, оно является точечно- рисующим устройством. Поэтому любой изображаемый на экране дисплея отрезок строится с помощью последовательности точек, аппроксимирующих идеальную траекторию отрезка, подобно тому, как можно строить изображение по клеткам на клетчатом листке бумаги. При этом отрезок получается прямым только в случаях, когда он горизонтален, вертикален или направлен под углом 45 $\deg$ к горизонтали. Все другие отрезки выглядят как последовательность "ступенек" (ступенчатый эффект).

При построении изображения в растровых графических устройствах используется буфер кадра, представляющий собой большой непрерывный участок памяти компьютера. Для каждой точки в растре отводится как минимум один бит памяти. Буфер кадра сам по себе не является устройством вывода, он лишь используется для хранения рисунка. Наиболее часто в качестве устройства вывода, используемого с буфером кадра, выступает видеомонитор.

Чтобы понять принципы работы растровых дисплеев, мы рассмотрим в общих чертах устройство цветной растровой электронно-лучевой трубки. Изображение на экране получается с помощью сфокусированного электронного луча, который, попадая на экран, покрытый люминофором, дает яркое цветовое пятно. Луч в растровом дисплее может отклоняться только в строго определенные позиции на экране, образующие своеобразную мозаику. Люминофорное покрытие тоже не непрерывно, а представляет собой множество близко расположенных мельчайших точек, куда может позиционироваться луч. Дисплей, формирующий черно-белые изображения, имеет одну

электронную пушку, и ее луч высвечивает однотонные цветовые пятна. В цветной ЭЛТ находятся три электронных пушки, по одной на каждый основной цвет: красный, зеленый и синий. Электронные пушки часто объединены в треугольный блок, соответствующий треугольным блокам красного, зеленого и синего люминофоров на экране. Электронные лучи от каждой из пушек, проходя через специальную теневую маску, попадают точно на пятно своего люминофора. Изменение интенсивности каждого из трех лучей позволяет получить не только три основных цвета, но и цвета, получаемые при их смешении в разных пропорциях, что дает очень большое количество цветов для каждого пикселя экрана.

Дисплеи на **жидкокристаллических индикаторах** работают аналогично индикаторам в электронных часах, но, конечно, изображение состоит не из нескольких крупных сегментов, а из большого числа отдельно управляемых точек. Эти дисплеи имеют наименьшие габариты и энергопотребление, поэтому широко используются в портативных компьютерах. Они имеют как преимущества, так и недостатки по сравнению с дисплеями на ЭЛТ. Хотя исторически такой способ вывода изображения появился раньше, чем растровый дисплей с ЭЛТ, но быстро развиваться он начал значительно позднее. Эти дисплеи также являются растровыми устройствами (их тоже можно представить как матрицу элементов - жидких кристаллов).

Существуют и другие виды дисплеев, например плазменная панель, но мы не будем их касаться, поскольку они также являются растровыми, а техническая реализация не является предметом нашего курса. Важно то, что рассматриваемые нами алгоритмы разработаны для растровых графических дисплеев, а общие принципы работы этих устройств нам понятны.

Помимо дисплеев, в качестве устройств вывода изображений используются плоттеры (графопостроители), предназначенные для вывода графической информации на бумагу. Ранние графические пакеты были ориентированы именно на модель перьевого плоттера, формирующего изображение с помощью пера. Перо может перемещаться вдоль двух направляющих, соответствующих двум координатным осям, причем оно может находиться в двух состояниях - поднятом и опущенном. В поднятом состоянии оно просто перемещается над поверхностью бумаги, а в опущенном оставляет на бумаге линии, формирующие изображение. Таким образом, плоттер стоит ближе к векторным дисплеям, но отличается от них тем, что стирать выводимые изображения невозможно. Поэтому для них изображение сначала полностью формируется в памяти компьютера, а затем выводится.

Кроме того, следует упомянуть принтеры, выводящие изображение на бумагу или пленку. Изображение, получаемое с помощью современных принтеров, также формируется как точечное (растровое), но, как правило, с лучшим разрешением, чем экранное. Как и в случае с графопостроителем, стереть изображение или его часть невозможно.

Теперь сделаем небольшой обзор устройств ввода информации, позволяющих решать различные задачи компьютерной графики, не вдаваясь в детали физических принципов их работы. Эти устройства позволяют организовать диалог "человек-компьютер", а особенности конструкции каждого устройства позволяют ему специализироваться на выполнении определенного круга задач. Нас они интересуют именно как логические устройства, т.е. с точки зрения выполняемых ими функций.

Первую группу устройств, с помощью которых пользователь может указать позицию на экране, назовем **устройствами указания** (**pointing device**): мышь, трекбол (trackball), световое перо (lightpen), джойстик (joystick), спейсбол (spaceball). Практически все устройства этой группы оснащены парой или несколькими кнопками, которые позволяют сформировать и передать в компьютер какие-либо сигналы или прерывания.

Рис. 1.1. Мышь

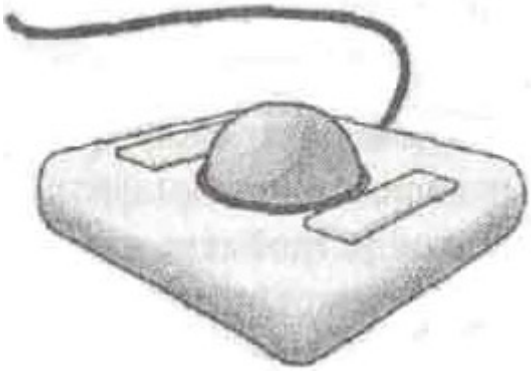


Рис. 1.2. Трекбол

Мышь (рис. 1.1) и трекбол (рис. 1.2) похожи не только по назначению, но часто и по конструкции. В механической мыши и трекболе вращение шарика преобразуется с помощью пары преобразователей в сигналы, передаваемые в компьютер. Преобразователи измеряют вращение относительно двух взаимно перпендикулярных осей. Существует очень много модификаций устройств этих групп. В оптической мыши используются не механические, а оптические чувствительные элементы для измерения перемещения: измеряется расстояние путем подсчета штрихов на специальной подложке. Маленькие трекболы широко применяются в портативных компьютерах, где их встраивают прямо в клавиатуру.

В некоторые клавиатуры встраиваются приборы, чувствительные к давлению, которые выполняют те же функции, что и мышь или трекбол, но при этом в них отсутствуют подвижные элементы. Преобразователи в таких устройствах измеряют величину давления на небольшой выпуклый набалдашник, размещенный между двумя кнопками в средней части клавиатуры. Они, как и трекбол, используются преимущественно в портативных компьютерах.

Выходные сигналы мыши или трекбола можно рассматривать как две независимые величины и преобразовывать их в координаты положения на двумерной плоскости экрана или в какой-либо другой системе координат. Считанные с устройства значения можно сразу же использовать для управления специальной отметкой (курсором) на экране.

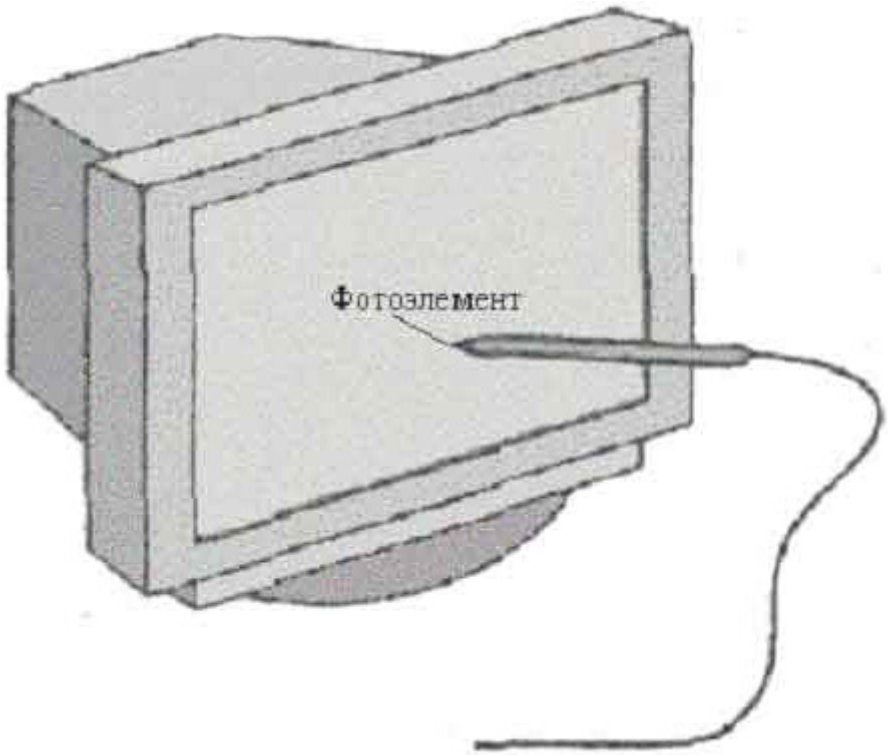


Рис. 1.3. Световое перо

Ветераном среди устройств ввода в компьютерной графике является устройство, названное при его создании световым пером. Впервые оно появилось в уже упомянутом проекте А.Сазерленда Sketchpad. Световое перо содержит фоточувствительный элемент (рис. 1.3), который при приближении к экрану воспринимает излучение, порождаемое при столкновении электронов с люминофорным покрытием экрана. Если мощность светового импульса превышает определенный порог, фоточувствительный элемент формирует импульс, который передается в компьютер. Анализируя смещение по времени этого импульса относительно начала цикла регенерации, компьютер может точно определить координаты той точки экрана, возбуждение которой "высветило" фотоэлемент. Таким образом, в распоряжении пользователя оказывается устройство непосредственного указания, работающее напрямую с изображением на экране. В настоящее время это устройство уже практически вышло из употребления: оно вытеснено более простым и надежным - мышью.

Еще одно устройство, которое достаточно активно используется в мультимедийных приложениях, а также в различного рода компьютерных тренажерах - джойстик (рис. 1.4). Перемещение джойстика в двух взаимно перпендикулярных направлениях воспринимается преобразователями, интерпретируется как вектор скорости, а полученные значения используются для управления положением маркера на экране. Обработка сигнала выполняется таким образом, что неподвижный джойстик в каком-либо промежуточном положении не изменяет положения маркера, а чем дальше джойстик отклонен от начального положения, тем быстрее маркер перемещается по экрану. Таким образом, джойстик играет роль устройства ввода с переменной чувствительностью. Другое достоинство джойстика - наличие силовой обратной связи, обеспеченной наличием разного рода пружин. При этом пользователь чувствует, что чем дальше отклонен джойстик, тем большее усилие требуется для его дальнейшего движения. Это как раз те свойства, которые нужны при работе с разного рода симуляторами, а также в компьютерных играх.

Рис. 1.4. Джойстик

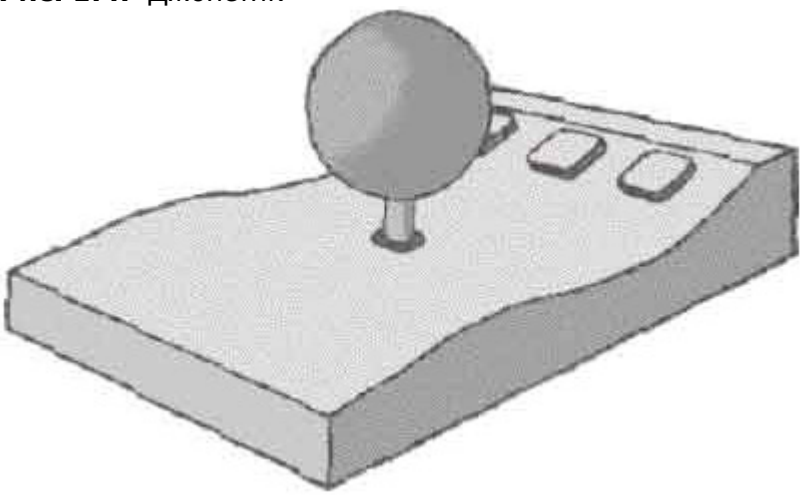


Рис. 1.5. Спейсбол

Спейсбол - это "трехмерное" устройство ввода. Хотя и существуют различные конструкции таких устройств, они все еще не получили широкого распространения, поскольку проигрывают популярным двумерным устройствам как по стоимости, так и по техническим характеристикам. Спейсбол похож на джойстик, но отличается от него тем, что он имеет вид закрепленного на рукоятке шара, причем рукоятка в этой конструкции неподвижна (рис. 1.5). Шар имеет датчики давления, которые измеряют усилие, прикладываемое пользователем. Шар может измерять не только составляющие усилия в трех основных направлениях (сверху вниз, от себя или на себя, влево-вправо), но и вращение относительно трех осей. Таким образом, это устройство способно передавать в компьютер шесть независимых параметров (т. е. имеет шесть степеней свободы), характеризующих как поступательное движение, так и вращение.

Существуют и другие трехмерные системы измерения и ввода, использующие самые современные технологии, например лазерные. В системах виртуальной реальности используются более сложные устройства, позволяющие динамически отслеживать положение и ориентацию пользователя. Для приложений, связанных с современной робототехникой и моделированием виртуальной реальности, иногда требуются устройства, обладающие еще большим числом степеней свободы, чем спейсбол. В последнее время появились новые разработки в этом направлении, в частности - перчатки с системой датчиков, которые способны улавливать движения отдельных частей руки человека (рис. 1.6).

Рис. 1.6. Перчатка для ввода данных

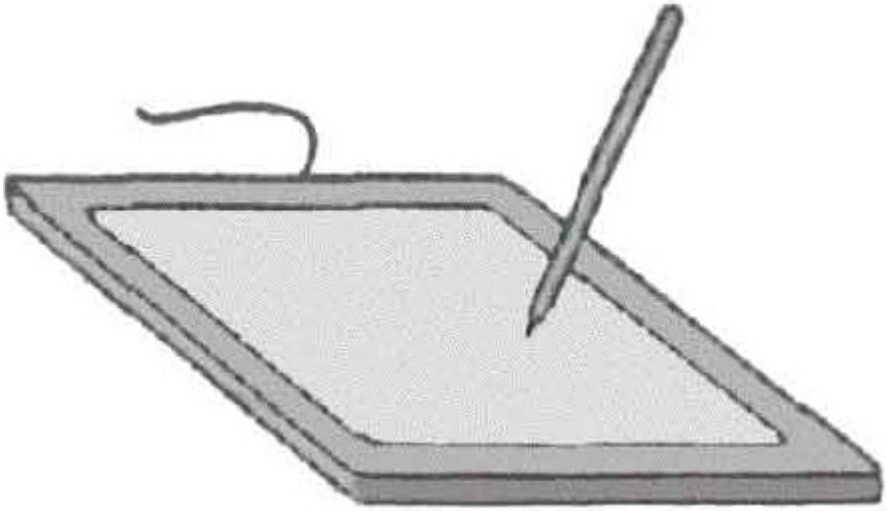


Рис. 1.7. Планшет

При использовании мыши или трекбола анализируется относительное положение устройства. Если переместить указатель на экране каким-либо способом в другое место, не вращая при этом шарик мыши или трекбола, то дальнейшие сигналы будут смещать указатель относительно новой позиции. Можно также аккуратно переместить мышь без вращения шарика и это не приведет к перемещению курсора на экране. Абсолютные координаты устройства не считываются обрабатывающей программой. Но при вводе в компьютер графиков прикладной программе зачастую требуются абсолютные координаты устройства ввода. Такую возможность обеспечивают разного рода планшеты (рис. 1.7). В планшете применяется, как правило, ортогональная сетка проводов, расположенная под его поверхностью. Положение пера определяется через электромагнитное взаимодействие сигналов, проходящих от проводов к щупу. Иногда в

качестве планшета используются чувствительные к прикосновению прозрачные экраны, которые наносятся на поверхность ЭЛТ. Небольшие экраны такого типа размещаются иногда на клавиатуре портативных компьютеров. Чувствительные панели можно использовать в режимах как абсолютных, так и относительных координат.

Для растрового ввода изображений используются сканеры, позволяющие не только ввести образ в компьютер, но и произвести их обработку и документирование. Одна из важных областей применения сканеров - ввод текстов. При этом обработка введенного изображения выполняется программным обеспечением распознавания текстов, которое в настоящее время стало уже достаточно развитым. В САПР сканеры используются для автоматизации ввода ранее подготовленной конструкторской документации. В этом случае проблема заключается в том, что данные от сканера представлены в растровой, а не векторной форме, и требуется выполнение обратного преобразования "растр-вектор". Эта задача очень сложна: необходимо распознавать различные изображения и тексты, в том числе рукописные, учитывать, что линия может при сканировании не только получить различную ширину на разных участках, но и оказаться разорванной и т.д. Для решения этой задачи средств одной лишь компьютерной графики недостаточно: необходимо привлечение и других дисциплин.

Все вышеперечисленные устройства ввода с точки зрения передачи информации прикладным программам следует рассматривать как логические. Функционирование систем ввода характеризуется тем, какую информацию устройство передает в программу, когда и как оно передает эту информацию. Эти вопросы становятся особенно существенными при разработке пользовательского интерфейса.

Вопросы и упражнения

1. Назовите четыре основные области применения компьютерной графики.
2. Каковы основные направления развития компьютерной графики? Какие задачи они решают?
3. Где и когда впервые был использован дисплей в качестве устройства вывода ЭВМ?
4. Кем и когда была разработана первая интерактивная программа для рисования?
5. Назовите основных разработчиков методов закрашивания гладких поверхностей.
6. Кто является автором ряда алгоритмов построения растровых образов различных геометрических объектов?
7. Назовите авторов алгоритмов удаления невидимых линий.
8. В чем состоит основное различие между дисплеями с произвольным сканированием и растровым сканированием?
9. Чем отличается дисплей на запоминающей трубке от векторного дисплея с регенерацией изображения?
10. Каковы основные принципы работы цветной растровой электронно-лучевой трубки?
11. Как работает перьевой плоттер?
12. Назовите основные устройства ввода, использующиеся в компьютерной графике.
13. Какие из устройств ввода дают возможность работать в абсолютных координатах?
14. Перечислите области применения сканеров.

Лекция 2. Цвет в компьютерной графике

Цветовые модели: RGB,HSV, CMY и другие. Переход от одной модели к другой. Цветовой график МКО. Однородные цветовые пространства Luv, PHS

О природе света и цвета

Свет как физическое явление представляет собой поток электромагнитных волн различной длины и амплитуды. Глаз человека, будучи сложной оптической системой, воспринимает эти волны в диапазоне длин приблизительно от 350 до 780 нм. Свет воспринимается либо непосредственно от источника, например, от осветительных приборов, либо как отраженный от поверхностей объектов или преломленный при прохождении сквозь прозрачные и полупрозрачные объекты. Цвет - это характеристика восприятия глазом электромагнитных волн разной длины, поскольку именно длина волны определяет для глаза видимый цвет. Амплитуда, определяющая энергию волны (пропорциональную квадрату амплитуды), отвечает за яркость цвета. Таким образом, само понятие цвета является особенностью человеческого "видения" окружающей среды.

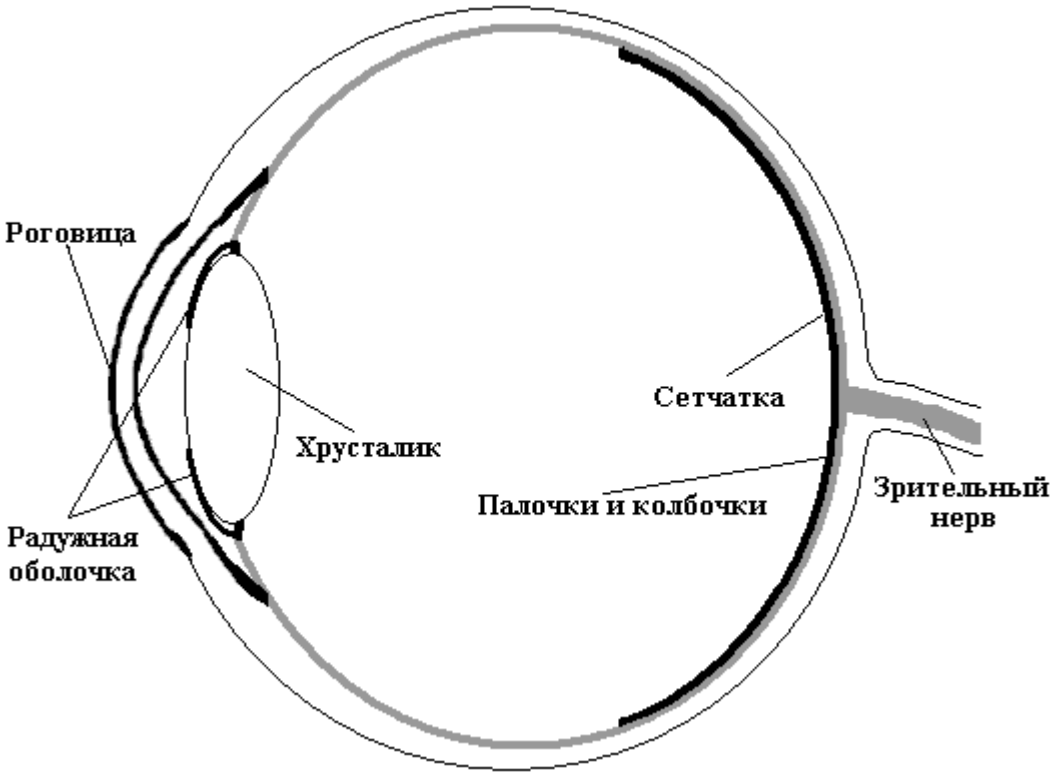


Рис. 2.1. Глаз человека

На рис. 2.1 схематически изображен глаз человека. Фоторецепторы, расположенные на поверхности сетчатки, играют роль приемников света. Хрусталик - это своеобразная линза, формирующая изображение, а радужная оболочка исполняет роль диафрагмы, регулируя количество света, пропускаемого внутрь глаза. Чувствительные клетки глаза неодинаково реагируют на волны различной длины. *Интенсивность* света есть мера энергии света, воздействующего на глаз, а *яркость* - это мера восприятия глазом этого воздействия. Интегральная кривая спектральной чувствительности глаза приведена на рис. 2.2; это **стандартная кривая Международной комиссии по освещению (МКО, или CIE - Comission International de l'Eclairage)**.

Фоторецепторы подразделяются на два вида: палочки и колбочки. Палочки являются высокочувствительными элементами и работают в условиях слабого освещения. Они нечувствительны к длине волны и поэтому не "различают" цвета. Колбочки же, наоборот, обладают узкой спектральной кривой и "различают" цвета. Палочек существует только один тип, а колбочки подразделяются на три вида, каждый из которых чувствителен к определенному диапазону длин волн (длинные, средние или короткие.) Чувствительность их также различна.

На рис. 2.3 представлены кривые чувствительности колбочек для всех трех видов. Видно, что наибольшей чувствительностью обладают колбочки, воспринимающие цвета зеленого спектра, немного слабее - "красные" колбочки и существенно слабее - "синие".

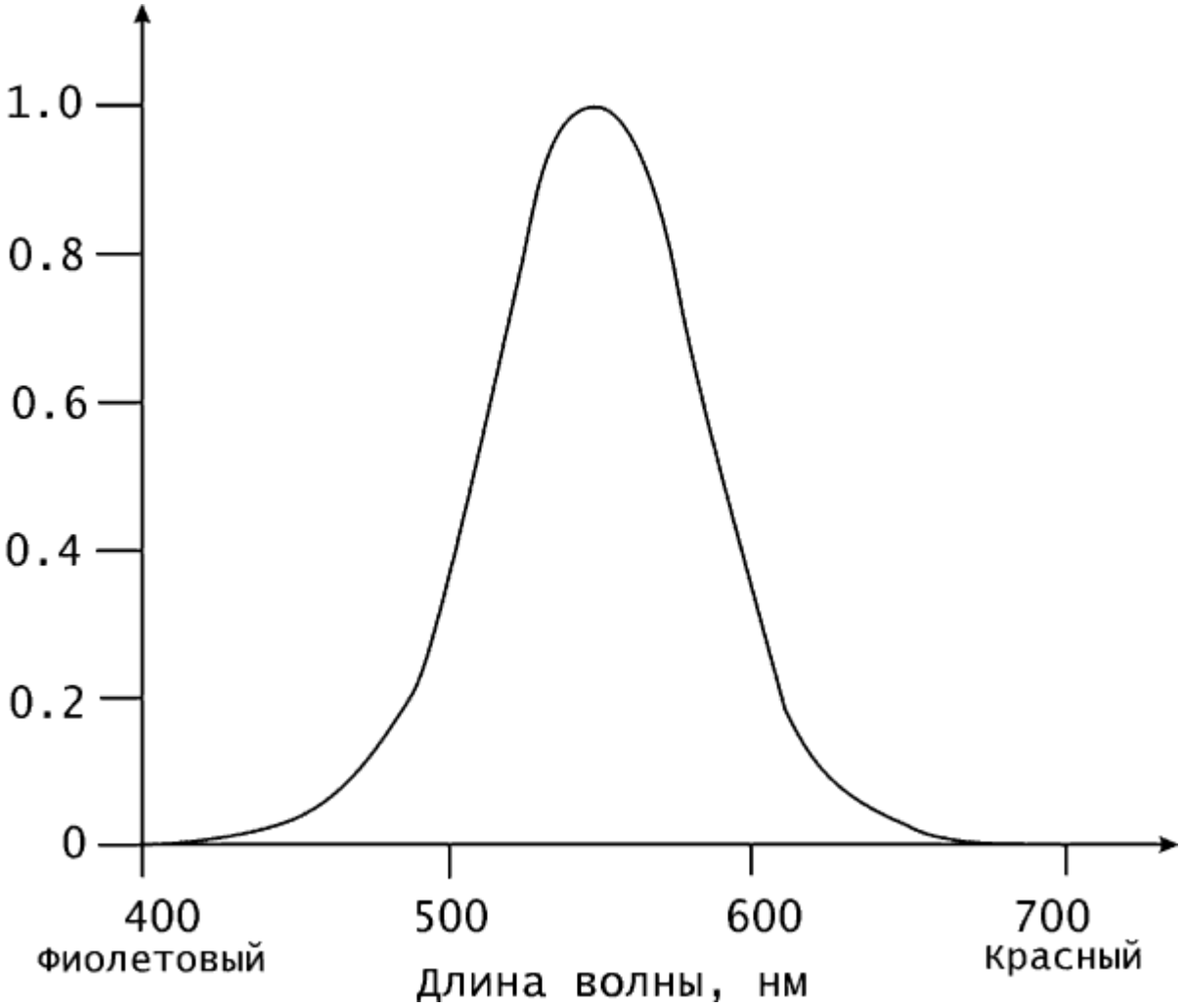


Рис. 2.2. Интегральная кривая спектральной чувствительности глаза

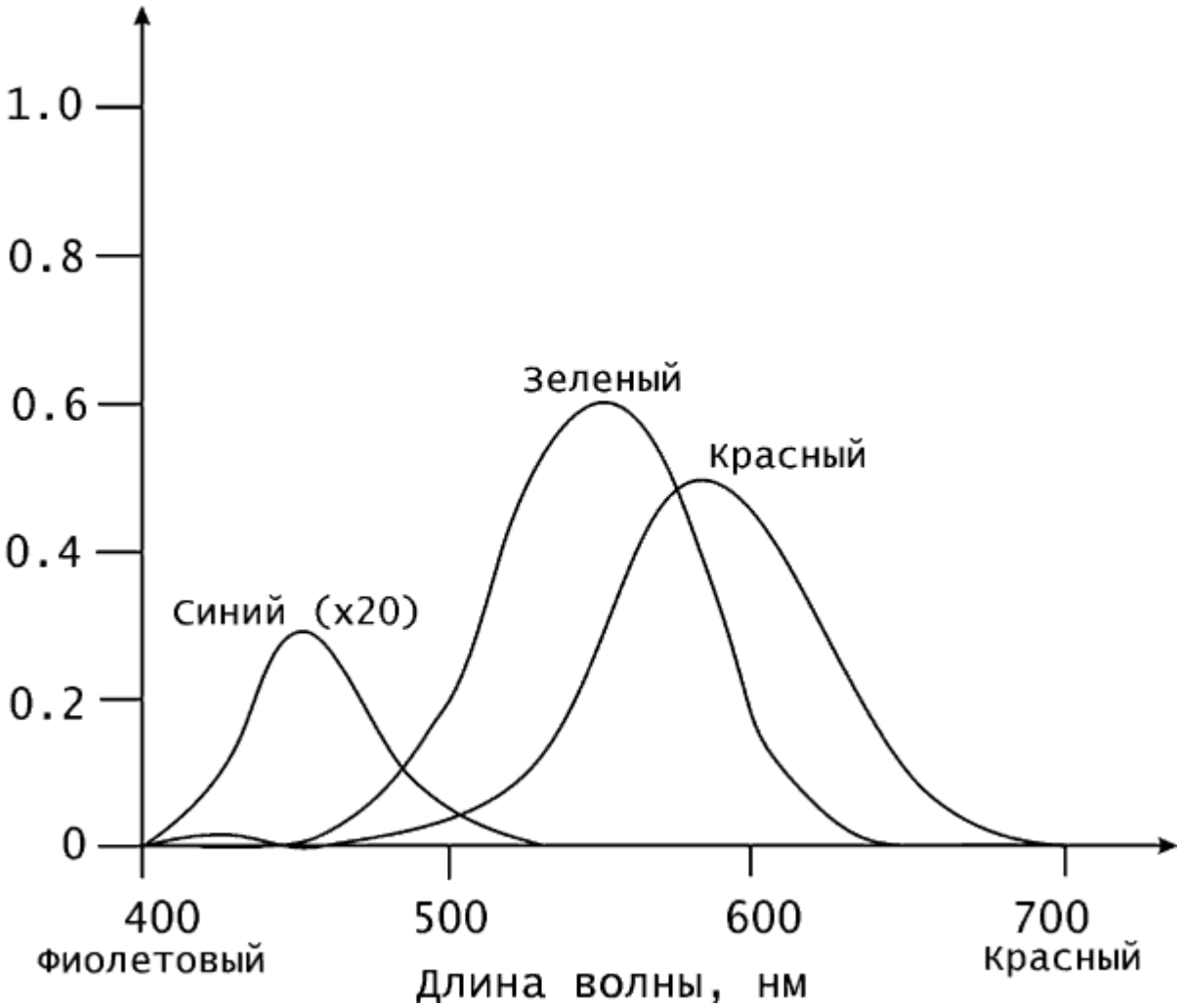


Рис. 2.3. Кривые чувствительности различных рецепторов

Таким образом, если функция $C(\lambda)$ характеризует спектральное разложение светового излучения от некоторого источника (рис. 2.4), т. е. распределение интенсивности по длинам волн, то три типа колбочек будут посылать в мозг сигналы R, G, B (красный, зеленый, синий), мощность которых определяется интегральными соотношениями

$$R = \int C(\lambda) S_R(\lambda) d\lambda,$$

$$G = \int C(\lambda) S_G(\lambda) d\lambda,$$

$$B = \int C(\lambda) S_B(\lambda) d\lambda,$$

где S_R, S_G, S_B - функции чувствительности соответствующих типов колбочек.

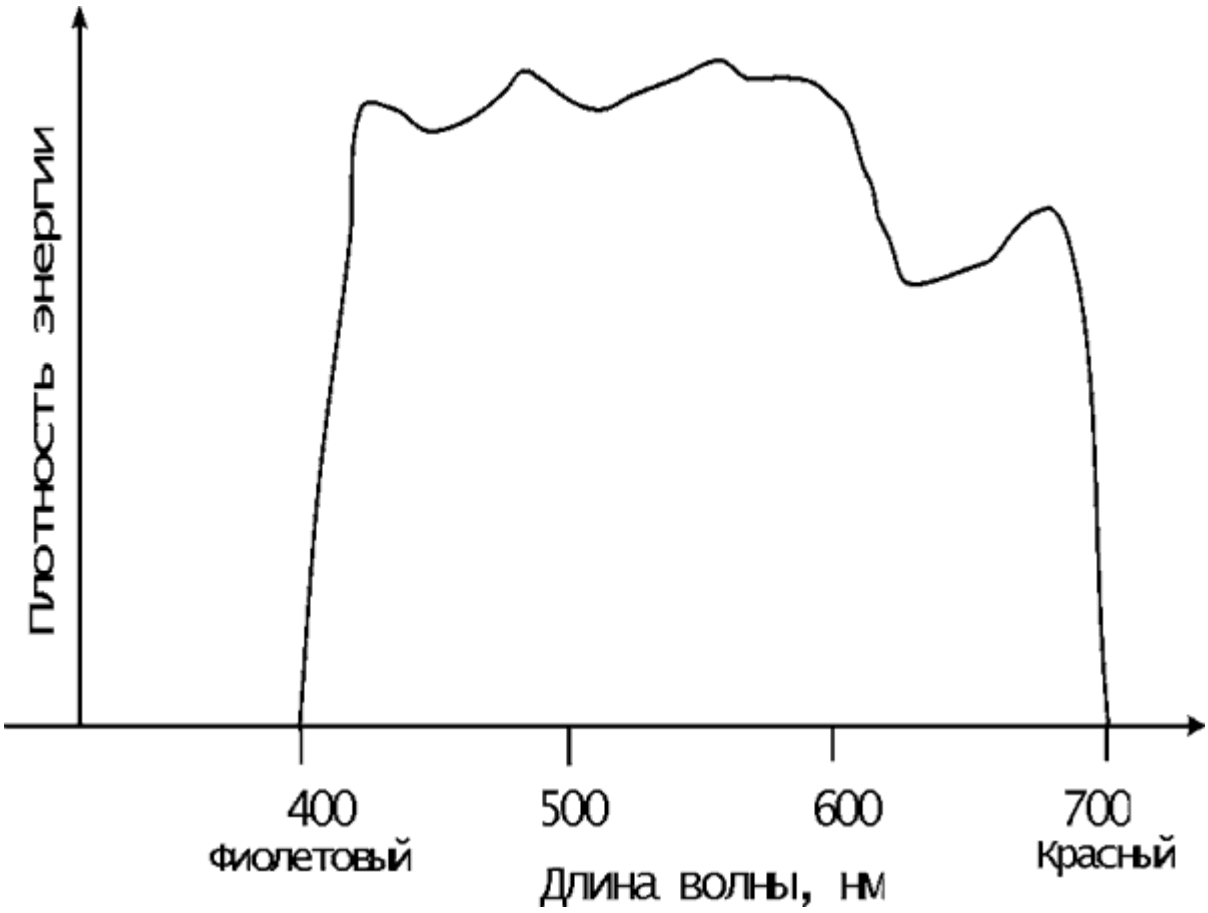


Рис. 2.4. Характерная спектральная кривая

Если воспринимаемый свет содержит все видимые длины волн в приблизительно равных количествах, то он называется **ахроматическим** и при максимальной интенсивности воспринимается как белый, а при более низких интенсивностях - как оттенки серого цвета. Интенсивность отраженного света удобно рассматривать в диапазоне от 0 до 1, и тогда нулевое значение будет соответствовать черному цвету. Если же свет содержит длины волн в неравных пропорциях, то он является **хроматическим**. Объект, отражающий свет, воспринимается как цветной, если он отражает или пропускает свет в узком диапазоне длин волн. Точно так же и источник света воспринимается как цветной, если он испускает волны в узком диапазоне длин. При освещении цветной поверхности цветным источником света могут получаться довольно разнообразные цветовые эффекты.

Цветовой график МКО

Трехмерная природа восприятия цвета позволяет отображать его в прямоугольной системе координат. Любой цвет можно изобразить в виде вектора, компонентами которого являются относительные веса красного, зеленого и синего цветов, вычисленные по формулам

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B}, \quad b = \frac{B}{R+G+B}.$$

Поскольку эти координаты в сумме всегда составляют единицу, а каждая из координат лежит в диапазоне от 0 до 1, то все представленные таким образом точки пространства будут лежать в одной плоскости, причем только в треугольнике, отсекаемом от нее положительным октантом системы координат (рис. 2.5а). Ясно, что при таком

представлении все множество точек этого треугольника можно описать с помощью двух координат, так как третья выражается через них посредством соотношения

$$b = 1 - r - g.$$

Таким образом, мы переходим к двумерному представлению области, т.е. к проекции области на плоскость (рис. 2.5б).

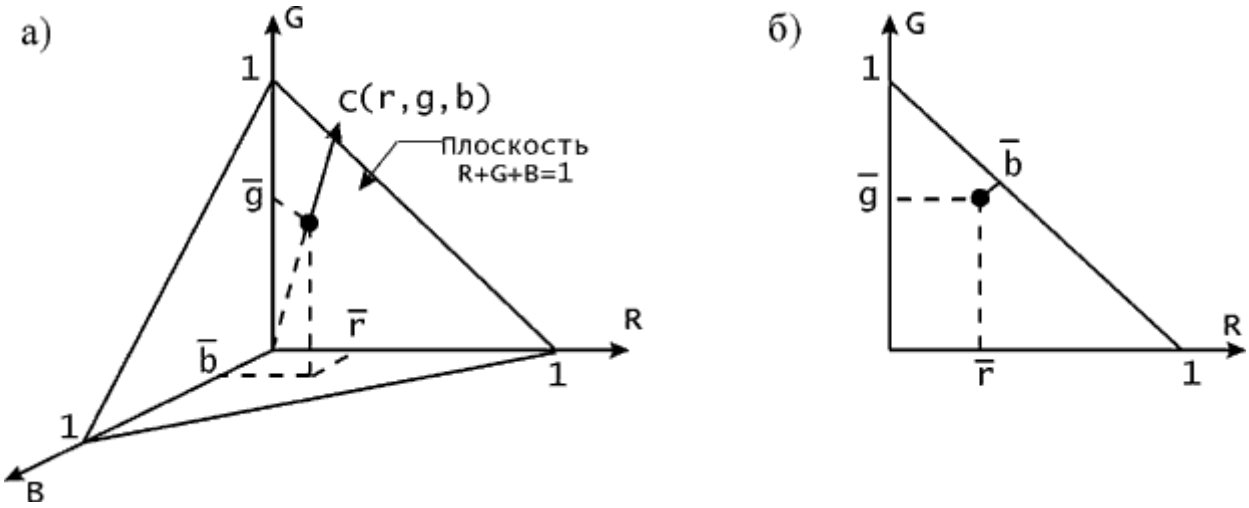


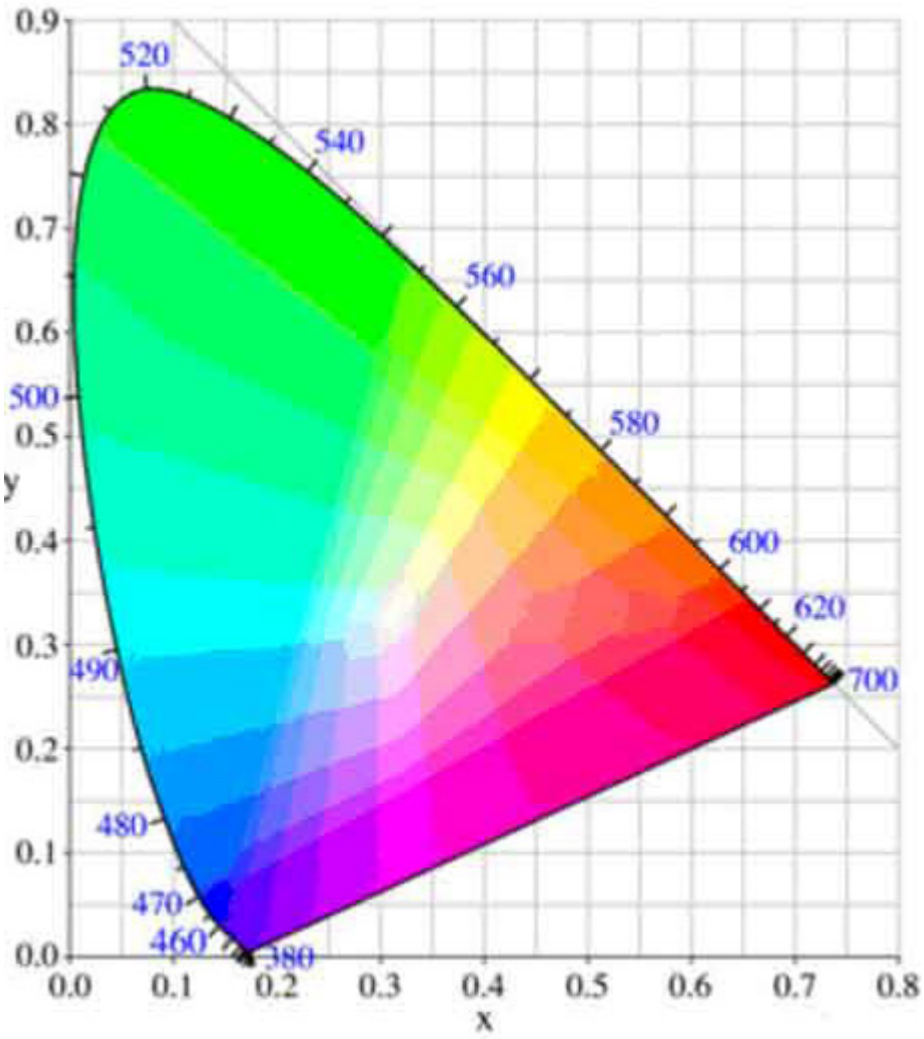
Рис. 2.5. Трехмерное цветовое пространство

С использованием такого преобразования в 1931 г. были выработаны международные стандарты определения и измерения цветов. Основой стандарта стал так называемый двумерный цветовой график МКО. Поскольку, как показали физические эксперименты, сложением трех основных цветов можно получить не все возможные цветовые оттенки, то в качестве базисных были выбраны другие параметры, полученные на основе исследования стандартных реакций глаза на свет. Эти параметры - X, Y, Z - являются чисто теоретическими, поскольку построены с использованием отрицательных значений основных составляющих цвета. Треугольник основных цветов был построен так, чтобы охватывать весь спектр видимого света. Кроме того, равное количество всех трех гипотетических цветов в сумме дает белый цвет. Координаты цветности строятся так же, как и в приведенной выше формуле:

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z}, \quad x+y+z = 1$$

При проекции этого треугольника на плоскость получается цветовой график МКО. Но координаты цветности определяют только относительные количества основных цветов, не задавая яркости результирующего цвета. Яркость можно задать координатой Y, а X, Z определить исходя из величин (x, y, Y), по формулам

$$X = \frac{Y}{y}x, \quad Z = \frac{Y}{y}(1-x-y).$$



увеличить изображение

Рис. 2.6. Цветовой график МКО. На контуре указаны длины волн в нанометрах

Цветовой график МКО приведен на рис. 2.6. Область, ограниченная кривой, охватывает весь видимый спектр, а сама кривая называется линией спектральных цветностей. Числа, проставленные на рисунке, означают длину волны в соответствующей точке. Точка C, соответствующая полуденному освещению при сплошной облачности, принята в качестве опорного белого цвета.

Цветовой график удобен для целого ряда задач. Например, с его помощью можно получить дополнительный цвет: для этого надо провести луч от данного цвета через опорную точку до пересечения с другой стороной кривой (цвета являются **дополнительными** друг к другу, если при сложении их в соответствующей пропорции получается белый цвет). Для определения доминирующей длины волны какого-либо цвета также проводится луч из опорной точки до пересечения с данным цветом и продолжается до пересечения с ближайшей точкой линии цветностей.

Для смешения двух цветов используются законы Грассмана. Пусть два цвета заданы на графике МКО координатами $D_1 = (x_1, y_1, Y_1)$ и $D_2 = (x_2, y_2, Y_2)$. Тогда смешение их дает цвет $D_{12} = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$. Если ввести обозначения $t_1 = \frac{Y_1}{y_1}, t_2 = \frac{Y_2}{y_2}$, то получим координаты цветности смеси

$$x_{12} = \frac{x_1 t_1 + x_2 t_2}{t_1 + t_2}, \quad y_{12} = \frac{y_1 t_1 + y_2 t_2}{t_1 + t_2}, \quad Y_{12} = Y_1 + Y_2.$$

Координаты МКО являются точным стандартом определения цвета. Но в различных областях, имеющих дело с цветом, есть свой подход к его моделированию. В частности, может использоваться другой набор основных цветов. Компьютерная графика опирается на систему RGB , поэтому представляет интерес переход между этими двумя наборами цветов (иными словами, преобразование координат цветности).

Цветовые модели RGB и CMY

Цветовые модели, используемые в компьютерной графике, - это средства описания цветов в определенном диапазоне.

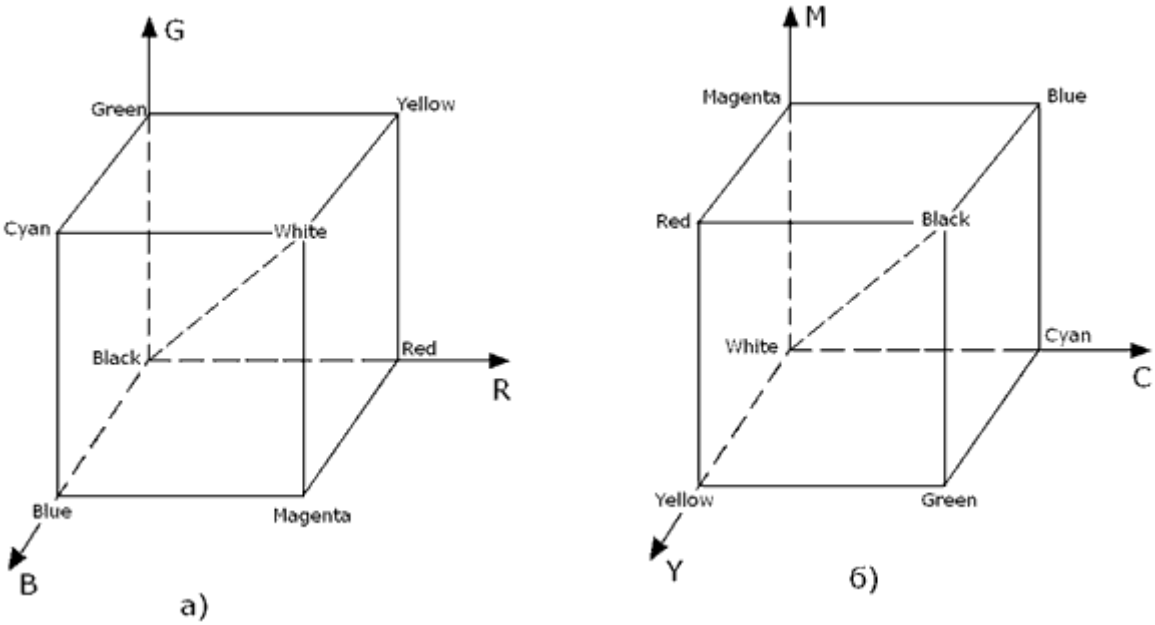


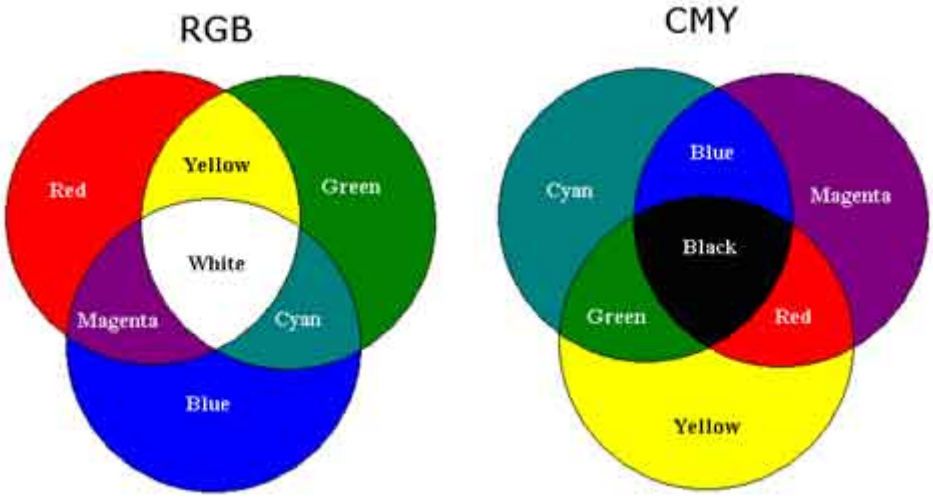
Рис. 2.7. Цветовой куб для моделей RGB и CMY

На основе описанных выше физических представлений в компьютерной графике была принята так называемая **аддитивная цветовая модель**, использующая **три первичных составляющих цвета**. Эта модель предполагает, что любой цвет можно рассматривать как взвешенную сумму трех основных цветов. Проиллюстрировать ее можно на примере освещения сцены с помощью трех прожекторов разного цвета. Каждый прожектор управляется независимо, и путем изменения мощности каждого из них можно воспроизвести практически все цвета. В модели RGB цвет можно представить в виде вектора в трехмерной системе координат с началом отсчета в точке (0,0,0). Максимальное значение каждой из компонент вектора примем за 1. Тогда вектор (1,1,1) соответствует белому цвету. Все цветовые векторы, таким образом, заключены внутри единичного куба, называемого цветовым кубом (рис. 2.7а).

Другая модель смешения цветов - **субстрактивная цветовая модель**, или модель CMY, использующая в качестве первичных составляющих цвета Cyan, Magenta, Yellow (голубой, пурпурный, желтый), которые являются дополнительными к Red, Green, Blue. В этой модели оттенки цвета получаются путем "вычитания" из падающего света волн определенной длины. Этот подход нуждается в пояснении. В этой системе координат вектор (0,0,0) соответствует белому цвету, а вектор (1,1,1) - черному. Соответствующий цветовой куб представлен на рис. 2.7б.

Связь между значениями (R,G,B) и (C,M,Y) для одного и того же цвета выражается формулой

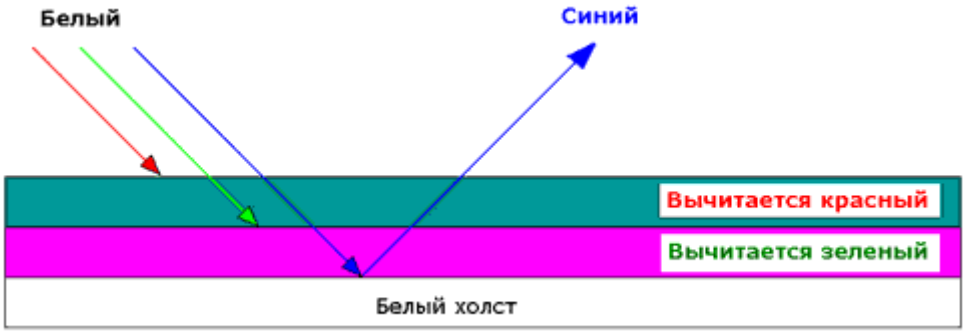
$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$



увеличить изображение

Рис. 2.8. Схема смешения цветов для моделей RGB и CMY

Цвета одной модели являются дополнительными к цветам другой (дополнительный цвет - это цвет, результатом смешения которого с данным является белый). Схема смешения цветов для двух моделей представлена на рис. 2.8. Пример субстрактивного формирования оттенков показан на рис. 2.9. При освещении падающим белым светом в слое голубой (Cyan) краски из спектра белого цвета поглощается (вычитается) красная часть как дополнительный цвет, затем из оставшегося света в слое пурпурной (Magenta) краски поглощается зеленая часть спектра, и, наконец, от белой поверхности отражается синий цвет, который мы и видим. Таким образом, смешение голубого и пурпурного цветов дает в итоге синий цвет.



увеличить изображение

Рис. 2.9. Субстрактивное формирование оттенков

Растровые дисплеи, как правило, используют аппаратно- ориентированную модель цветов RGB. Существуют также дисплеи с **таблицей цветности**, представляющей собой матрицу, каждый элемент которой - некоторый цвет (вектор RGB). В таких дисплеях значения кодов пикселей, заносимые в видеопамять, представляют собой индексы матрицы цветности. При отображении некоторого пикселя на экран по

значению кода выбирается элемент таблицы цветности, содержащий тройку значений R, G, B. Эта тройка и передается на монитор для задания цвета пикселя на экране.

В полноцветных дисплеях для каждого пикселя в видеопамять заносится тройка значений R, G, B. В этом случае для отображения пикселя из видеопамяти непосредственно выбираются значения R, G, B, которые и передаются на монитор (но могут и передаваться в таблицу цветности).

В моделях RGB и CMY легко задавать яркости для одного из основных цветов, но довольно затруднительно задать оттенок с требуемым цветовым тоном и насыщенностью, соответствующим какому-либо образцу цвета. В различного рода графических редакторах эта задача чаще всего решается с помощью интерактивного выбора из палитры цветов и формированием цветов в палитре путем подбора значений координат до получения требуемого визуального результата. Иногда такая палитра наглядно отображает выбор вектора из цветового куба: сначала посредством одного движка выбирается цветовая плоскость, а затем на этой плоскости выбирается конкретная точка. Но и таким методом не сразу удается достигнуть желаемого эффекта, поскольку не так просто выбрать правильную цветовую плоскость.

Цветовые модели HSV и HLS

Приведенные модели не охватывают всего диапазона видимого цвета, поскольку их цветовой охват - это лишь треугольник на графике МКО, вершинам которого соответствуют базовые цвета. Они являются аппаратно ориентированными, т.е. соответствуют технической реализации цвета в устройствах графического вывода. Но психофизиологическое восприятие света определяется не интенсивностью трех первичных цветов, а цветовым тоном, насыщенностью и светлотой. Цветовой тон позволяет различать цвета, насыщенность задает степень "разбавления" чистого тона белым цветом, а светлота - это интенсивность света в целом. Поэтому для адекватного нашему восприятию подбора оттенков более удобными являются модели, в числе параметров которых присутствует тон (Hue). Этот параметр принято измерять углом, отсчитываемым вокруг вертикальной оси. При этом красному цвету соответствует угол 0 $\deg$, зеленому - 120 $\deg$, синему - 240 $\deg$, а дополняющие друг друга цвета расположены один напротив другого, т.е. угол между ними составляет 180 $\deg$. Цвета CMY расположены посредине между составляющими их компонентами RGB. Существует две модели, использующие этот параметр.

Модель HSV (Hue, Saturation, Value, или тон, насыщенность, количество света) можно представить в виде световой шестигранной пирамиды (рис. 2.10), по оси которой откладывается значение V, а расстояние от оси до боковой грани в горизонтальном сечении соответствует параметру S (за диапазон изменения этих величин принимается интервал от нуля до единицы). Значение S равно единице, если точка лежит на боковой грани пирамиды. Шестиугольник, лежащий в основании пирамиды, представляет собой проекцию цветового куба в направлении его главной диагонали (рис. 2.11).

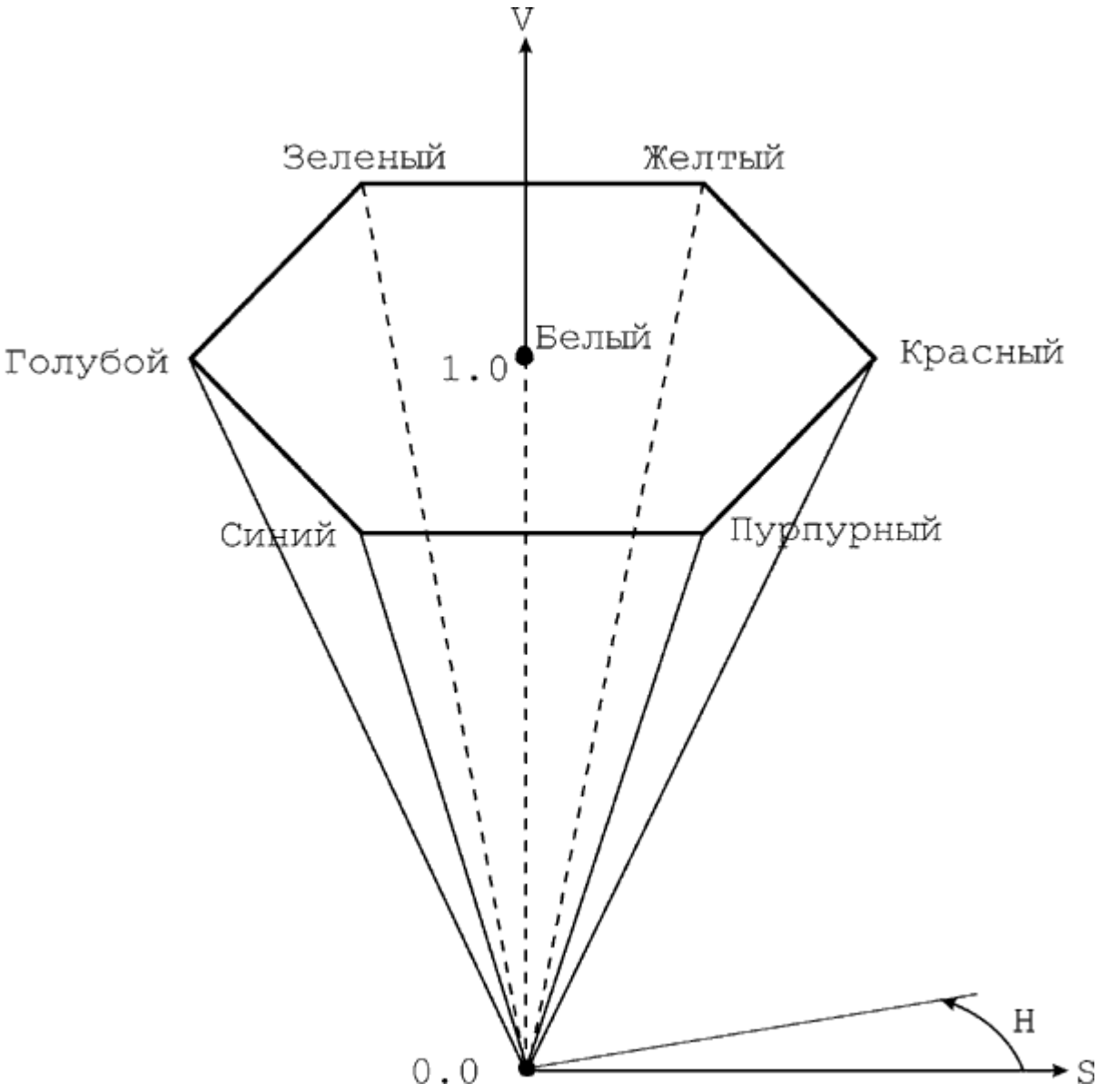


Рис. 2.10. Цветовое пространство HSV

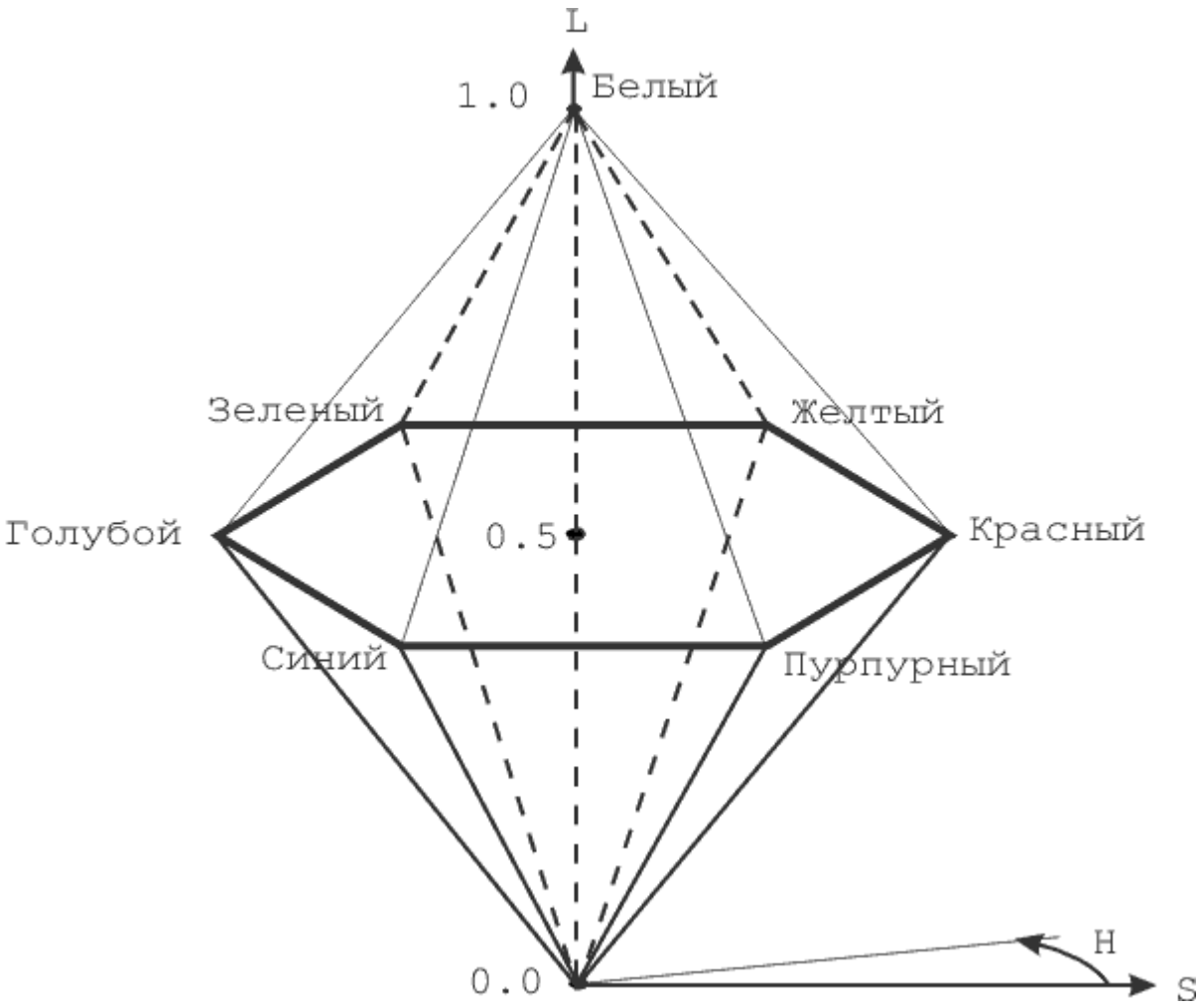


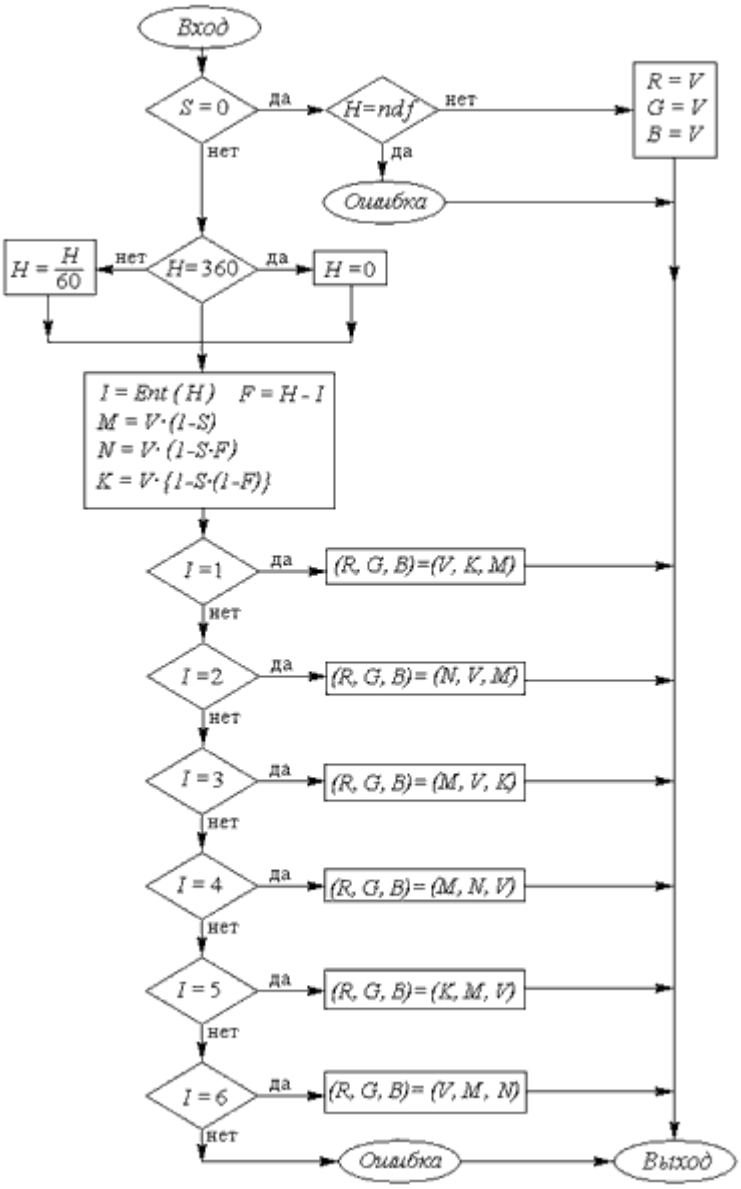
Рис. 2.11. Цветовое пространство HLS

Преобразование цветового пространства HSV в RGB осуществляется непосредственно с помощью геометрических соотношений между шестигранной пирамидой и кубом.

Цветовая модель HLS (Hue, Lightness, Saturation, или тон, светлота, насыщенность) является расширением модели HSV. Здесь цветовое пространство уже представляется в виде двойной пирамиды (рис. 2.11), в которой по вертикальной оси откладывается L (светлота), а остальные два параметра задаются так же, как и в предыдущей модели. В литературе эти пирамиды иногда называют шестигранным конусом.

На рис. 2.12 и 2.13 приведены блок-схемы преобразования моделей HSV и HLS в модель RGB. Алгоритмы обратного преобразования предлагаются читателю в качестве упражнения.

В первом алгоритме используется функция Ent, означающая целую часть числа. Кроме того, используется операция присваивания для векторов. Константа *ndf* (сокращенное от выражения "not defined") используется при входе в алгоритм для того, чтобы выяснить, задано ли значение переменной `H`. Например, по соглашению *ndf* может быть некоторым отрицательным значением, так как тон - это всегда положительная величина. Во втором алгоритме применяется вспомогательная функция *Value (H, M1, M2)* для вычисления значения компоненты R, G или B в зависимости от ситуации.



увеличить изображение

Рис. 2.12. Преобразование модели HSV в RGB

Алгоритм преобразования:

```
Приведение H к заданному диапазону:
   Пока H<0 H=H+360
   Пока H>360 H=H-360
Определение координат
   Если H<60 то Value=M1+(M2-M1)*H/60
   Если 60<=H<180 то Value=M2
   Если 180<=H<240 то Value=M1+(M2-M1)*(240-H)/60
   Если 240<=H то Value=M1
```

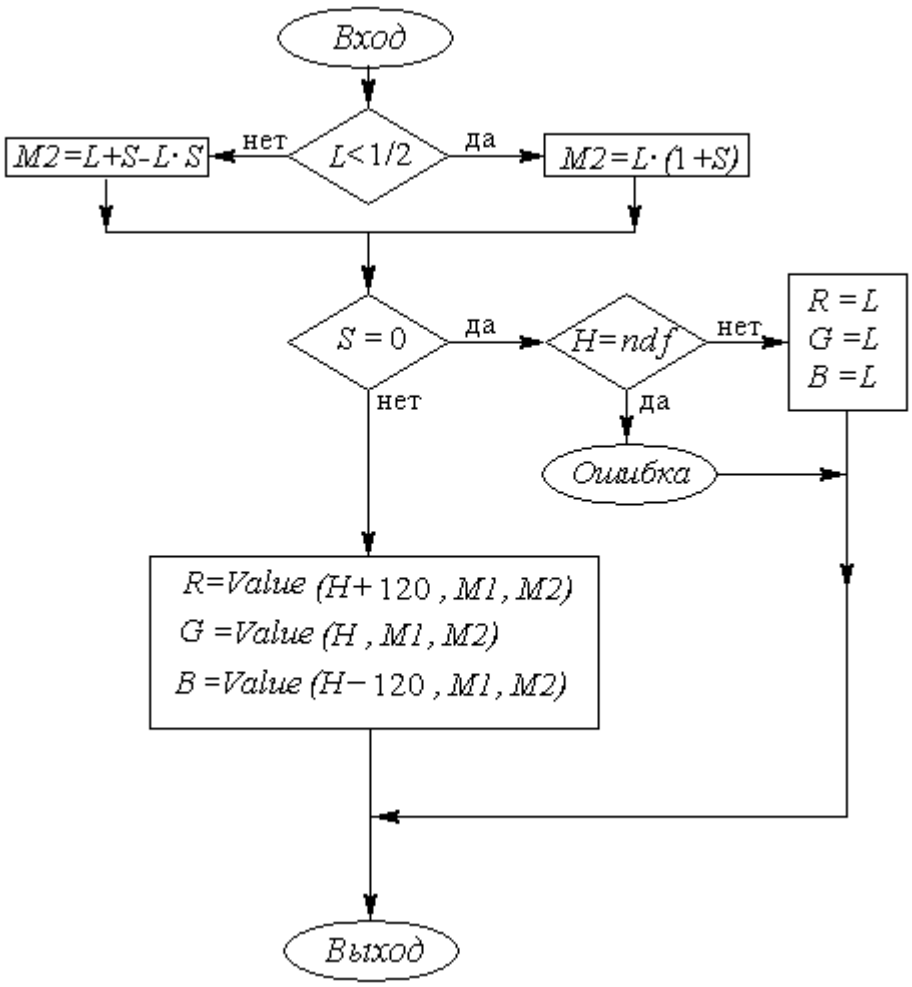


Рис. 2.13. Преобразование модели HLS в RGB

Пространство CIE Luv

Один из существенных минусов цветового пространства XYZ — это то, что оно не является перцептивно (визуально) равномерным и не может использоваться для вычисления цветовых расстояний. Поэтому CIE (МКО) продолжила разработку перцептивно равномерного пространства. Целью комитета CIE было создание повторяемой системы стандартов цветопередачи для производителей красок, чернил, пигментов и других красителей. Самая важная функция этих стандартов — предоставить универсальную схему, в рамках которой можно было бы устанавливать соответствие цветов.

В результате было создано цветовое пространство CIE Luv, позволяющее определить различение цветов для человека с "усредненным" зрением, (т.е. различные люди неодинаково воспринимают разницу между цветами). Свое название пространство получило благодаря его компонентам L, u и v. Параметр L соответствует яркости цвета, u отвечает за переход от зеленого к красному (при увеличении), а при увеличении параметра v происходит переход от синего к фиолетовому. Если u и v равны 0, то, меняя L, получаем цвета, являющиеся градациями серого.

Это цветовое пространство было разработано для количественного измерения различия двух цветов. CIE были проведены исследования с участием большого числа людей, результатом чего явилось создание пространства Luv. Измерения проводились в "хороших" условиях (достаточное освещение и неяркий монотонный фон); перед испытуемым находились два листа бумаги, окрашенных соответственно двумя цветами, и он должен был дать ответ, насколько, по его мнению, различаются эти цвета. В

случае реальных изображений мы должны находить различия между цветами на более сложном фоне, при этом не всегда при хорошем освещении (например, слишком ярком). Но освещение зависит и от помещения, и от времени суток, и от того, под каким углом находится поверхность к источнику света.

Переход из RGB в Luv осуществляется следующим образом. Сначала нормируем R, G, B:

$$\begin{pmatrix} R^* \\ G^* \\ B^* \end{pmatrix} = \begin{pmatrix} R/255 \\ G/255 \\ B/255 \end{pmatrix}$$

Далее совершаем преобразование пространства RGB в XYZ:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.35758 & 0.180423 \\ 0.212671 & 0.71516 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \times \begin{pmatrix} R^* \\ G^* \\ B^* \end{pmatrix}$$

Цветовое пространство CIE Luv — непрерывное однородное преобразование пространства CIE XYZ, описываемое следующими формулами:

$$L = \begin{cases} 116 \times \sqrt[3]{\frac{Y}{Y_n}} - 16, & \frac{Y}{Y_n} > 0.008856 \\ 903.3 \times \frac{Y}{Y_n}, & \frac{Y}{Y_n} \le 0.008856 \end{cases}$$

$$u' = \frac{4X}{X + 15Y + 3Z} = \frac{4x}{-2x + 12y + 3}, \quad v' = \frac{9Y}{X + 15Y + 3Z} = \frac{9y}{-2x + 12y + 3}$$

$$u = 13L(u' - u_n), \quad v = 13L(v' - v_n)$$

Для определения параметров Y_n, u_n и v_n, вводится понятие **белой точки** (**white point**). Белая точка - это пара параметров цветности (x, y), определяющая эталон белого цвета для различных источников света. CIE составила таблицу белых точек для источников света разной яркости. При этом значение компоненты Y белой точки в XYZ нормализовано до 100 (в приведенных выше формулах Y_n как раз соответствует нормализованной Y компоненте). Параметры u_n и v_n вычисляются по тем же формулам, что u' и v', в которых используются значения x и y для белой точки.

Как уже упоминалось выше, компонента L соответствует яркости цвета, а из формул видно, что L пропорциональна кубическому корню из компоненты Y пространства XYZ. Однако существует мнение, что человеческому восприятию больше соответствует корень второй степени из освещенности. Так, например, в цветовом пространстве Lab параметр L вычисляется с использованием квадратного корня.

Немного о свойствах величин L, u, v:

- L меняется от 0 до 100;
- u, v лежат в пределах -200, 200;
- u отвечает за переход от зеленого к красному (при увеличении u);
- v отвечает за переход от синего к фиолетовому (при увеличении v);
- если u и v равны 0, меняя L, получаем изображение, содержащее градации серого (grayscale).

Наконец, самое важное, к чему мы стремились, переходя в это пространство. Нам заданы два цвета - L_1, u_1, v_1 и L_2, u_2, v_2. Как определить расстояние между цветами, то есть насколько человек заметил бы различие между ними? Оказывается, оно задается евклидовой нормой

$$D = \sqrt{(L_1 - L_2)^2 + (u_1 - u_2)^2 + (v_1 - v_2)^2}$$

При расстоянии между двумя цветами $D > 5$ большинство людей уже замечают различие, при $D > 10$ оно заметно всем. В этом и состоит главное достоинство этого пространства. Оно учитывает восприятие цветов человеком, и различие между цветами определяется очень простой формулой. Необходимо заметить, что эта формула применима в определенных условиях: освещение, фон не должны мешать и отвлекать.

Одновременно с разработкой CIE Luv было также разработано перцептивно равномерное цветовое пространство CIE Lab. Из этих двух моделей более широко применяется модель CIE Lab. Структура цветового пространства Lab основана на той теории, что цвет не может быть одновременно зеленым и красным или желтым и синим (рис. 2.14). Следовательно, для описания атрибутов "красный/зеленый" и "желтый/синий" можно воспользоваться одними и теми же значениями. Формулы перехода от пространства XYZ к пространству Lab осуществляется следующим образом:

$$L = \begin{cases} 116 \cdot [(Y/Y_n)^{1/3}] - 16 & \text{если } (Y/Y_n) > 0.008856 \\ 903.3 \cdot Y/Y_n & \text{если } (Y/Y_n) \le 0.008856 \end{cases} \quad a = 500 \cdot [f(X/X_n) - f(Y/Y_n$$

$$b = 200 \cdot [f(Y/Y_n) - f(Z/Z_n$$

$$\text{где } f(t) = \begin{cases} t^{1/3} & \text{если } (Y/Y_n) > 0.008856 \\ 7.787 \cdot t + 16/116 & \text{если } (Y/Y_n) \le 0.008856 \end{cases}$$

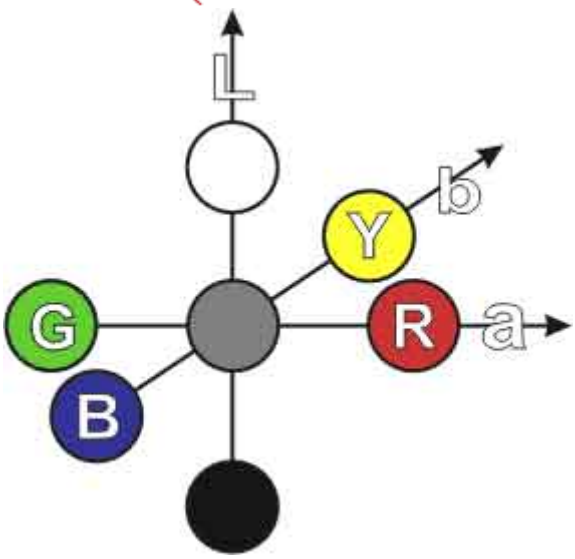


Рис. 2.14. Представление цвета в пространстве CIE Lab

Рис. 2.15. Видимое стандартным наблюдателем пространство Lab

Каждая цветовая модель, помимо преимуществ, также имеет и свои индивидуальные недостатки. Существуют и другие модели, которые здесь не рассматриваются.

Вопросы и упражнения

1. Расположите в убывающем порядке чувствительность рецепторов глаза к цветам: красный, зеленый, синий.
2. Что такое хроматический спектр?
3. Что такое ахроматический спектр?
4. Как осуществляется проекция трехмерного цветового пространства на плоскость?
5. Чем отличается цветовой график МКО от треугольной проекционной области цветового пространства?
6. Что такое дополнительный цвет?
7. Что такое аддитивная и субстрактивная цветовые модели? Чем отличаются их цветовые кубы?
8. Что является основой цветовой модели HSV и HLS?
9. Являются ли цветовые модели HSV и HLS аддитивными или субстрактивными?
10. Постройте алгоритм преобразования модели RGB в HSV.
11. Постройте алгоритм преобразования модели RGB в HLS.
12. В чем состоит главное достоинство цветового пространства Luv?
13. В чем состоит главное достоинство цветового пространства Lab?

Лекция 3. Геометрические преобразования

Системы координат и геометрические преобразования (параллельный перенос, масштабирование, вращение). Задание геометрических преобразований с помощью матриц. Конгруэнтные преобразования. Переход в другую систему координат. Задача вращения относительно произвольной оси

Системы координат и векторы

Для дальнейшего изложения нам понадобятся некоторые сведения из аналитической геометрии и линейной алгебры. Не ставя перед собой задачу подробного рассмотрения всех этих вопросов, приведем (или напомним) те основные понятия и операции, которые используются в алгоритмах компьютерной графики.

Две взаимно перпендикулярные пересекающиеся прямые с заданным масштабом образуют **декартову прямоугольную систему координат на плоскости**. Точка пересечения *O* называется **началом координат**, прямые называются **осями координат**. Одну из осей называют *осью OX*, или **осью абсцисс**, другую - *осью OY*, или *осью ординат*. Эти оси также называют **координатными осями**.

Возьмем произвольную точку M на плоскости с заданной системой координат. Пусть M_x и M_y - проекции этой точки на оси абсцисс и ординат соответственно, причем длина отрезка OM_x равна x, а длина OM_y равна y. Тогда пара чисел (x, y) называется декартовыми координатами точки M на плоскости (**абсциссой** и **ординатой** точки).

Три взаимно перпендикулярные пересекающиеся прямые с заданным масштабом образуют **декартову прямоугольную систему координат в пространстве**. Так же как и в случае плоскости, точка пересечения *O* называется **началом координат**, прямые называются **осями координат**. Одну из осей называют *осью OX*, или **осью абсцисс**, другую - *осью OY*, или **осью ординат**, третью - *осью OZ*, или **осью аппликат**.

Пусть M_x, M_y и M_z - проекции произвольной точки M в пространстве на оси абсцисс, ординат и аппликат соответственно, причем длина отрезка OM_x равна x, длина OM_y равна y, а длина OM_z равна z. Тогда тройка чисел x, y, z называется декартовыми координатами точки M в пространстве (**абсциссой**, **ординатой** и **аппликатой** точки).

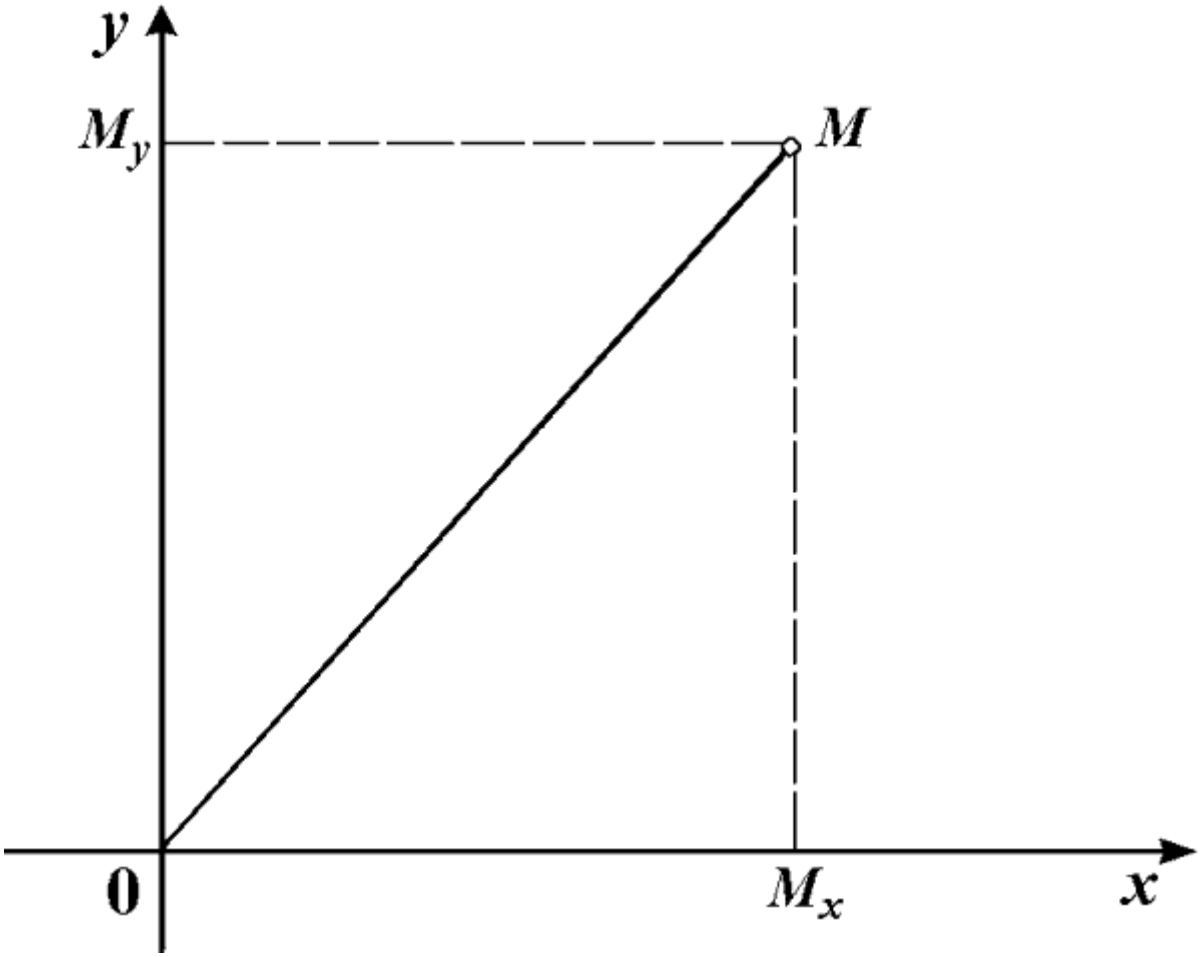


Рис. 3.1. Система координат на плоскости

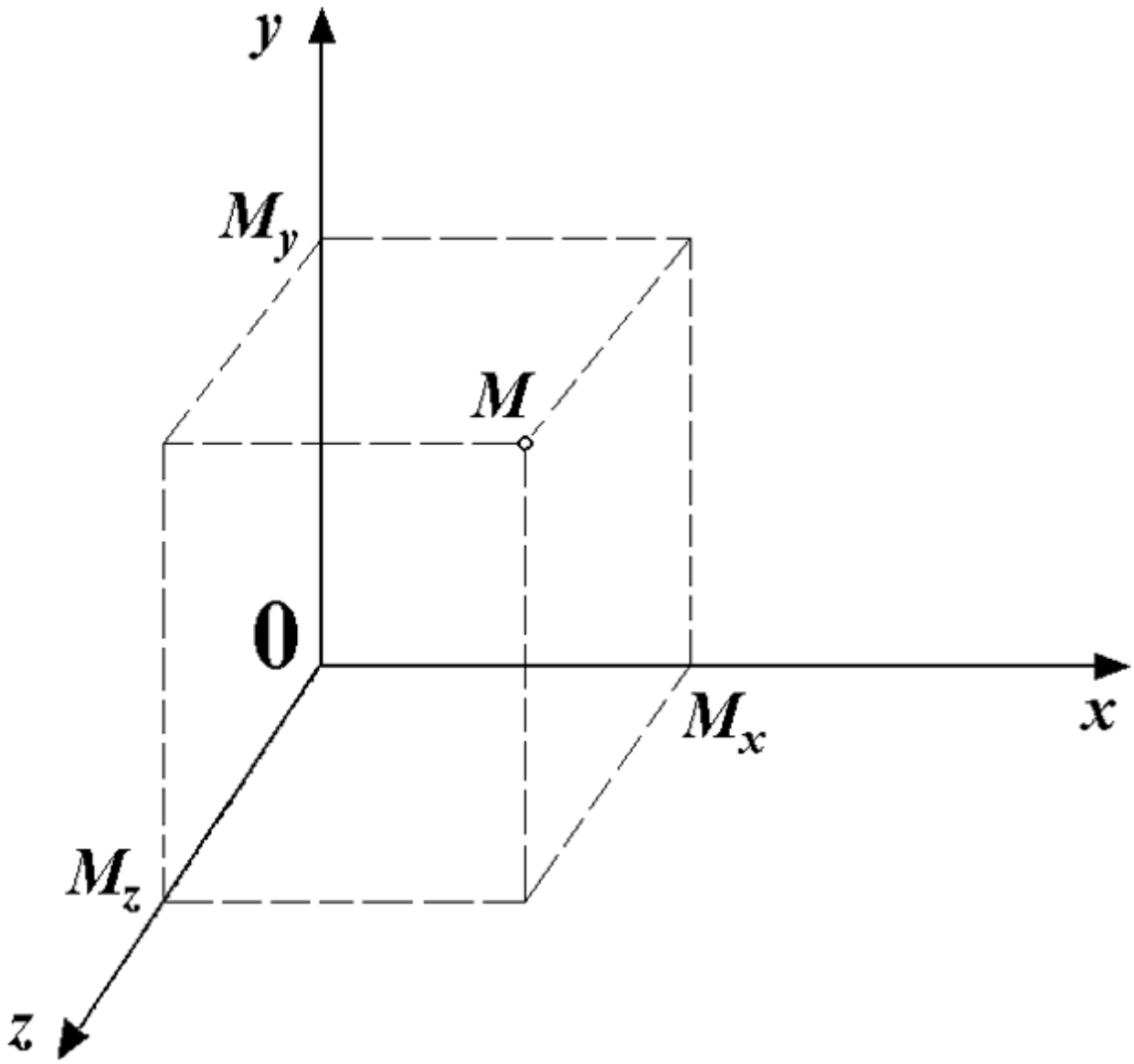


Рис. 3.2. Система координат в пространстве

Пусть на плоскости задана декартова система координат. Возьмем две точки с координатами (x_1, y_1) и (x_2, y_2) соответственно. Тогда, используя теорему Пифагора, можно получить, что расстояние между этими двумя точками выражается формулой

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

Расстояние между двумя точками в пространстве с координатами (x_1, y_1, z_1) и (x_2, y_2, z_2) выражается аналогичной формулой:

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

Отрезок на плоскости и в пространстве задается с помощью двух точек, указывающих его границы. **Геометрическим вектором**, или просто **вектором в пространстве**, будем называть отрезок, у которого указано, какая из его граничных точек является началом, а какая - концом (т.е. указано **направление** вектора). Начало вектора называют **точкой его приложения**. Вектор называется **нулевым**, если его начало и конец совпадают. Векторы называются **коллинеарными**, если они лежат на

параллельных прямых. Векторы считаются **равными**, если они коллинеарны, имеют одинаковую длину и одинаковое направление. Таким образом, все векторы, получающиеся параллельным переносом из одного и того же вектора, равны между собой. Любая точка на плоскости и в пространстве может рассматриваться как вектор, начало которого совпадает с началом координат (**радиус-вектор**), а каждый вектор, перенесенный в начало координат, задает своим концом единственную точку пространства. Поэтому любой вектор может быть представлен совокупностью своих координат в декартовой системе.

Линейными операциями над векторами принято называть операции сложения векторов и операцию умножения вектора на число.

Суммой двух векторов $\overrightarrow{a}$ и $\overrightarrow{b}$ называется вектор, идущий из начала вектора $\overrightarrow{a}$ в конец вектора $\overrightarrow{b}$, при условии, что вектор $\overrightarrow{b}$ приложен к концу вектора $\overrightarrow{a}$.

Перечислим основные свойства операции сложения векторов:

- $\overrightarrow{a} + \overrightarrow{b} = \overrightarrow{b} + \overrightarrow{a}$.
- $(\overrightarrow{a} + \overrightarrow{b}) + \overrightarrow{c} = \overrightarrow{a} + (\overrightarrow{b} + \overrightarrow{c})$.
- Существует нулевой вектор $\overrightarrow{0}$, такой, что $\overrightarrow{a} + \overrightarrow{0} = \overrightarrow{a}$ для любого вектора $\overrightarrow{a}$.
- Для каждого вектора $\overrightarrow{a}$ существует противоположный ему вектор $\overrightarrow{a}'$, такой, что $\overrightarrow{a} + \overrightarrow{a}' = \overrightarrow{0}$.

Разностью двух векторов $\overrightarrow{a}$ и $\overrightarrow{b}$ называется такой вектор $\overrightarrow{c}$, который в сумме с вектором $\overrightarrow{b}$ дает вектор $\overrightarrow{a}$.

Произведением $\alpha\overrightarrow{a}$ вектора $\overrightarrow{a}$ на число α называется вектор $\overrightarrow{b}$, коллинеарный вектору $\overrightarrow{a}$, имеющий длину $|\alpha| \cdot |\overrightarrow{a}|$ и направление, совпадающее с направлением вектора $\overrightarrow{a}$ при $\alpha > 0$ и противоположное направлению $\overrightarrow{a}$ при $\alpha < 0$.
Геометрический смысл умножения вектора на число состоит в том, что длина вектора увеличивается в $|\alpha|$ раз.

Операция умножения вектора на число обладает следующими свойствами:

- $\alpha(\overrightarrow{a} + \overrightarrow{b}) = \alpha\overrightarrow{a} + \alpha\overrightarrow{b}$ (распределительное свойство числового сомножителя относительно суммы векторов);
- $(\alpha + \beta)\overrightarrow{a} = \alpha\overrightarrow{a} + \beta\overrightarrow{a}$ (распределительное свойство векторного сомножителя относительно суммы чисел);
- $(\alpha\beta)\overrightarrow{a} = \alpha(\beta\overrightarrow{a})$ (сочетательное свойство числовых сомножителей);
- если вектор $\overrightarrow{b}$ коллинеарен ненулевому вектору $\overrightarrow{a}$, то существует вещественное число β, такое, что $\overrightarrow{b} = \beta\overrightarrow{a}$.

Линейной комбинацией векторов $\vec{a}$ **и** $\vec{b}$ **называется вектор** $\vec{c} = \alpha\vec{a} + \beta\vec{b}$. При этом числа α и β называются **коэффициентами разложения** вектора $\vec{c}$ по векторам $\vec{a}$ и $\vec{b}$.

Если два вектора $\vec{r}_1$ и $\vec{r}_2$ заданы своими координатами (x_1, y_1, z_1) и (x_2, y_2, z_2), то операции над ними легко выразить через эти координаты:

- $\vec{r}_1 + \vec{r}_2 = \vec{r} = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$;
- $\vec{r}_1 - \vec{r}_2 = \vec{r} = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$;
- $\alpha\vec{r}_1 = \vec{r} = (\alpha x_1, \alpha y_1, \alpha z_1)$.

Векторы $\vec{a}$, $\vec{b}$ и $\vec{c}$ **называются компланарными**, если они лежат в одной плоскости.

Векторы называются линейно независимыми, если равенство нулю их линейной комбинации возможно только в случае равенства нулю коэффициентов α и β.

Справедливы следующие свойства:

- Каковы бы ни были неколлинеарные векторы $\vec{a}$ и $\vec{b}$, для любого вектора $\vec{c}$, лежащего в одной плоскости с ними, существуют числа α и β, такие, что $\vec{c} = \alpha\vec{a} + \beta\vec{b}$, причем такая пара чисел для каждого вектора единственная. Такое представление вектора $\vec{c}$ называется разложением по векторам $\vec{a}$ и $\vec{b}$.
- Каковы бы ни были некомпланарные векторы $\vec{a}$, $\vec{b}$ и $\vec{c}$, для любого вектора $\vec{d}$ существуют числа α, β и γ, такие, что $\vec{d} = \alpha\vec{a} + \beta\vec{b} + \gamma\vec{c}$, причем эта тройка чисел для каждого вектора - единственная (разложение вектора $\vec{d}$ по векторам $\vec{a}, \vec{b}, \vec{c}$).
- Любые три вектора в системе координат плоскости являются линейно зависимыми.
- Любые четыре вектора в системе координат пространства являются линейно зависимыми.

Говорят, что пара линейно независимых векторов на плоскости (тройка линейно независимых векторов в пространстве) образуют **базис**, поскольку любой вектор может быть представлен в виде линейной комбинации этих векторов. Коэффициенты разложения вектора по базисным векторам называются **координатами вектора в этом базисе**. Если векторы базиса взаимно перпендикулярны и имеют единичную длину, то базис называется ортонормированным, а векторы базиса называются **ортами**. Таким образом, базис из единичных векторов, направленных вдоль осей декартовой системы координат, является ортонормированным.

Скалярным произведением векторов $\vec{r}_1$ и $\vec{r}_2$ называется число, равное произведению длин этих векторов на косинус угла между ними. Будем обозначать скалярное произведение векторов символом $(\vec{r}_1 \cdot \vec{r}_2)$. Тогда скалярное произведение можно выразить формулой

$(\overrightarrow{r}_1 \cdot \overrightarrow{r}_2 = |\overrightarrow{r}_1| \cdot |\overrightarrow{r}_2| \cos\alpha).$

Несложно доказать следующие свойства данной операции.

- Скалярное произведение двух ненулевых векторов равно нулю тогда и только тогда, когда эти векторы ортогональны.
- Если угол между двумя векторами острый, то скалярное произведение этих векторов положительно, если же угол тупой, то скалярное произведение отрицательно.
- $(\overrightarrow{r}_1 \cdot \overrightarrow{r}_2) = (\overrightarrow{r}_2 \cdot \overrightarrow{r}_1)$ (свойство коммутативности).
- $\alpha(\overrightarrow{r}_1 \cdot \overrightarrow{r}_2) = (\alpha\overrightarrow{r}_1 \cdot \overrightarrow{r}_2) = (\overrightarrow{r}_1 \cdot \alpha\overrightarrow{r}_2)$ (сочетательное относительно числового множителя свойство).
- $((\overrightarrow{r}_1 + \overrightarrow{r}_2) \cdot \overrightarrow{r}_3) = (\overrightarrow{r}_1 \cdot \overrightarrow{r}_3) + (\overrightarrow{r}_2 \cdot \overrightarrow{r}_3)$ (распределительное относительно суммы векторов свойство).
- Скалярное произведение вектора самого на себя равно квадрату длины вектора.

Приведем некоторые формулы, связанные с разложением вектора в декартовой системе координат.

Пусть векторы $\overrightarrow{r}_1$ и $\overrightarrow{r}_2$ заданы своими координатами (x_1, y_1, z_1) и (x_2, y_2, z_2). Тогда их скалярное произведение может быть вычислено по формуле

$$(\overrightarrow{r}_1 \cdot \overrightarrow{r}_2) = x_1x_2 + y_1y_2 + z_1z_2. \quad (3.1)$$

Отсюда следует условие перпендикулярности векторов:

$$x_1x_2 + y_1y_2 + z_1z_2 = 0.$$

И, наконец, косинус угла между векторами вычисляется по формуле

$$\cos\varphi = \frac{x_1x_2 + y_1y_2 + z_1z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2} \cdot \sqrt{x_2^2 + y_2^2 + z_2^2}.} \quad (3.2)$$

Теперь расстояние между двумя точками с координатами (x_1, y_1, z_1) и (x_2, y_2, z_2) можно выразить через скалярное произведение соответствующих векторов:

$$d = \sqrt{(\overrightarrow{r}_1 - \overrightarrow{r}_2)}.$$

Введем еще одно понятие, касающееся векторов. Три вектора называются упорядоченной тройкой, если указано, какой из этих векторов является первым, какой - вторым и какой - третьим. При записи тройки векторов будем располагать эти векторы в порядке их следования. Так, запись $\overrightarrow{b}\,\overrightarrow{a}\,\overrightarrow{c}$ означает, что первым вектором тройки является вектор $\overrightarrow{b}$, вторым - $\overrightarrow{a}$, третьим - $\overrightarrow{c}$.

Тройка векторов называется **правой** (**левой**), если после приведения к общему началу вектор $\overrightarrow{c}$ располагается по ту сторону от плоскости, содержащей векторы $\overrightarrow{a}$,

$\vec{b}$, откуда кратчайший поворот от $\vec{a}$ к $\vec{b}$ кажется **совершающимся против часовой стрелки** (**по часовой стрелке**).

Векторным произведением вектора $\vec{a}$ на вектор $\vec{b}$ называется вектор $\vec{c}$, обозначаемый символом $\vec{a} \times \vec{b}$ и удовлетворяющий следующим требованиям:

- длина вектора $\vec{c}$ равна произведению длин векторов $\vec{a}$, $\vec{b}$ на синус угла между ними, т.е.

$$|\vec{c}| = |\vec{a}| \cdot |\vec{b}| \sin\varphi;$$

- вектор $\vec{c}$ ортогонален векторам $\vec{a}$, $\vec{b}$;
- вектор $\vec{c}$ направлен так, что тройка векторов $\vec{a}\,\vec{b}\,\vec{c}$ является правой.

Приведем (без доказательства) основные свойства векторного произведения.

- $[\vec{a} \times \vec{b}] = -[\vec{b} \times \vec{a}]$(антисимметричность);
- $\alpha[\vec{a} \times \vec{b}] = [\alpha\vec{a} \times \vec{b}]$(сочетательное свойство относительно умножения на число);
- $[(\vec{a} + \vec{b}) \times \vec{c}] = [\vec{a} \times \vec{c}] + [\vec{b} \times \vec{c}]$(распределительное свойство относительно сложения);
- $[\vec{a} \times \vec{a}] = 0$для любого вектора $\vec{a}$.

Ясно, что векторное произведение двух коллинеарных векторов дает нулевой вектор. Выведем теперь формулу для векторного произведения. Пусть базисные векторы декартовой системы координат $\vec{i}, \vec{j}, \vec{k}$ образуют правую тройку. Тогда справедливы следующие соотношения:

$$[\vec{i} \times \vec{j}] = \vec{k} = -[\vec{j} \times \vec{i}], \quad [\vec{j} \times \vec{k}] = \vec{i} = -[\vec{k} \times \vec{j}],$$
$$[\vec{k} \times \vec{i}] = \vec{j} = -[\vec{i} \times \vec{k}]$$

Если заданы два вектора $\vec{r}_1 = x_1\vec{i} + y_1\vec{j} + z_1\vec{k}$ и $\vec{r}_2 = x_2\vec{i} + y_2\vec{j} + z_2\vec{k}$, то, учитывая свойства векторного произведения, отсюда легко вывести, что

$$\vec{r}_3 = [\vec{r}_1 \times \vec{r}_2] = x_3\vec{i} + y_3\vec{j} + z_3\vec{k},$$

где

$$x_3 = y_1z_2 - z_1y_2, \quad y_3 = z_1x_2 - x_1z_2, \quad z_3 = x_1y_2 - y_1x_2. \text{ (3.3)}$$

Уравнения прямой и плоскости

Уравнение прямой на плоскости в декартовой системе координат можно задать уравнением вида

$y = kx + b$
для случая, когда прямая не параллельна оси *OY*, и уравнением
$x = c$
для вертикальной прямой. Но прямая может быть также задана и другим способом.

Достаточно указать вектор направления этой прямой $\overrightarrow{l} = (l_x, l_y)$ и какую-нибудь точку $\overrightarrow{r}_0 = (x_0, y_0)$, лежащую на этой прямой. При этом точки, лежащие на прямой, могут быть заданы с использованием векторных операций в виде так называемого **параметрического уравнения прямой**

$\overrightarrow{r} = \overrightarrow{r}_0 + t\overrightarrow{l},$

в котором параметр *t* пробегает все значения числовой прямой. Координаты точки, соответствующей некоторому значению этого параметра, определяются соотношениями

$x = x_0 + tl_x, \quad y = y_0 + tl_y.$ (3.4)

Прямую в пространстве тоже можно задавать параметрическим уравнением, которое очень легко получить из предыдущего простым переходом от двумерных векторов к трехмерным. Пусть $\overrightarrow{l} = (l_x, l_y, l_z), \ \overrightarrow{r}_0 = (x_0, y_0, z_0)$. Тогда это уравнение будет определять прямую в пространстве, а координаты точек этой прямой будут определяться формулами

$x = x_0 + tl_x, \quad y = y_0 + tl_y, \quad z = z_0 + tl_z, \quad -\infty < t < +\infty.$ (3.5)

Как известно из элементарной геометрии, через любые три точки в пространстве проходит плоскость. С другой стороны, через каждую точку плоскости можно провести единственную прямую, перпендикулярную данной плоскости. При этом все эти прямые будут параллельны друг другу, а значит, они имеют общий вектор направления. Этот вектор будем называть **нормалью к плоскости**. Если длина вектора равна единице, мы будем называть его **единичной нормалью**. В компьютерной графике часто приходится решать задачу построения нормали к некоторой плоскости, заданной тремя точками, а также задачи пересечения прямой с плоскостью и двух плоскостей.

Плоскость в пространстве можно задать, указав вектор нормали к ней и какую-либо точку, принадлежащую данной плоскости. Пусть $\overrightarrow{n} = (n_1, n_2, n_3)$- вектор единичной нормали, а $\overrightarrow{r}_0 = (x_0, y_0, z_0)$- некоторая точка на плоскости. Тогда для любой точки $\overrightarrow{r} = (x, y, z)$, лежащей на плоскости, вектор $\overrightarrow{r} - \overrightarrow{r}_0$ будет ортогонален вектору нормали, а следовательно, выполняется равенство

$((\overrightarrow{r} - \overrightarrow{r}_0) \cdot \overrightarrow{n}) = 0.$
Раскрывая это выражение в координатном виде, получаем
$n_1x + n_2y + n_3z - n_1x_0 - n_2y_0 - n_3z_0 = 0.$
Теперь перепишем это уравнение в виде
$n_1x + n_2y + n_3z + d = 0,$ (3.6)
где $d = -n_1x_0 - n_2y_0 - n_3z_0$. Это уравнение называется каноническим уравнением плоскости. При этом совершенно ясно, что если все это уравнение умножить на какой-либо отличный от нуля множитель, то оно будет описывать ту же самую плоскость, т.е. коэффициенты n_1, n_2, n_3 для каждой плоскости задаются с точностью до произвольного ненулевого множителя. Но если при этом вектор $\overrightarrow{n}$ имеет единичную длину, то $|d|$ задает расстояние от начала координат до данной плоскости.

В алгоритмах компьютерной графики довольно часто приходится сталкиваться с задачей построения плоскости, проходящей через три заданные точки. Пусть три точки $\overrightarrow{r}_1$, $\overrightarrow{r}_2$ и $\overrightarrow{r}_3$, не лежащие на одной прямой, имеют координатами $(x_1, y_1, z_1), (x_2, y_2, z_2)$ и (x_3, y_3, z_3). Для канонического уравнения необходимо построить нормаль к плоскости, что легко можно осуществить, используя операцию векторного произведения. Поскольку векторы $\overrightarrow{v}_1 = \overrightarrow{r}_2 - \overrightarrow{r}_1$ и $\overrightarrow{v}_2 = \overrightarrow{r}_3 - \overrightarrow{r}_1$ лежат в искомой плоскости, то вектор $\overrightarrow{N} = \overrightarrow{v}_1 \times \overrightarrow{v}_2$ будет ортогонален этой плоскости. Пусть $\overrightarrow{N} = (N_x, N_y, N_z)$, тогда уравнение плоскости будет иметь вид

$$N_x x + N_y y + N_z z + D = 0.$$

Остается определить значение D. Так как точка $\overrightarrow{r}_1$ принадлежит этой плоскости, то ее координаты должны удовлетворять полученному уравнению. Подставим их в уравнение и получим

$$N_x x_1 + N_y y_1 + N_z z_1 + D = 0,$$

следовательно

$$D = -N_x x_1 - N_y y_1 - N_z z_1,$$

и после подстановки окончательно получим:

$$N_x(x - x_1) + N_y(y - y_1) + N_z(z - z_1) = 0 \quad (3.7)$$

В большинстве алгоритмов, использующих плоскости, достаточно знать нормаль к ней и какую-либо точку, принадлежащую плоскости. Очевидно, что по аналогии можно вывести каноническое уравнение прямой на плоскости, если задана нормаль к ней и принадлежащая прямой точка.

Аналитическое представление кривых и поверхностей

Пусть на плоскости задана декартова система координат.

Кривая на плоскости - это геометрическое место точек (x, y), удовлетворяющих уравнению

$$F(x, y) = 0 \quad (3.10)$$

где F - функция двух переменных. Ясно, что далеко не каждая функция будет задавать линию. Так, например, уравнению

$$x^2 + y^2 + 1 = 0$$

не удовлетворяет ни одна точка плоскости, а уравнению

$$x^2 + y^2 = 0$$

удовлетворяет только одна точка $(0, 0)$.

Для аналитического представления кривой во многих случаях удобнее задавать кривую параметрическими уравнениями, используя вспомогательную переменную (параметр) t:

$$x = \varphi(t), \quad y = \psi(t), t \in [a, b], \quad (3.11)$$

где φ и ψ - непрерывные функции на заданном интервале изменения параметра. Если функция $\varphi(t)$ такова, что можно выразить t через $x(t = \varphi^{-1}(x))$, то от параметрического представления кривой легко перейти к уравнению (3.10):
$y = \psi(\varphi^{-1}(t)) = 0.$
Систему уравнений (3.11) можно записать в векторном виде:
$\overrightarrow{r} = \overrightarrow{f}(t), \quad \overrightarrow{r} = (x, y), \quad \overrightarrow{f}(t) = (\varphi(t), \psi(t)).$

Отрезок прямой представляет собой частный случай кривой, причем параметрическое представление его может иметь вид

$x = t, \quad y = at + b, \quad t \in [t_1, t_2]$
или
$x = at + b, y = t, t \in [t_1, t_2]$

Окружность радиуса r с центром в точке (x_0, y_0) может быть представлена параметрическими уравнениями

$x = x_0 + r \cdot \cos t, \quad y = y_0 + r \cdot \sin t, t \in [0, 2\pi].$

Перейдем к трехмерному пространству с заданной декартовой системой координат.

Поверхность в пространстве - это геометрическое место точек (x, y, z), удовлетворяющих уравнению вида

$F(x, y, z) = 0.$ (3.12)

Так же как и в случае кривой на плоскости, не всякая функция F описывает какую-либо поверхность. Например, уравнению

$x^2 + y^2 + z^2 + 1 = 0$
не удовлетворяет ни одна точка пространства. Поверхность также может быть задана в **параметрическом виде**, но в отличие от кривой для этого требуются две вспомогательные переменные (параметры):
$x = \varphi(u, v), \quad y = \psi(u, v), \quad z = \zeta(u, v), \quad u \in [a, b], \quad v \in [c, d].$ (3.13)

Например, сфера радиуса r с центром в точке (x_0, y_0, z_0) может быть задана уравнением

$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 = 0$
либо же параметрическими уравнениями
$x = x_0 + r \cdot \cos u \cdot \cos v, \quad y = y_0 + r \cdot \sin u, \quad z = z_0 + r \cdot \cos u \cdot \sin v.$

Кривую в пространстве можно описать как пересечение двух поверхностей, т.е. с помощью системы уравнений

$F_1(x, y, z) = 0, \quad F_2(x, y, z) = 0$ (3.14)
или параметрическими уравнениями вида
$x = \varphi(t), \quad y = \psi(t), z = \zeta(t), \quad t \in [a, b].$ (3.15)

Пересечение луча с плоскостью и сферой

Прямая на плоскости и в пространстве является бесконечной в обе стороны. **Лучом** называется полупрямая, т.е. множество всех точек прямой, лежащих по одну сторону от заданной ее точки, называемой началом луча. Луч будем задавать в параметрическом виде, как это было описано в одном из предыдущих разделов. Пусть $\overrightarrow{l} = (l_x, l_y, l_z)$ - направляющий вектор прямой, а $\overrightarrow{r}_0 = (x_0, y_0, z_0)$ - начальная точка. Тогда координаты точек луча будут определяться формулами

$$x = x_0 + tl_x, \quad y = y_0 + tl_y, \quad z = z_0 + tl_z. \text{ (3.8)}$$

Будем считать, что направляющий вектор единичный, т.е. $l_x^2 + l_y^2 + l_z^2 = 1$.

Сначала рассмотрим задачу о нахождении точки пересечения луча с плоскостью, заданной каноническими уравнением

$$n_1 x + n_2 y + n_3 z + d = 0. \text{ (3.9)}$$

Вектор нормали $\overrightarrow{n} = (n_1, n_2, n_3)$ тоже будем считать единичным. Сначала надо определить значение параметра t, при котором луч пересекает плоскость. Для этого подставим координаты из формулы (3.8) в уравнение (3.9) и получим

$$n_1(x_0 + tl_x) + n_2(y_0 + tl_y) + n_3(z_0 + tl_z) + d = 0,$$

откуда легко определить, что луч пересекает плоскость в точке со значением

$$t_0 = -\frac{(\overrightarrow{r}_0 \cdot \overrightarrow{n}) + d}{(\overrightarrow{l} \cdot \overrightarrow{n})}.$$

Очевидно, что такая точка существует только при условии $(\overrightarrow{l} \cdot \overrightarrow{n}) \neq 0$. В свою очередь, эта величина обращается в нуль только в случае, когда векторы $\overrightarrow{l}$ и $\overrightarrow{n}$ ортогональны друг другу.

Пусть теперь нам задана сфера с центром в точке $\overrightarrow{r}_c = (x_c, y_c, z_c)$ и радиусом d. Тогда уравнение сферы будет иметь вид

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = d^2.$$

Подставив сюда координаты луча из уравнения (3.9), получим, что параметр, при котором луч пересекает сферу, должен удовлетворять квадратному уравнению

$$at_0^2 + bt_0 + c = 0,$$

где $a = |\overrightarrow{r}_c|^2,\ b = 2((\overrightarrow{r}_0 - \overrightarrow{r}_c) \cdot \overrightarrow{l}),\ c = |\overrightarrow{r}_0 - \overrightarrow{r}_c|^2 - d^2$. Определим корни этого уравнения. Если дискриминант $D = \frac{b^2}{4} - c \geq 0$, то корни существуют. Их может быть либо два $(D > 0)$, либо один $(D = 0)$. В первом случае имеем две точки пересечения, во втором - одну (луч касается сферы). Соответствующие значения параметра определяются соотношением

$$t_{1,2} = -\frac{b}{2} \mp \sqrt{D}$$

Интерполяция функций одной и двух переменных

Помимо функций, заданных аналитически (т. е. с помощью элементарных функций, значения которых легко могут быть вычислены в любой точке области определения), на практике часто приходится иметь дело с таблично заданными функциями. В этом случае функция задается своими значениями на некотором дискретном множестве точек (**узлов**) из области определения. Если необходимо получить значение функции в какой-либо точке, не совпадающей с узлом, используют различные методы приближенного вычисления, которые основываются на некоторых априорных предположениях относительно этой функции. При этом сама процедура вычисления называется **интерполяцией** в случае, когда точка принадлежит заданной области, и экстраполяцией, если она лежит вне области.

В качестве предположений о характере дискретно заданной функции наиболее часто используемой и простой является то, что она кусочно- линейная, т. е. что в промежутках между узлами она ведет себя в соответствии с линейным законом. Тогда интерполяция называется **линейной**, и этот метод мы будем довольно часто применять в алгоритмах компьютерной графики.

Пусть на плоскости задана система координат XOY и отрезок $[x_1, x_2]$ на оси OX , на концах которого заданы значения y_1, y_2 некоторой **линейной** функции (рис. 3.3). Тогда для любой точки x внутри заданного отрезка соответствующее значение y вычисляется по формулам

$$y = ty_1 + (1-t)y_2, \qquad t = (x_2 - x)/(x_2 - x_1).$$

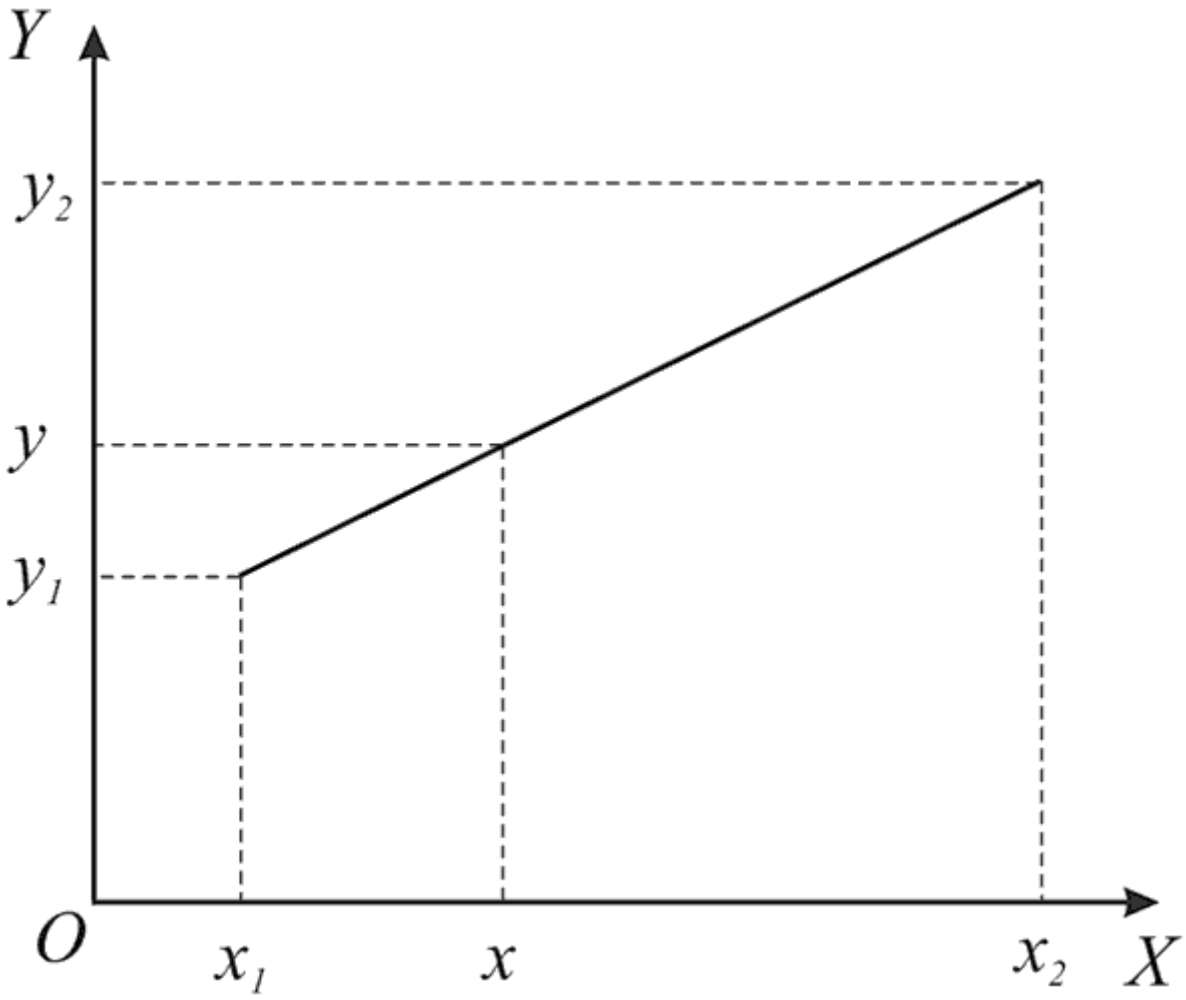


Рис. 3.3. Линейная интерполяция функции одной переменной

Обратимся теперь к задаче интерполяции функций двух переменных. В этом случае наиболее простой также является интерполяция по трем заданным точкам опять же с помощью кусочно-линейной функции. Пусть на плоскости задан треугольник с вершинами $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ и заданы значения функции в этих точках z_1, z_2, z_3. Тогда три точки (x_i, y_i, z_i) определяют в пространстве треугольник, который является плоской фигурой. Предполагается, что площадь треугольника больше нуля, или, как говорят, **треугольник невырожденный**. Для определения значения функции в произвольной точке (x, y), лежащей внутри треугольника, воспользуемся так называемыми **барицентрическими координатами** (α, β, γ) этой точки. Геометрический смысл этих координат заключается в том, что они равны отношению площадей треугольников, изображенных на рис. 3.4:

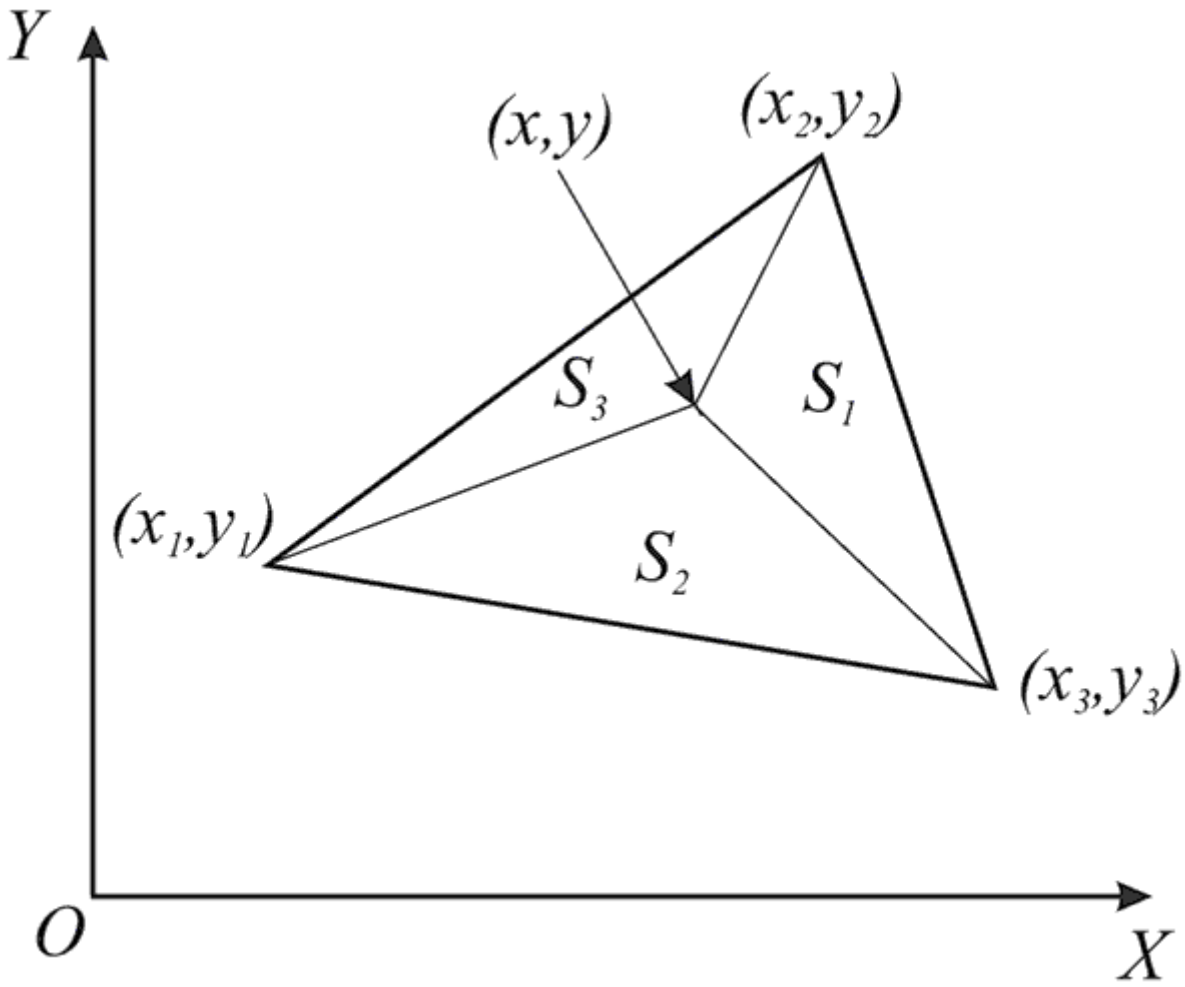


Рис. 3.4. Линейная интерполяция функции двух переменных

$$\alpha = \frac{S_1}{S},\ \beta = \frac{S_2}{S},\ \gamma = \frac{S_3}{S},\ S = S_1 + S_2 + S_3.$$

Эти числа неотрицательны и удовлетворяют следующим соотношениям:

$$\left.\begin{aligned} &\alpha + \beta + \gamma = 1 \\ &\alpha x_1 + \beta x_2 + \gamma x_3 = x \\ &\alpha y_1 + \beta y_2 + \gamma y_3 = y \end{aligned}\right\}.$$

Эти соотношения будем рассматривать как уравнения для нахождения чисел (α, β, γ).

Определитель этой системы уравнений есть

$$\Delta = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1),$$

и он по модулю равен удвоенной площади треугольника, поэтому $\Delta \neq 0$, следовательно, система имеет единственное решение при любой правой части. Воспользуемся формулами Крамера и выпишем вид этого решения:

$$\alpha = \frac{\Delta_1}{\Delta}, \quad \beta = \frac{\Delta_2}{\Delta}, \quad \gamma = 1 - \alpha - \beta,$$

где

$$\Delta_1 = \begin{vmatrix} 1 & 1 & 1 \\ x & x_2 & x_3 \\ y & y_2 & y_3 \end{vmatrix} = (x_2 y_3 - x_3 y_2) + x(y_2 - y_3) + y(x_3 - x_2),$$

$$\Delta_2 = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x & x_3 \\ y_1 & y & y_3 \end{vmatrix} = (x_3 y_1 - x_1 y_3) + x(y_3 - y_1) + y(x_1 - x_3).$$

После того как получены барицентрические координаты точки (x, y), значение функции в ней рассчитывается по формуле
$z = \alpha z_1 + \beta z_2 + \gamma z_3.$

Существуют хорошо разработанные методы гладкой интерполяции функций. Особенно часто при интерполяции кривых и поверхностей используются **сплайн-функции**, которые гладко "склеиваются" из полиномов. Среди них следует выделить **кубические сплайны**, которые строятся из полиномов третьей степени. Они широко используются в инженерной геометрии благодаря простоте их вычисления и другим полезным свойствам. Мы их рассмотрим подробнее в последующих главах.

Матрицы

Для выполнения преобразований векторов в пространстве мы будем использовать матричный метод. **Матрицей** размерности $n \times n$ называется таблица чисел вида

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

В дальнейшем будем использовать сокращенную запись для матрицы: $A = (a_{ij})$.
Строки матрицы $A_i = (a_{i1}, a_{i2}, \dots, a_in)$ будем называть **вектор-строками** (поскольку каждая из них определяет вектор), а столбцы A^j - **вектор-столбцами**. Матрицы являются эффективным инструментом для выполнения преобразований на плоскости и в пространстве. В этих случаях применяются матрицы размерности 2×2 и 3×3.

Сначала введем ряд операций над матрицами и векторами.

Пусть заданы матрицы $A = (a_{ij})$ и $B = (b_{ij})$. **Суммой матриц** называется матрица $C = (c_{ij})$, элементами которой являются $c_{ij} = a_{ij} + b_{ij}$.

Определим также операцию **умножения матрицы на число**. Результатом умножения матрицы $A = (a_{ij})$ на число α является матрица $B = (b_{ij})$, элементы которой $b_{ij} = \alpha a_{ij}$.

Произведением двух матриц $A = (a_{ij})$ и $B = (b_{ij})$ называется матрица $C = (c_{ij})$, элементы которой определяются следующим образом:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Произведение матриц некоммутативно, т.е. в общем случае $A \cdot B \neq B \cdot A$.

Предыдущие определения мы вводили для матриц произвольной размерности. Следующие операции будут связаны с векторами, и мы будем подразумевать, что $n = 2$ или $n = 3$. Пусть задана матрица $A = (a_{ij})$ и вектор $\overrightarrow{r} = (x_1, \ldots, x_n)$.

Результатом **умножения матрицы на вектор** является вектор $\overrightarrow{r}_0 = (x_1^0, \ldots, x_n^0)$, координаты которого вычисляются как скалярное произведение строки матрицы на вектор:

$$x_i^0 = \sum_{k=1}^{n} a_{ik} x_k \equiv (A_i \cdot \overrightarrow{r}).$$

Если матрица $B = (b_{ij})$ получена из матрицы $A = (a_{ij})$ путем замены всех вектор-строк на вектор-столбцы, т.е. $b_{ij} = a_{ji},\ i,\ j = 1, \ldots, n$, то ее называют транспонированной матрицей A и обозначают A^T.

Аналогичным образом определяется **умножение вектора на матрицу**, только в этом случае вектор скалярно умножается на вектор-строки матрицы. Матрица вида

$$E = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$$

называется **единичной** и обладает следующими свойствами:

- $A \cdot E = E \cdot A = A$ для любой матрицы A.
- $\overrightarrow{r} = E \cdot \overrightarrow{r}$ для любого вектора $\overrightarrow{r}$.
- Если для матрицы A существует матрица B, такая, что $A \cdot B = E$, то B называется обратной матрицей к A и обозначается A^{-1}. При этом $A \cdot A^{-1} = A^{-1} \cdot A = E$, и для любого вектора $\overrightarrow{r}$ получаем соотношения: если $\overrightarrow{r}' = A \cdot \overrightarrow{r}$, то $\overrightarrow{r} = A^{-1} \cdot \overrightarrow{r}'$.
- Если для матриц A и B существуют обратные матрицы, то существует и обратная матрица для их произведения и $(a \cdot B)^{-1} = B^{-1} \cdot A^{-1}$.

Благодаря операции умножения матрицы на вектор любая матрица определяет преобразование в пространстве, по которому каждому вектору сопоставляется некоторый другой по вполне определенному закону.

Отметим, что для геометрических преобразований удобно использовать матрицы размерностью на единицу больше, чем размерность пространства, но об этом подробнее речь пойдет в следующей главе.

Геометрические преобразования (перенос, масштабирование, вращение)

Геометрические объекты на плоскости и в пространстве можно подвергать ряду различных преобразований. Наиболее употребительными в задачах компьютерной графики являются:

- перемещение (параллельный перенос);
- изменение размеров (масштабирование);
- повороты вокруг некоторой точки на плоскости или некоторой оси в пространстве (вращение).

В дальнейшем мы часто будем отождествлять точки пространства с радиус-вектором, определяемым этой точкой.

Сначала рассмотрим преобразования на плоскости, или **двумерные преобразования**.

Параллельный перенос объекта сводится к перемещению всех его точек на одно и то же расстояние d в одном и том же направлении, заданном определенным вектором $\overrightarrow{v}$. Если этот вектор имеет длину d, то операция переноса может быть реализована путем сложения всех точек объекта с вектором $\overrightarrow{v}$. Довольно просто доказать, что при такой операции сохраняются расстояния между точками и, как следствие, углы между отрезками. Понятно также, что отрезки прямых перейдут в отрезки прямых. Поэтому при переносе многоугольника нет необходимости подвергать этой операции бесконечное множество точек, достаточно просто перенести вершины, а затем соединить их отрезками.

Масштабирование объекта можно реализовать путем умножения координат всех его точек на некоторое число. Пусть имеются точки с координатами (x_1, y_1) и (x_2, y_2), над которыми выполняется такое преобразование. Результатом будут новые точки с координатами $(S_x x_1, S_y y_1)$ и $(S_x x_2, S_y y_2)$. Если $S_x = S_y = S_0$, то несложно доказать, что обе точки переместятся вдоль прямых, проходящих через саму точку и начало координат, т.е. в направлении своего же радиус-вектора (рис. 3.5). При этом расстояние между новыми точками будет в S_0 раз отличаться от прежнего, но углы между отрезками сохранятся (это можно показать, если выразить косинус угла через скалярное произведение векторов). Ясно, что если коэффициент масштабирования S_0 больше единицы, соответствующий отрезок растягивается, а если меньше, то сжимается. Кроме того, при таком преобразовании объект смещается.

В случае, когда $S_x \neq S_y$, расстояния между точками изменятся неравномерно, поскольку растяжения в горизонтальном и вертикальном направлениях будут различными. Углы между отрезками также не сохранятся (рис. 3.6).

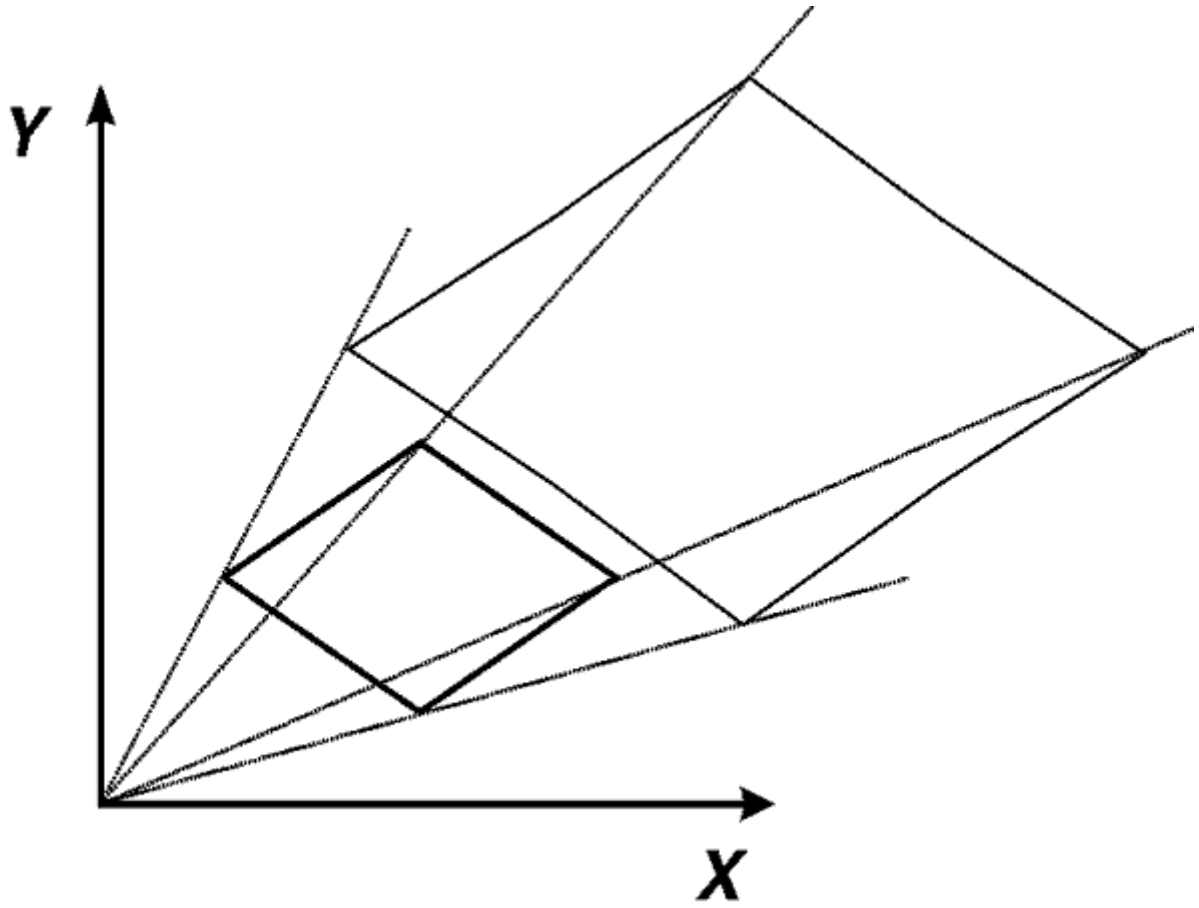


Рис. 3.5. Масштабирование с сохранением углов

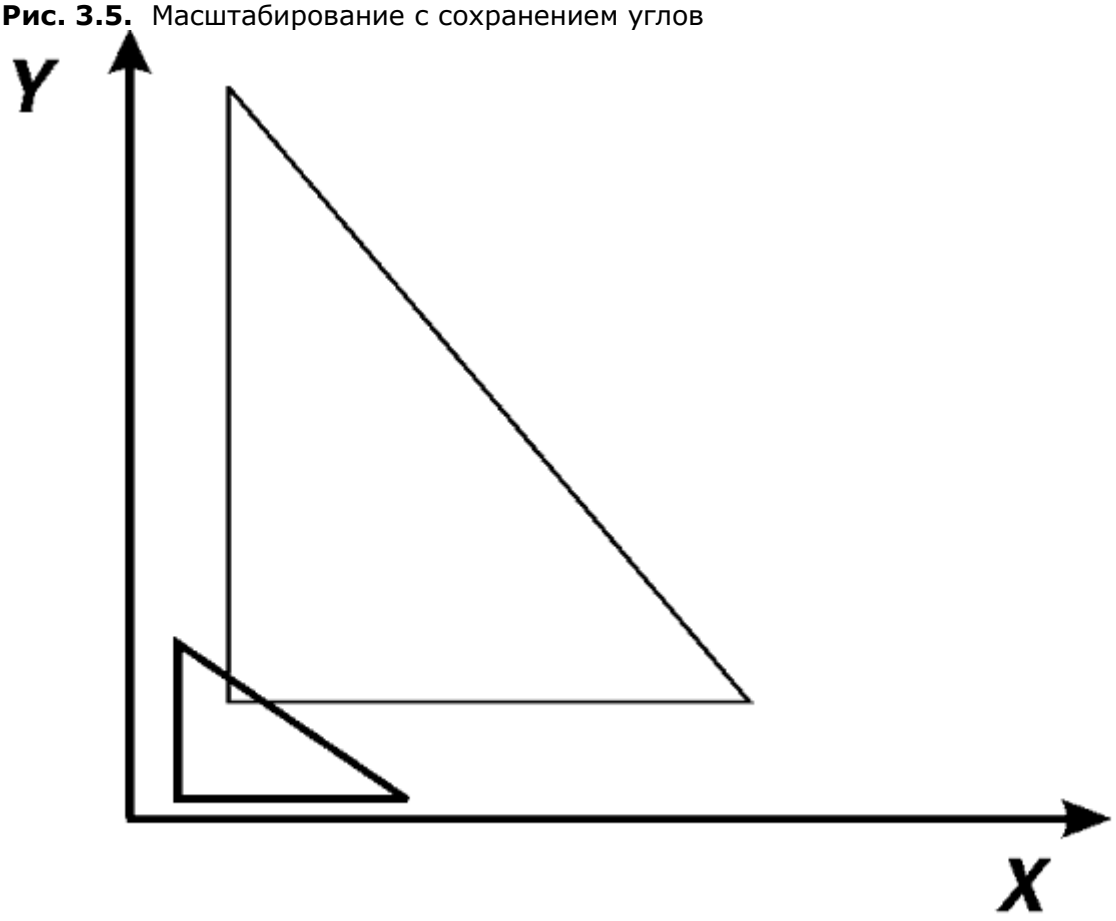


Рис. 3.6. Масштабирование с искажением углов

Вращения в плоскости перемещают точки по дуге окружности, центр которой находится в начале координат. Рассмотрим сначала движение одной точки при повороте на угол α (положительным является направление против часовой стрелки), т. е. поворот радиус-вектора на угол (рис. 3.7). Пусть точка располагалась на расстоянии r от начала координат, а ее радиус-вектор составлял угол β с осью абсцисс. Тогда координаты точки определяются формулами

$$x = r\cos\beta, \quad y = r\sin\beta.$$

После поворота вектор будет составлять угол $\beta + \alpha$, а новые координаты точки будут определяться соотношениями

$$\begin{aligned} x' &= r\cos(\beta+\alpha) = r\cos\beta\cos\alpha - r\sin\beta\sin\alpha = x\cos\alpha - y\sin\alpha \\ y' &= r\sin(\beta+\alpha) = r\sin\beta\cos\alpha + r\cos\beta\sin\alpha = x\sin\alpha + y\cos\alpha \end{aligned}. \quad (3.7)$$

Можно показать, что при таком преобразовании сохраняются расстояния между точками, а следовательно, и углы между отрезками.

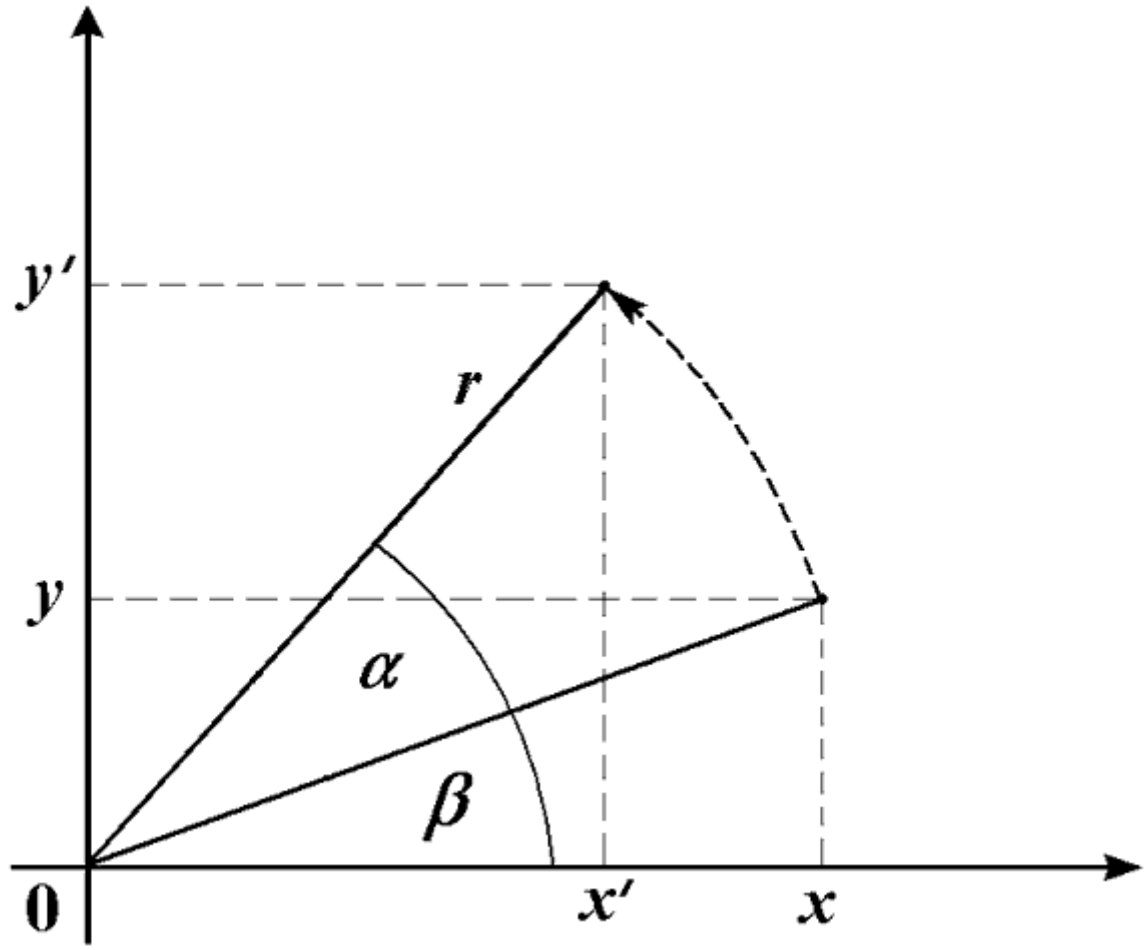


Рис. 3.7. Поворот на плоскости

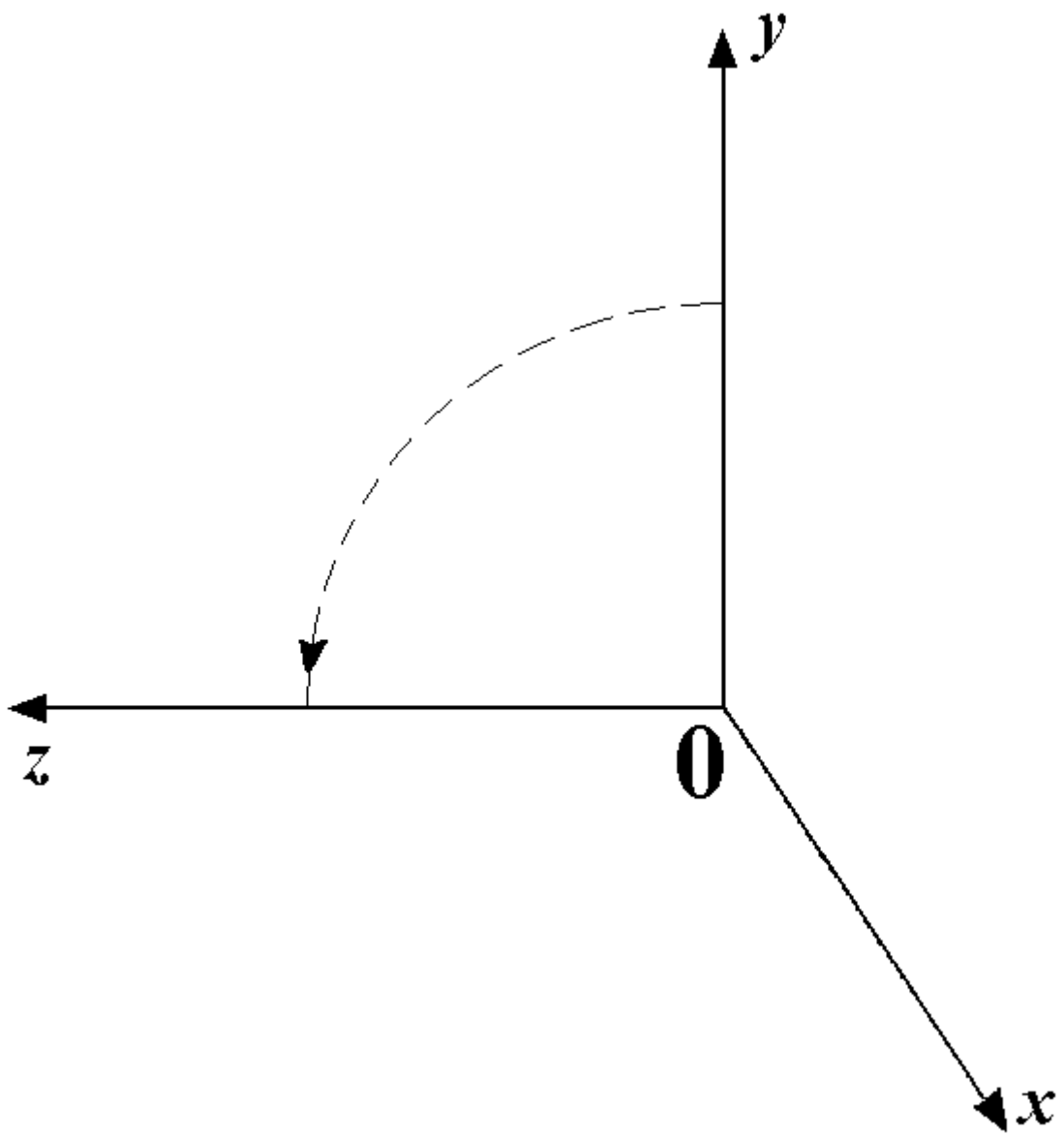


Рис. 3.8. Поворот в пространстве

В случае трехмерного пространства рассуждения, касающиеся переноса и масштабирования, полностью аналогичны, только они распространяются на третью координату точек. С вращением же дело обстоит иначе, поскольку здесь вращательное движение есть перемещение вдоль поверхности сферы и поворот на какой-то угол относительно точки нельзя определить однозначно. Но перемещение из одной точки сферы в другую всегда можно осуществить последовательностью поворотов относительно осей координат, поэтому выведем формулы для этих трех вращений.

При повороте относительно оси OX на угол α у всех точек координата x остается неизменной. Если смотреть на плоскость YOZ со стороны конца оси OX, то оси будут расположены так, как показано на рис. 3.8. Положительным считается поворот от оси OY к оси OZ. Если воспользоваться формулами для плоских поворотов, то координаты y' и z' новой точки определяются выражениями

$$y' = y\cos\alpha - z\sin\alpha$$
$$z' = y\sin\alpha + z\cos\alpha.$$

Формулы поворота относительно оси OZ полностью совпадают с теми, которые были выведены для плоского случая, а поворот относительно оси OY выглядит так:

$$x' = x\cos\alpha + z\sin\alpha$$
$$z' = -x\sin\alpha + z\cos\alpha.$$

Во всех этих формулах следует обратить внимание на знаки, так как они зависят от того, какой поворот считается положительным (в данном случае мы имеем дело с правой тройкой базисных векторов).

Преобразования масштабирования и поворота на плоскости и в пространстве можно выразить с помощью матриц. Если заданы коэффициенты масштабирования S_x, S_y, S_z, то преобразование точки осуществляется посредством умножения матрицы на ее радиус-вектор,

$$\overrightarrow{r}' = S \cdot \overrightarrow{r} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} S_x x \\ S_y y \\ S_z z \end{pmatrix} . \quad (3.8)$$

Двумерный случай выглядит подобным же образом.

Поворот на плоскости можно осуществить с помощью матрицы

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} . \quad (3.9)$$

И наконец, повороты в пространстве относительно осей координат можно выполнить с помощью трех матриц вращения

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \quad R_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix},$$
$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.10)$$

Нетрудно проверить, что для матриц вращения справедливо соотношение

$$R(-\alpha) = R^{(}-1)(\alpha)$$

Для выполнения последовательных поворотов вокруг осей на углы α, β, γ можно создать матрицу преобразования путем перемножения трех матриц:

$$R = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha).$$

Использование этой матрицы даст заметную экономию в вычислениях по сравнению с последовательными умножениями на каждую из трех матриц вращения.

Переход в другую систему координат

Мы рассмотрели преобразования геометрических объектов, заданных в определенной декартовой системе координат. Но во многих случаях удобно рассматривать те же объекты в другой системе координат, поскольку их описание может стать более простым. Самый простой пример - задание координат параллелепипеда: проще всего это сделать в системе координат, совмещенной с одной из его вершин с осями, направленными вдоль ребер. В связи с этим остановимся на вопросе, как изменятся координаты точки при переходе от одной декартовой системы координат к другой.

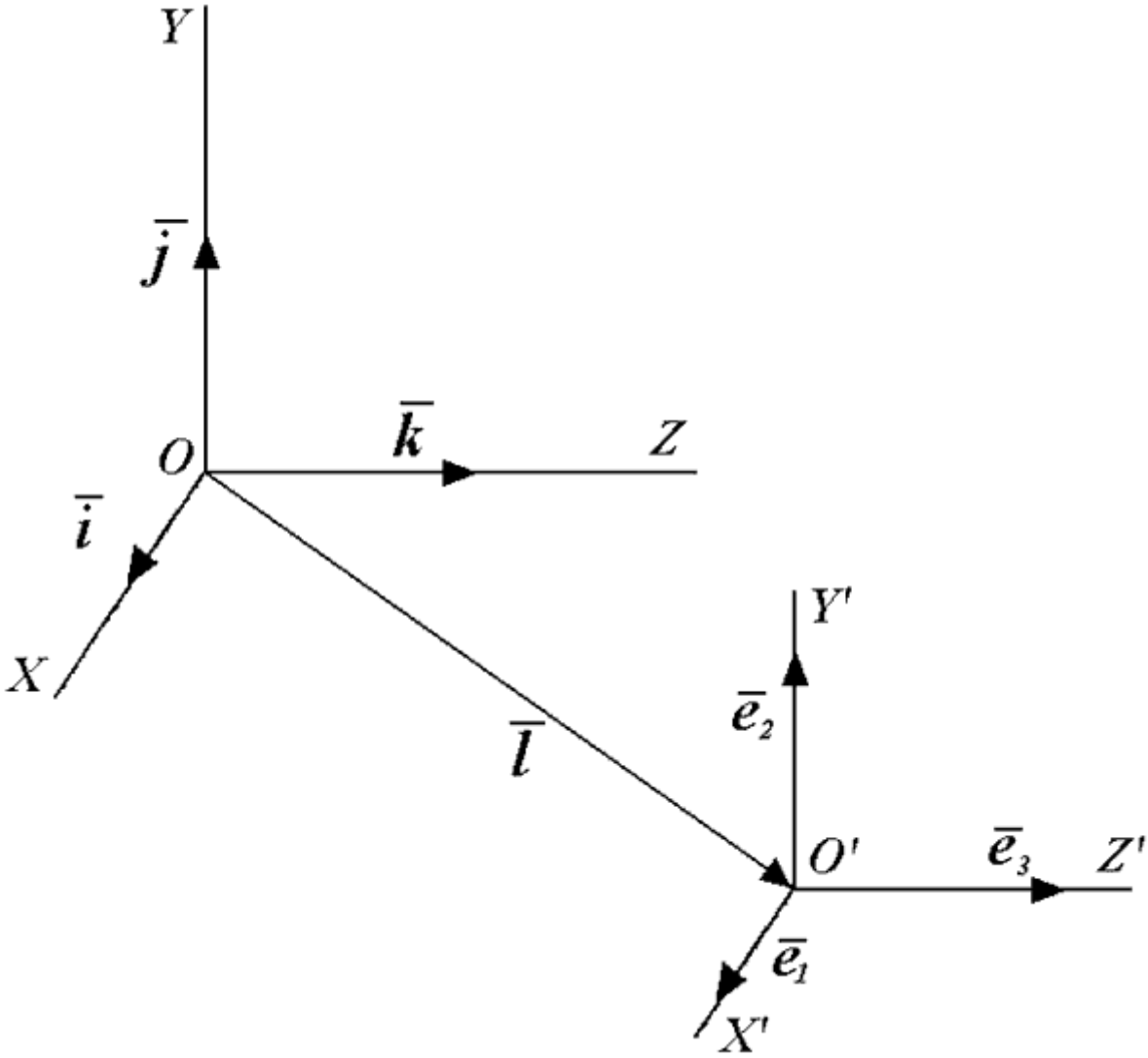


Рис. 3.9. Две системы координат в пространстве

Пусть единичные орты первой системы координат обозначаются $\vec{i}, \vec{j}, \vec{k}$, а оси координат - OX, OY, OZ. Введем еще одну систему координат, единичные орты которой обозначим $\vec{e}_1, \vec{e}_2, \vec{e}_3$, а оси координат - $O'X', O'Y', O'Z'$. Эта система имеет свое начало координат и свои направления осей. Считаем, что в обеих системах координат орты образуют левую тройку (рис. 3.9).

Сначала рассмотрим ситуацию, когда точка O' совпадает с точкой O. Векторы $\vec{e}_1, \vec{e}_2, \vec{e}_3$ можно задать в первой системе координат, разложив их по векторам $\vec{i}, \vec{j}, \vec{k}$:

$$\left.\begin{aligned}\overrightarrow{e}_1 &= e_x^1\overrightarrow{i} + e_y^1\overrightarrow{j} + e_z^1\overrightarrow{k}\\ \overrightarrow{e}_2 &= e_x^2\overrightarrow{i} + e_y^2\overrightarrow{j} + e_z^2\overrightarrow{k}\\ \overrightarrow{e}_3 &= e_x^3\overrightarrow{i} + e_y^3\overrightarrow{j} + e_z^3\overrightarrow{k}\end{aligned}\right\}.$$

Если в первой системе точка $\overrightarrow{M}$ имеет координаты (x, y, z), а во второй системе - (x', y', z'), то, очевидно,

$$x'\overrightarrow{e}_1 + y'\overrightarrow{e}_2 + z'\overrightarrow{e}_2 = x\overrightarrow{i} + y\overrightarrow{j} + z\overrightarrow{k}.$$

Умножая скалярно это соотношение на векторы $\overrightarrow{e}_1, \overrightarrow{e}_2, \overrightarrow{e}_3$, получим связь между значениями координат в разных системах:

$$\left.\begin{aligned}x' &= e_x^1 x + e_y^1 y + e_z^1 z\\ y' &= e_x^2 x + e_y^2 y + e_z^2 z\\ z' &= e_x^3 x + e_y^3 y + e_z^3 z\end{aligned}\right\}.$$

Эти соотношения можно записать в матричном виде

$$\left(x'//y'//z'\right) = \left(e_x^1 \quad e_y^1 \quad e_z^1//e_x^2 \quad e_y^2 \quad e_z^2//e_x^3 \quad e_y^3 \quad e_z^3\right)\cdot\left(x//y//z\right), \quad (3.11)$$

или в векторной записи

$$\overrightarrow{M'} = A\cdot\overrightarrow{M}.$$

Предположим, что вторая система координат получена из первой путем поворота на угол φ относительно оси OY. Тогда

$$\overrightarrow{e}_1 = R_y(\varphi)\cdot\begin{pmatrix}1\\0\\0\end{pmatrix} = \begin{pmatrix}\cos\varphi\\0\\-\sin\varphi\end{pmatrix}, \quad \overrightarrow{e}_2 = R_y(\varphi)\cdot\begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}0\\1\\0\end{pmatrix},$$

$$\overrightarrow{e}_3 = R_y(\varphi)\cdot\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}\sin\varphi\\0\\\cos\varphi\end{pmatrix},$$

следовательно

$$A = \begin{pmatrix}\cos\varphi & 0 & -\sin\varphi\\0 & 1 & 0\\\sin\varphi & 0 & \cos\varphi\end{pmatrix} = R_y(-\varphi) = R_y^{-1}(\varphi).$$

Таким образом, при поворотах системы координат новые координаты точек получаются путем умножения матрицы поворота на противоположный угол на вектор исходных координат.

Если новая система координат получена из старой путем сдвига на вектор $\overrightarrow{l} = (l_x, l_y, l_z)$, то очевидно, что новые координаты точки задаются формулами

$$x' = x - l_x, \quad y' = y - l_y, \quad z' = z - l_z.$$

Теперь можно рассмотреть композицию двух преобразований системы координат - переноса и вращения. Тогда координаты точек преобразуются по формуле

$$\overrightarrow{M'} = A \cdot (\overrightarrow{M'} - \overrightarrow{l}).$$ (3.12)

Задача вращения относительно произвольной оси

Вращение относительно произвольной оси также можно реализовать посредством умножения матрицы на вектор, но предварительно эту матрицу надо построить. Предположим, что прямая проходит через начало координат и задана единичным вектором $\overrightarrow{l} = (l_x, l_y, l_z)$, и требуется выполнить поворот точки на угол α относительно нее. Для этого воспользуемся следующим алгоритмом:

1. Совместим прямую с осью OZ посредством поворота системы координат относительно оси OX на угол φ, а затем поворота относительно оси OY на угол ψ.
2. Выполним поворот относительно оси OZ на угол α.
3. Выполним повороты системы сначала относительно оси OY на угол $-\psi$, а затем относительно оси OX на угол $-\varphi$ (в обратном порядке по отношению к первым поворотам), тем самым возвращая ее в исходное положение.

Итоговая матрица преобразования, таким образом, является произведением нескольких матриц, а именно

$$R = R_x(\varphi) \cdot R_y(\psi) \cdot R_z(\alpha) \cdot R_y(-\psi) \cdot R_x(-\varphi).$$

Матрицы $R_y(-\psi), R_x(-\varphi)$ являются матрицами преобразования координат при поворотах системы координат, как было показано в предыдущем разделе. Определим сначала угол φ, который является углом между осью OZ и его проекцией вектора $\overrightarrow{l}$ на плоскость YOZ $\overrightarrow{l}_1 = (0, l_y, l_z)$. Пусть $d = \sqrt{l_y^2 + l_z^2}$ - длина этой проекции. Тогда $\cos\varphi = \frac{l_z}{d}$, $\sin\varphi = -\frac{l_y}{d}$, (синус отрицателен, поскольку поворот идет от оси OZ к оси OY, т.е. в отрицательном направлении). После поворота системы координат новыми координатами вектора $\overrightarrow{l}$ будут $l_x, 0, d$. Угол ψ - это угол между векторами $\overrightarrow{l}_1$ и $\overrightarrow{l}$, поэтому $\cos\psi = d, \sin\psi = l_x$. Теперь мы можем выписать вид матриц преобразования координат для каждого шага алгоритма, учитывая то, что матрицы преобразования координат при повороте системы координат обратны по отношению к соответствующим матрицам вращения:

$$R_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{l_z}{d} & -\frac{l_y}{d} \\ 0 & \frac{l_y}{d} & \frac{l_z}{d} \end{pmatrix}, \quad R_y(\psi =) \begin{pmatrix} d & 0 & -l_x \\ 0 & 1 & 0 \\ l_x & 0 & d \end{pmatrix},$$

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

Нетрудно убедиться, что последовательное умножение матриц $R_x(-\varphi)$ и $R_y(-psi)$ на вектор $\overrightarrow{l}$ дадут в результате вектор $(0,0,1)$, т.е. этот вектор действительно станет осью аппликат.

Остается только выписать окончательный вид матрицы R (для сокращения записи введем следующие обозначения: $c = \cos\alpha,\ s = \sin\alpha,\ c_1 = 1 - \cos\alpha$):

$$R = \begin{pmatrix} l_x^2 + c & l_x l_y c_1 - l_z s & l_x l_z c_1 + l_y s \\ l_x l_y c_1 + l_z s & l_y^2 c_1 + c & l_y l_z c_1 - l_x s \\ l_x l_z c_1 - l_y s & l_x l_z c_1 + l_x s & l_z^2 c_1 + c \end{pmatrix}. \quad (3.13)$$

Напомним, что l_x, l_y, l_z являются направляющими косинусами прямой, относительно которой выполняется поворот. Нетрудно убедиться, что если в качестве осей вращения взять оси координат, то мы в точности получим формулы (3.10).

Вопросы и упражнения

1. Дайте определение декартовой системы координат.
2. Что такое вектор?
3. Какие векторы считаются равными?
4. Какие векторы называются линейно независимыми?
5. Как выразить длину вектора, используя операцию скалярного произведения?
6. Как определить косинус угла между векторами, используя операцию скалярного произведения?
7. Докажите, что векторное произведение удовлетворяет соотношению

$$\alpha[\overrightarrow{r}_1 \times \overrightarrow{r}_2] = [\overrightarrow{r}_1 \times (\alpha\overrightarrow{r}_2)].$$

8. Как из произвольного вектора $\overrightarrow{r}$ получить единичный вектор, совпадающий с ним по направлению? (Эта операция называется **нормировкой вектора**).
9. Каково максимальное число линейно независимых векторов в пространстве?
10. Что такое орты?
11. Как построить параметрическое уравнение прямой, проходящей через две заданные точки плоскости или пространства?
12. Докажите, что если в формуле (3.7) заменить координаты (x_1, y_1, z_1) координатами любой другой точки плоскости, то уравнение будет описывать ту же самую плоскость. **Указание**: возьмите произвольную точку, удовлетворяющую уравнению (3.7), напишите новое уравнение плоскости и покажите, что любая точка второй плоскости принадлежит первой и наоборот.
13. В каких случаях луч с плоскостью не пересекаются?
14. В каких случаях луч пересекает сферу только в одной точке?

15. Исходя из определения умножения матрицы на вектор, докажите, что для любых двух векторов $\overrightarrow{r}_1, \overrightarrow{r}_2$ и любой матрицы A справедливо соотношение

$$A \cdot (\overrightarrow{r}_1 + \overrightarrow{r}_2) = A \cdot \overrightarrow{r}_1 + A \cdot \overrightarrow{r}_2.$$

16. Докажите, что для любого вектора $\overrightarrow{r}$, числа α и матрицы A справедливо соотношение

$$A \cdot (\alpha \overrightarrow{r}) = \alpha(A \cdot \overrightarrow{r}) = (\alpha A) \cdot \overrightarrow{r}.$$

17. При каком условии масштабирование сохраняет углы между отрезками?
18. Какую траекторию описывают точки объекта при повороте?
19. Вокруг чего осуществляется поворот на плоскости?
20. Вокруг чего осуществляется поворот в пространстве?
21. Какие шаги выполняются в алгоритме поворота относительно произвольной оси в пространстве?
22. Докажите, что если матрица A является матрицей поворота, то $A \cdot A^T = E$.

Лекция 4. Представление геометрической информации

Геометрические примитивы. Системы координат: мировая, объектная, наблюдателя и экранная. Однородные координаты. Задание геометрических преобразований в однородных координатах с помощью матриц

Геометрические примитивы

Под геометрическими примитивами понимают тот базовый набор геометрических фигур, который лежит в основе всех графических построений, причем эти фигуры должны образовывать "базис" в том смысле, что ни один из этих объектов нельзя построить через другие. Однако вопрос о том, что включать в набор геометрических примитивов, нельзя считать окончательно решенным в компьютерной графике. Например, количество примитивов можно свести к некоему минимуму, без которого нельзя обойтись, и этот минимум сводится к аппаратно реализованным графическим объектам. В этом случае базисный набор ограничивается отрезком, многоугольником и набором литер (символов).

Другая точка зрения состоит в том, что в набор примитивов необходимо включить гладкие кривые различного рода (окружности, эллипсы, кривые Безье), некоторые классы поверхностей и даже сплошные геометрические тела. В качестве трехмерных геометрических примитивов в таком случае предлагаются пространственные кривые, параллелепипеды, пирамиды, эллипсоиды. Но если такой расширенный набор примитивов связан с аппаратной реализацией, то возникает проблема перенесения программных приложений с одного компьютера на другой, поскольку такая аппаратная поддержка существует далеко не на всех графических станциях. Кроме того, при создании трехмерных геометрических примитивов программисты сталкиваются с проблемой их математического описания, а также разработки методов манипулирования такими объектами, поскольку те типы объектов, которые не попали в список базовых, надо уметь приближать с помощью этих примитивов.

Во многих случаях для аппроксимации сложных поверхностей используются многогранники, но форма граней может быть различной. Пространственный многоугольник с числом вершин больше трех не всегда бывает плоским, а в этом случае алгоритмы изображения многогранников могут привести к некорректному

результату. Поэтому программист должен сам позаботиться о том, чтобы многогранник был описан правильно. В этом случае оптимальным выходом из положения является использование треугольников, поскольку треугольник всегда является плоским. В современной графике это, пожалуй, самый распространенный подход.

Но существует и альтернативное направление, которое называется **конструктивной геометрией тел**. В системах, использующих этот подход, объекты строятся из объемных примитивов с использованием теоретико- множественных операций (объединение, пересечение).

Любая графическая библиотека определяет свой набор примитивов. Так, например, широко распространенная интерактивная система трехмерной графики OpenGL включает в список своих примитивов точки (вершины), отрезки, ломаные, многоугольники (среди которых особо выделяются треугольники и четырехугольники), полосы (группы треугольников или четырехугольников с общими вершинами) и шрифты. Кроме того, в нее входят и некоторые геометрические тела: сфера, цилиндр, конус и др.

Понятно, что для изображения таких примитивов должны быть разработаны эффективные и надежные алгоритмы, поскольку они являются конструктивными элементами. Исторически сложилось так, что первые дисплеи были векторными, поэтому базовым примитивом был отрезок. Но, как уже было отмечено в первой главе нашего курса, самая первая интерактивная программа Sketchpad А.Сазерленда в качестве одного из примитивов имела прямоугольник, после чего этот объект уже традиционно входил в различные графические библиотеки.

Здесь мы рассмотрим такие примитивы, как **вершина**, **отрезок**, **воксель** и модели, строящиеся на их основе, а также **функциональные модели**.

Полигональные модели

Для этих пространственных моделей используются в качестве примитивов вершины (точки в пространстве), отрезки прямых (векторы), из которых строятся **полилинии**, **полигоны** и **полигональные поверхности**. Главным элементом описания является вершина, все остальные являются производными. В трехмерной декартовой системе координаты вершины определяются своими координатами (x,y,z), линия задается двумя вершинами, полилиния представляет собой незамкнутую ломаную линию, полигон - замкнутую ломаную линию. Полигон моделирует плоский объект и может описывать плоскую грань объемного объекта. Несколько граней составляют этот объект в виде полигональной поверхности - многогранник или незамкнутую поверхность ("полигональная сетка").

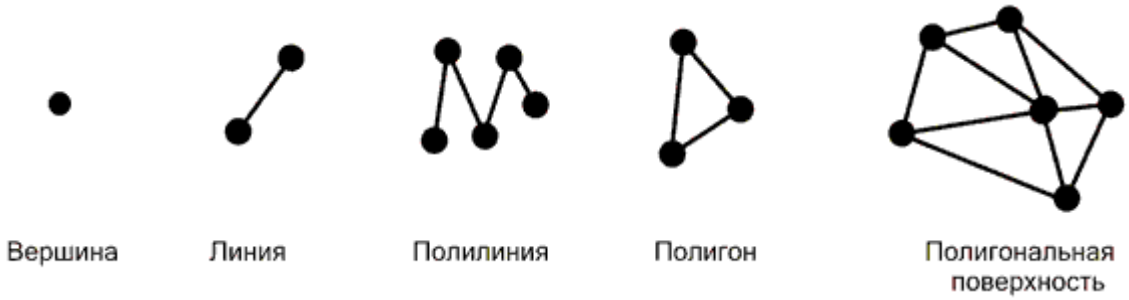


Рис. 4.1. Полигональные модели

В современной компьютерной графике векторно-полигональная модель является наиболее распространенной. Она применяется в системах автоматизированного проектирования, компьютерных играх, тренажерах, ГИС, САПР и т. д. Достоинства этой модели заключаются в следующем:

- Удобство масштабирования объектов.
- Небольшой объем данных для описания простых поверхностей.
- Аппаратная поддержка многих операций.

К числу недостатков полигональных моделей можно отнести то, что алгоритмы визуализации выполнения топологических операций (например, построение сечений) довольно сложны. Кроме того, аппроксимация плоскими гранями приводит к значительной погрешности, особенно при моделировании поверхностей сложной формы.

Воксельные модели

Воксельная модель - это представление объектов в виде трехмерного массива объемных (кубических) элементов. Само название "воксель" составлено из двух слов: volume element. Так же как и пиксель, воксель имеет свои атрибуты (цвет, прозрачность и т. п.). Полная прозрачность вокселя означает пустоту в соответствующей точке объема. Чем больше вокселей в определенном объеме и меньше их размер, тем точнее моделируются трехмерные объекты.

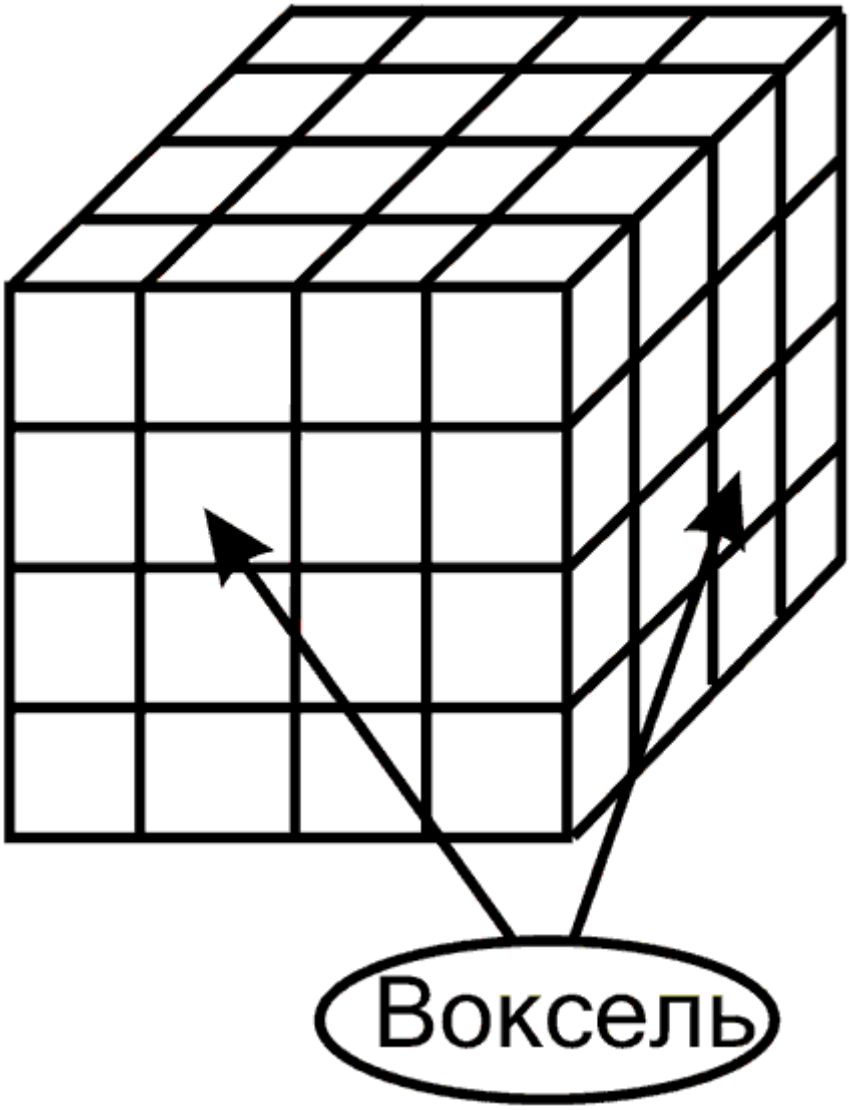


Рис. 4.2. Воксельная модель

Положительными чертами воксельной модели являются:

- Возможность представлять внутренность объекта, а не только внешний слой; простая процедура отображения объемных сцен.
- Простое выполнение топологических операций; например, чтобы показать сечение пространственного тела, достаточно воксели сделать прозрачными.

К ее недостаткам относятся:

- Большое количество информации, необходимое для представления объемных данных.
- Значительные затраты памяти, ограничивающие разрешающую способность, точность моделирования.
- Проблемы при увеличении или уменьшении изображения; например, с увеличением ухудшается разрешающая способность изображения.

Поверхности свободных форм (функциональные модели)

Характерной особенностью предлагаемого способа задания поверхностей является то, что основным примитивом здесь является поверхность второго порядка - **квадрик**. Он определяется с помощью вещественной непрерывной функции трех переменных (x, y, z) в виде неравенства

$$F(x, y, z) \geq 0.$$

Таким образом, квадрик есть замкнутое подмножество евклидова пространства, все точки которого удовлетворяют указанному неравенству. Уравнение $F(x, y, z) = 0$ описывает **границу** этого множества. Множество точек, удовлетворяющих неравенству $F(x, y, z) < 0,$ образует **внешнюю область** квадрика.

Свободная форма - это произвольная поверхность, обладающая свойствами гладкости, непрерывности и неразрывности. На базе квадриков строятся свободные формы, которые описывают функциональные модели. Свободная форма, построенная на этих принципах, имеет ряд достоинств, к которым, в первую очередь, надо отнести следующие:

- Легкая процедура расчета координат каждой точки.
- Небольшой объем информации для описания достаточно сложных форм.
- Возможность строить поверхности на основе скалярных данных без предварительной триангуляции.

Этот подход будет более подробно изложен в следующих главах.

В нашем курсе предполагается рассмотреть растровые алгоритмы для изображения таких геометрических примитивов, как отрезки, многоугольники, окружности и эллипсы. Но сначала мы займемся тем геометрическим аппаратом, который позволит адекватно описывать объекты в пространстве, работать с ними и формировать изображение.

Системы координат: мировая, объектная, наблюдателя и экранная

Одной из распространенных задач компьютерной графики является изображение двумерных графиков в некоторой системе координат. Эти графики предназначены для отображения зависимости между переменными, заданной с помощью функции. Например, во второй главе настоящего курса приведен ряд графиков,

характеризующих восприятие света глазом человека. Чтобы получить такой график, прикладная программа должна описать различные выходные примитивы (точки, линии, цепочки символов), указав их местоположение и размеры в прямоугольной системе координат. Единицы измерения, в которых задаются эти объекты, зависят от их природы: изменение температуры, например, можно отображать в градусах за час, перемещение тела в пространстве - в километрах в секунду, и т. д. Эти прикладные (или ориентированные на пользователя) координаты позволяют задавать объекты в двумерном или трехмерном мире пользователя, и их принято называть **мировыми координатами**.

Изображение трехмерных объектов сопряжено с целым рядом задач. Прежде всего надо помнить, что изображение является плоским, поэтому надо добиться адекватной передачи визуальных свойств предметов, дать достаточно наглядное представление о глубине. В дальнейшем группы трехмерных объектов, предназначенных для изображения, будем называть **пространственной сценой**, а ее двумерное изображение - **образом**.

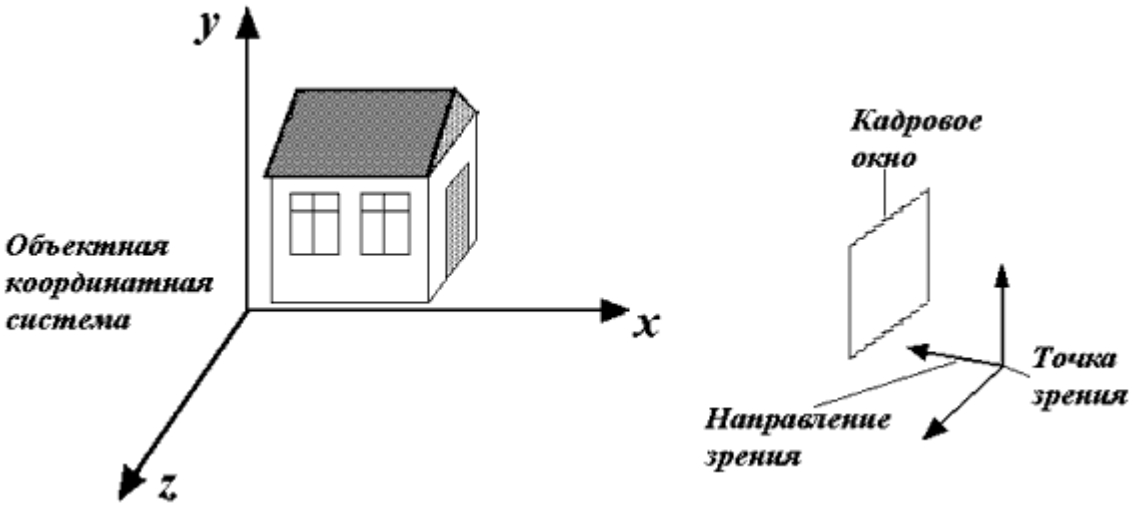


Рис. 4.3. Объектная система координат и система координат наблюдателя

Как и в случае с двумерными объектами, первым шагом построения является ввод информации об объектах. Сцена занимает какое-то определенное место в пространстве, а ее описание привязывается к трехмерной декартовой системе координат, связанной с нею, - **объектной координатной системе**. Координаты объектов, составляющих сцену, определяются на основе их реальных размеров и взаимного расположения. В зависимости от точки, из которой рассматривается сцена, можно получить множество различных ее образов. Если построено достаточно много таких образов, то по ним можно восстановить объемную структуру предмета. Выбор точки и направления зрения тоже можно описать математически, введя декартову **систему координат наблюдателя**, начало которой находится в точке обзора, а одна из осей совпадает с направлением зрения (рис. 4.3). Переход от объектных координат к координатам наблюдателя математически реализуется так, как это было описано в третьей главе. На этом этапе преобразований сохраняются реальные размеры объектов.

Видимый образ формируется на некоторой плоскости, которую в дальнейшем будем называть **картинной плоскостью**. Способы преобразования трехмерного объекта в двумерный образ (**проекции**) могут быть различными. Так или иначе, но полученный образ также должен быть описан в некоторой двумерной системе координат. В зависимости от способа его получения реальные размеры образа также могут быть различны. Различные виды проецирования будут подробно рассмотрены в последующих главах.

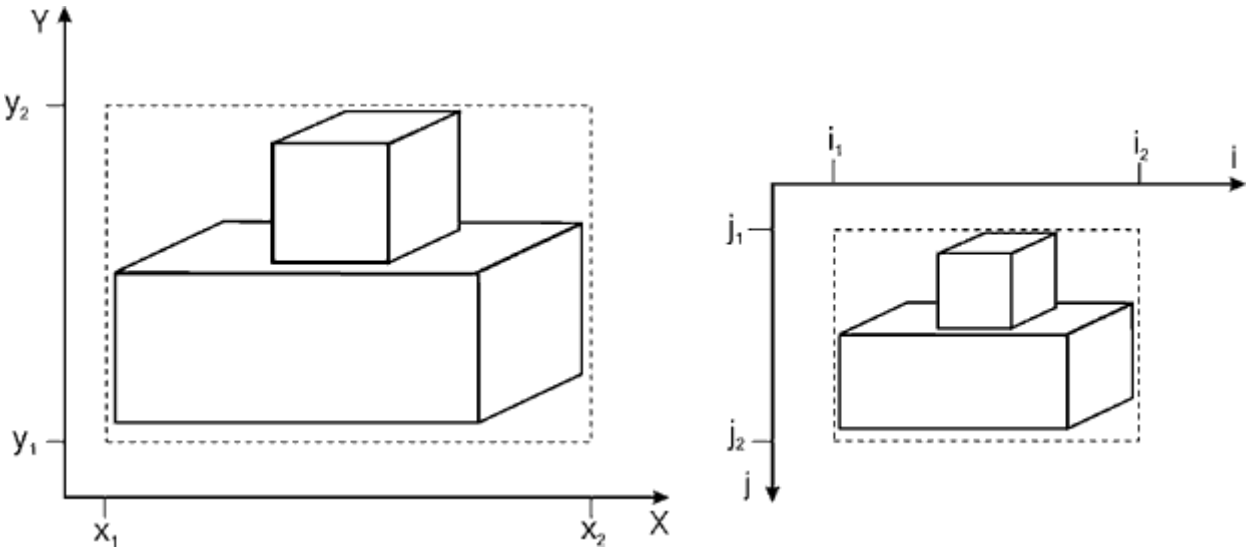


Рис. 4.4. Картинная плоскость и экран

Поскольку нашей конечной целью является получение изображения на экране, то перенесение образа сопровождается изменением масштаба в соответствии с размерами экрана. Обычно началом координат в системе координат образа считается левый нижний угол листа с изображением. На экране дисплея начало координат традиционно находится в левом верхнем углу. Отображение рисунка с картинной плоскости на экран должно производиться с минимальным искажением пропорций, что само по себе вносит ограничение на область экрана, занимаемую рисунком. Изменение масштаба должно осуществляться с сохранением пропорций области (рис. 4.4).

Объекты в системе координат картинной плоскости задаются в каких- либо единицах измерения, причем масштаб одинаков по обеим осям координат. На экране единицей измерения является пиксель, который следует рассматривать как прямоугольный, поэтому масштабы по горизонтальной и вертикальной осям могут быть различны, что необходимо учитывать при задании коэффициентов масштабирования.

Рассмотрим ситуацию, когда изображение занимает на картинной плоскости прямоугольную область $x_1 \le x_2,\ y_1 \le y \le y_2$. При отображении рисунка на экран каждая точка исходного прямоугольника с координатами (x, y) перейдет в некоторую точку с целочисленными координатами (i, j). Введем обозначения:

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1, \quad \Delta i = i_2 - i_1, \quad \Delta j = j_2 - j_1,$$

$$S_x = \frac{\Delta i}{\Delta x}, \quad S_y = \frac{\Delta j}{\Delta y}$$

(предполагается, что изображение займет на экране прямоугольник $i_1 \le i \le i_2,\ j_1 \le j_2$). Определим преобразование координат образа (x, y) в экранные координаты (i, j) формулами

$$i = i_1 + S_x(x - x_1), \quad j = j_2 - S_y(y - y_1).$$

Ясно, что при таком отображении прямоугольная область образа в точности перейдет в соответствующий экранный прямоугольник, как показано на рисунке. Теперь надо определить сам экранный прямоугольник так, чтобы его пропорции соответствовали прямоугольнику образа, т.е.

$$\frac{\Delta x}{\Delta y} = \frac{\Delta i \cdot l_x}{\Delta j \cdot l_y} \equiv \kappa \frac{\Delta i}{\Delta j},$$

где l_x, l_y - горизонтальный и вертикальный размер одного пикселя. Эти параметры легко установить, зная размеры экрана и разрешение. Отсюда получаем
$\Delta j = \kappa \cdot S_x \cdot \Delta y, \quad S_y = \kappa \cdot S_x$
Теперь достаточно задать на экране начало отсчета и горизонтальный размер окна, а остальные параметры легко вычисляются.

Однородные координаты. Задание геометрических преобразований в однородных координатах с помощью матриц

В предыдущей главе описывались геометрические преобразования на плоскости и в пространстве, а также было показано, как можно использовать аппарат матриц для таких задач. Для преобразований на плоскости применялись двумерные векторы и матрицы размерностью 2×2. В пространстве, соответственно, с этой же целью использовались трехмерные векторы и матрицы 3×3. Но такой подход не позволяет задавать с помощью матриц преобразования переноса и проекции. В связи с этим в проективной геометрии был разработан аппарат, позволяющий унифицировать все геометрические преобразования путем введения так называемых **однородных координат**.

Для пояснения такого подхода сначала рассмотрим случай двумерного пространства. Каждая точка плоскости с координатами (x, y) может одновременно рассматриваться как точка трехмерного пространства с координатами $(x, y, 1)$, т.е. как точка, лежащая на плоскости $z = 1$. С другой стороны, каждой точке трехмерного пространства (x, y, z) при условии $z \neq 0$ соответствует единственная точка этой же плоскости $\left(\frac{x}{z}, \frac{y}{z}, 1\right)$. При этом получается, что каждой точке плоскости $(x, y, 1)$ соответствует прямая, проходящая через начало координат, т. е. устанавливается взаимно однозначное соответствие между точками плоскости и множествами $(tx, ty, t),\ -\infty < t < \infty,\ t \neq 0$.

Если теперь рассматривать точку плоскости как принадлежащую **трехмерному** пространству, то ее **двумерные** преобразования можно будет описывать с помощью матриц 3×3, причем можно будет задавать таким способом не только повороты и масштабирование, но и сдвиги и проекции (как ортографические, так и центральные).

Поворот на угол α относительно начала координат можно осуществить с помощью новой матрицы поворота:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Операция масштабирования может быть записана в виде

$$\begin{pmatrix} ax \\ by \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Перенос на вектор (x_0, y_0) также можно задать с помощью матрицы:

$$\begin{pmatrix} x + x_0 \\ y + y_0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Проекции точки на оси координат определяются с помощью матриц проекции:

$$P_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad P_y = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Перейдем теперь к трехмерному пространству. Каждой точке (x, y, z)будем ставить в соответствие точку четырехмерного пространства $(x, y, z, 1)$, а для выполнения основных преобразований будем использовать матрицы размерностью 4×4. Строятся они совершенно аналогично тому, как это делалось в двумерном случае. Матрица сдвига на вектор (x_0, y_0, z_0)имеет вид

$$P = \begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

матрица масштабирования тоже очевидным образом строится из трехмерной матрицы:

$$S = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Проекции точек на координатные плоскости осуществляются с помощью матриц (более подробно проекции и их виды будут рассмотрены позднее):

$$P_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad P_{yz} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad P_{zx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Умножение этих матриц на вектор приводит к тому, что обнуляется одна из координат, и в результате получаем проекцию точки на соответствующую плоскость.

Матрица поворота относительно оси OX на угол α выглядит следующим образом:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Отсюда легко понять, как строятся матрицы поворота относительно других координатных осей, а также матрица поворота относительно произвольной оси. Просто берем матрицы, построенные в третьей главе, и расширяем их путем добавления уже известных единичных вектора-строки и вектора- столбца:

$$\begin{pmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Путем объединения приведенных элементарных преобразований можно построить и более сложные. В третьей главе мы использовали произведение простых матриц вращения для построения матрицы поворота относительно произвольной оси. Приведем один пример.

Пусть в пространстве заданы два отрезка - $\overline{AB}$ и $\overline{CD}$. Будем строить матрицу преобразования, переводящую первый отрезок во второй. Это преобразование разложим на следующие элементарные действия.

1. Сдвиг, перемещающий точку $A = (A_x, A_y, A_z)$ в точку $C = (C_x, C_y, C_z)$.
2. Сдвиг начала координат в эту же точку.
3. Если отрезки неколлинеарны:
    - строится вектор нормали к плоскости, в которой лежат отрезки (для этого можно использовать векторное произведение исходных векторов);
    - поворот относительно вектора нормали, совмещающий два отрезка по направлению (угол поворота можно определить с помощью скалярного произведения исходных векторов).
4. Масштабирование с целью выравнивания длины отрезков.
5. Возвращение начала координат в исходную точку.

Каждое из этих преобразований реализуется с помощью матрицы, а полное преобразование можно выполнить, используя произведение матриц.

Использование матриц очень удобно для выполнения преобразований в пространстве, хотя в некоторых случаях это приводит к избыточному числу выполняемых операций. Например, поворот одной точки в пространстве относительно координатной оси OZ с помощью матриц в однородных координатах требует 16 операций умножения и 12 операций сложения. В то же время он легко может быть выполнен с помощью формул преобразования

$$x' = x\cos\alpha - y\sin\alpha$$
$$y' = x\sin\alpha + y\cos\alpha$$

т.е. с помощью всего лишь четырех умножений и одного сложения и одного вычитания. Операции сдвига также гораздо более экономично выполнять без использования матриц. Но когда речь идет о суперпозиции многих преобразований (как, например, в случае поворота относительно произвольной оси), то целесообразно применять соответствующую матрицу поворота. Эффективность матричного подхода очень сильно возрастает, если матричные операции реализованы аппаратно. Вопрос о том, в каких случаях использовать матрицы, а в каких нет, во многом зависит от возможностей вычислительной техники, уровня сложности задачи и требований к временным характеристикам процесса визуализации.

Вопросы и упражнения

1. Какие геометрические объекты считаются примитивами?
2. Какие требования предъявляются к набору геометрических примитивов?
3. В какой программе впервые в качестве геометрического примитива использовался прямоугольник?
4. Что такое объектная система координат?
5. Что такое система координат наблюдателя?
6. Соответствуют ли размеры объектов в системе координат наблюдателя их реальным размерам?
7. Что такое картинная плоскость?
8. Как называется операция перехода от трехмерной системы координат к двумерной?
9. Что происходит при перенесении изображения с картинной плоскости на экран?

10. Чем отличаются однородные координаты точки от обычных декартовых координат?
11. С какой целью вводятся однородные координаты?
12. Сколько элементарных действий требуется для совмещения двух отрезков в пространстве?
13. Всегда ли использование матриц для выполнения преобразований в пространстве эффективней, чем другие способы их реализации?

Лекция 5. Отсечение (клиппирование) геометрических примитивов

Алгоритм деления отрезка пополам. Коды Сазерланда — Коэна. Клиппирование многоугольников. Штрихование многоугольной области. Переход к трехмерному клиппированию пирамидой видимости

В компьютерной графике часто приходится решать задачу выделения некоторой области изображаемой сцены, причем задача эта может решаться как в применении к плоской области (если сцена уже спроецирована на картинную плоскость), так и к трехмерной. Алгоритмы отсечения применяются для удаления невидимых поверхностей и линий, для построения теней, при формировании текстур. Отсекаемая область может быть как правильной формы (прямоугольник или параллелепипед со сторонами, параллельными осям координат или координатным плоскостям), так и неправильной (произвольный многоугольник или многогранник). Для того чтобы эти алгоритмы можно было использовать в задачах изображения динамичных сцен, они должны быть эффективными в отношении времени вычислений. Мы рассмотрим несколько наиболее часто применяемых алгоритмов.

Алгоритм Сазерленда-Коэна отсечения прямоугольной областью

Рассмотрим плоскую сцену, состоящую из отрезков различной длины и направлений, в которой надо выделить часть, находящуюся внутри прямоугольника. Прямоугольник задан списком ребер: **<top>, <bottom>, <left>, <right>**, отрезки также задаются координатами концевых точек. Область, отсекаемая окном (с учетом его границы), состоит из точек (x, y), удовлетворяющих соотношениям

$$x_l \le x \le x_r, \quad y_b \le y \le y_t. \quad (5.1)$$

Пусть концы отрезка заданы точками (x_1, y_1) и (x_2, y_2). Первый шаг алгоритма нацелен на то, чтобы выявить полностью видимые и полностью невидимые отрезки. Отрезок целиком принадлежит выделяемой (клиппируемой) области, если оба его конца удовлетворяют условиям (5.1).

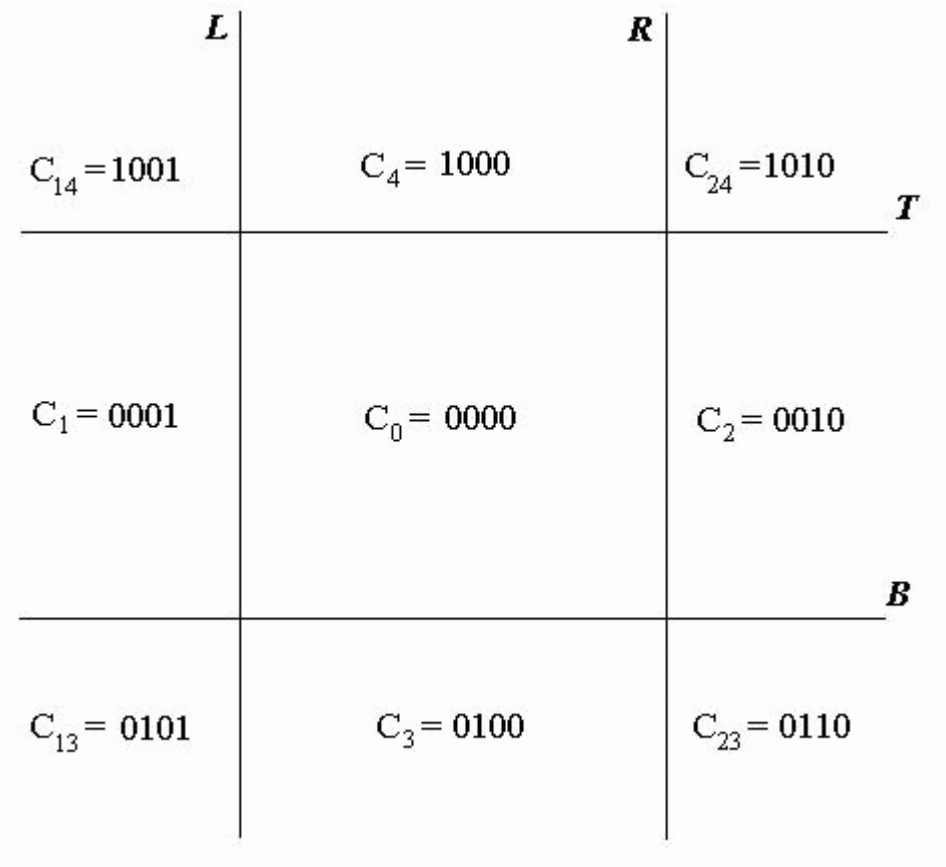


Рис. 5.1. Коды Сазерленда-Коэна для областей

Отрезок полностью невидим, если оба его конца лежат

1. справа от ребра $r(x_1 > x_r, \quad x_2 > x_r)$;
2. слева от ребра $l(x_1 < x_l, \quad x_2 < x_l)$;
3. снизу от ребра $b(y_1 < y_b, \quad y_2 < y_b)$;
4. сверху от ребра $t(y_1 > y_t, \quad y_2 > y_t)$.

Во всех остальных случаях отрезок может (но не обязан) пересекать прямоугольное окно.

Для выполнения анализа полной видимости или невидимости отрезка А.Сазерленд и Д.Коэн предложили следующий алгоритм. Прямые, которым принадлежат ребра прямоугольника, разбивают плоскость на девять областей, каждой из которых присваивается четырехразрядный код. Каждый бит этого кода "отвечает" за одну из прямых: 1-й (старший) бит - за прямую t, 2-й - за прямую b, 3-й - за r, 4-й - за l. Если в коде области какой-либо бит установлен в 1, то это означает, что она отделена от окна соответствующей прямой. Схема идентификации областей приведена на рис. 5.1.

Концевым точкам отрезков сцены теперь можно присвоить коды в зависимости от расположения точек. Ясно, что если коды обоих концов отрезка равны нулю, то отрезок полностью лежит внутри окна. Для дальнейшего анализа воспользуемся операцией логического умножения кодов (поразрядное логическое "И"). Тогда таблица истинности для кодов, согласно схеме на рис. 5.1, будет выглядеть следующим образом:

Таблица 5.1. Значения истинности для логического умножения кодов

областей								
	C_1	C_2	C_3	C_4	C_{13}	C_{14}	C_{23}	C_{24}
C_1	T	F	F	F	T	T	F	F
C_2	F	T	F	F	F	F	T	T
C_3	F	F	T	F	T	F	T	F
C_4	F	F	F	T	F	T	F	T
C_{13}	T	F	T	F	T	T	T	F
C_{14}	T	F	F	T	T	T	F	T
C_{23}	F	T	T	F	T	F	T	T
C_{24}	F	T	F	T	F	T	T	T

Из сопоставления таблицы с рисунком видно, что если произведение кодов концов отрезка принимает значение **<True>**, то отрезок целиком лежит по одну сторону какой-то из прямых, причем внешнюю сторону по отношению к окну, следовательно, он полностью невидим. Во всех остальных случаях отрезок может частично лежать внутри окна, поэтому для определения их видимой части надо решать задачу о пересечении отрезков с ребрами окна. При этом желательно по возможности сократить число перебираемых пар "отрезок-ребро".

В самом общем случае существуют две точки пересечения отрезка с ребрами, и эти две точки принимаются за новые концевые точки изображаемого отрезка. Но сначала можно выделить некоторые более простые частные случаи, поиск пересечений для которых является более эффективным. Прежде всего это горизонтальные и вертикальные отрезки, для которых поиск точки пересечения тривиален. Далее, если код одного из концов отрезка равен нулю, то существует только одно пересечение этого отрезка с ребром (или с двумя ребрами, если отрезок проходит через угловую точку окна). На рис. 5.2 приведена общая блок-схема алгоритма отсечения для одного произвольно направленного отрезка.

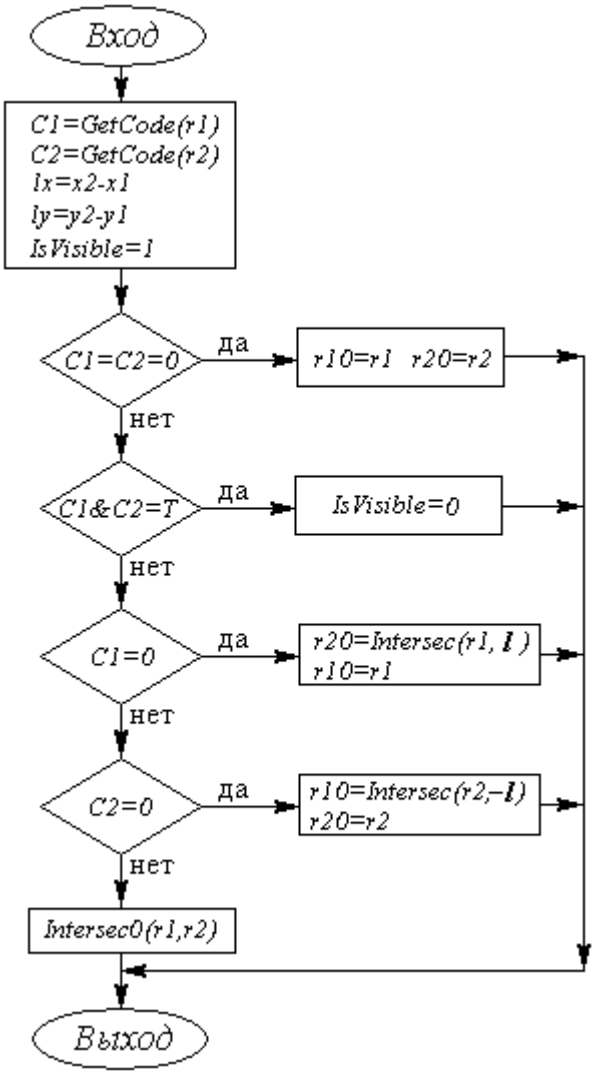


Рис. 5.2. Блок-схема алгоритма Сазерленда-Коэна

В блок-схеме используются следующие соглашения и обозначения:

- входными данными являются точки $\overrightarrow{r}_1 = (x_1, y_1),\ \overrightarrow{r}_2 = (x_2, y_2)$, массив координат окна $w = \{L, R, B, T\};\ \overrightarrow{l} = (x_2 - x_1, y_2 - y_1) \equiv (l_x, l_y)$;
- на выходе получаем новые концевые точки $\overrightarrow{r}_{10} = (x_{10}, y_{10}),\ \overrightarrow{r}_{20} = (x_{20}, y_{20})$, а также значение переменной *IsVisible* (0 - отрезок видимый);
- используются следующие вспомогательные функции:

GetCode(r) - определение кода точки;

Intersec0(r1,l) - поиск пересечения отрезка со сторонами окна при условии, что обе точки лежат вне окна; если пересечения нет, устанавливает переменную *IsVisible* в 0;

Intersec(r1,l) - поиск пересечения отрезка со сторонами окна при условии, что точка *r1* лежит в окне;

C1, C2 - коды точек *r1, r2*.

В приведенном алгоритме теперь остается только детализовать функции *Intersec0* и *Intersec*, эффективность работы которых является ключевым моментом. Рассмотрим один из методов поиска пересечений, который использует параметрическое уравнение прямой, проходящей через точки $\overrightarrow{r}_1 = (x_1, y_1),\ \overrightarrow{r}_2 = (x_2, y_2)$:

$$\overrightarrow{r} = \overrightarrow{r}_1 + s(\overrightarrow{r}_2 - \overrightarrow{r_1}), \quad s \in [-\infty, +\infty],$$

или в координатном виде

$$x = x_1 + sl_x, \quad y = y_1 + sl_y, \quad l_x = x_2 - x_1, \quad l_y = y_2 - y_1.$$

Попробуем определить точку пересечения отрезка с верхней границей окна. Поскольку эта граница описывается соотношениями

$$L \le x \le R, \quad y = T,$$

то условие пересечения с ней клиппируемого отрезка выглядит следующим образом:

$$s_T = \frac{T - y_1}{l_y}, \quad L \le x_1 + s_T l_x \le R.$$

Аналогично выглядят формулы и для остальных границ окна. Если точка $\overrightarrow{r}_1$ расположена внутри окна, то, в зависимости от знака l_y, следует искать пересечение либо с верхней, либо с нижней границей окна. При отсутствии таковых отыскиваются пересечения с левой или правой стороной окна. Но прежде чем перебирать эти варианты, необходимо исключить случаи горизонтальных $(l_y = 0)$ и вертикальных $(l_x = 0)$ направлений отрезка. В первом случае точками пересечения с правой или левой границей (в зависимости от знака l_x) могут быть (L, y_1) или (R, y_1), а во втором - (x_1, B) или (x_1, T).

Этот алгоритм реализован в виде функции *Intersec*, блок-схема которой приведена на рис. 5.3.

Теперь рассмотрим случай, когда ни один из концов отрезка не лежит внутри окна. Здесь мы можем найти первую точку пересечения, заменить ею один из концов отрезка и использовать предыдущий алгоритм нахождения второй точки. Заметим, что в данном случае первый шаг не всегда кончается успешно, поскольку предварительный анализ не позволяет полностью исключить все невидимые отрезки. Как и в предыдущем алгоритме, сначала выполняется проверка на горизонтальное и вертикальное направления отрезка. Поскольку часть отрезков мы уже исключили из рассмотрения благодаря предварительному анализу, то для этих простых ситуаций остаются только две возможности, приведенные на рис. 5.4. Здесь точки пересечения определяются совершенно очевидным образом, причем сразу обе.

Затем последовательно рассматриваются четыре случая расположения точки $\overrightarrow{r}_1$ относительно прямых, ограничивающих окно. Если в каком-то из вариантов будет найдена точка пересечения, то анализ прекращается, точка $\overrightarrow{r}_1$ заменяется новой точкой и вызывается функция *Intersec*. В случае же, когда все четыре варианта не дали положительного результата, переменной *IsVisible* присваивается значение 0. Алгоритм реализуется функцией *Intersec0* (блок-схема на рис. 5.5).

Предложенная реализация алгоритма отсечения не является единственной в своем роде. Существуют и другие подходы, в частности использование метода деления отрезка пополам для поиска точек пересечения и выявления видимой части отрезка. Такой вариант алгоритма был предложен Сазерлендом и Спрулом для аппаратной реализации, но он может быть реализован и на программном уровне, хотя при этом эффективность его будет ниже, чем у предыдущего. В нем нет прямого вычисления

координат новой точки по явным алгебраическим соотношениям. Поиск осуществляется итерационным методом, в котором на каждом шаге для отрезка, "подозреваемого" в частичной видимости, находится его средняя точка, определяется ее код, затем из двух отрезков оставляются либо оба (если они оба не окажутся полностью невидимыми), либо только один, после чего операции продолжаются с новыми отрезками. Ситуация, когда дальнейшему дроблению подвергаются сразу два вновь полученных отрезка, может возникнуть только на первом итерационном шаге в тех случаях, когда исходный отрезок имеет две точки пересечения с границами окна. На последующих итерационных шагах количество анализируемых отрезков уже не будет увеличиваться. Процесс продолжается до тех пор, пока длина очередного отрезка не станет меньше наперед заданной точности. После этого найденная точка проверяется на предмет пересечения со стороной окна. На рис. 5.6 приведены три варианта отрезков, предварительный анализ которых не классифицирует их как полностью невидимые, и показан первый итерационный шаг. Отрезок а после первого деления дает два частично видимых отрезка, после чего ищутся две точки пересечения. В остальных случаях остается лишь один из двух отрезков, причем в случае отрезка с точка пересечения со стороной окна отсутствует, т.е. обнаруживается полная невидимость отрезка. По сути дела общий алгоритм, показанный на рис. 5.3, сохраняется, изменяется лишь метод поиска точки пересечения.

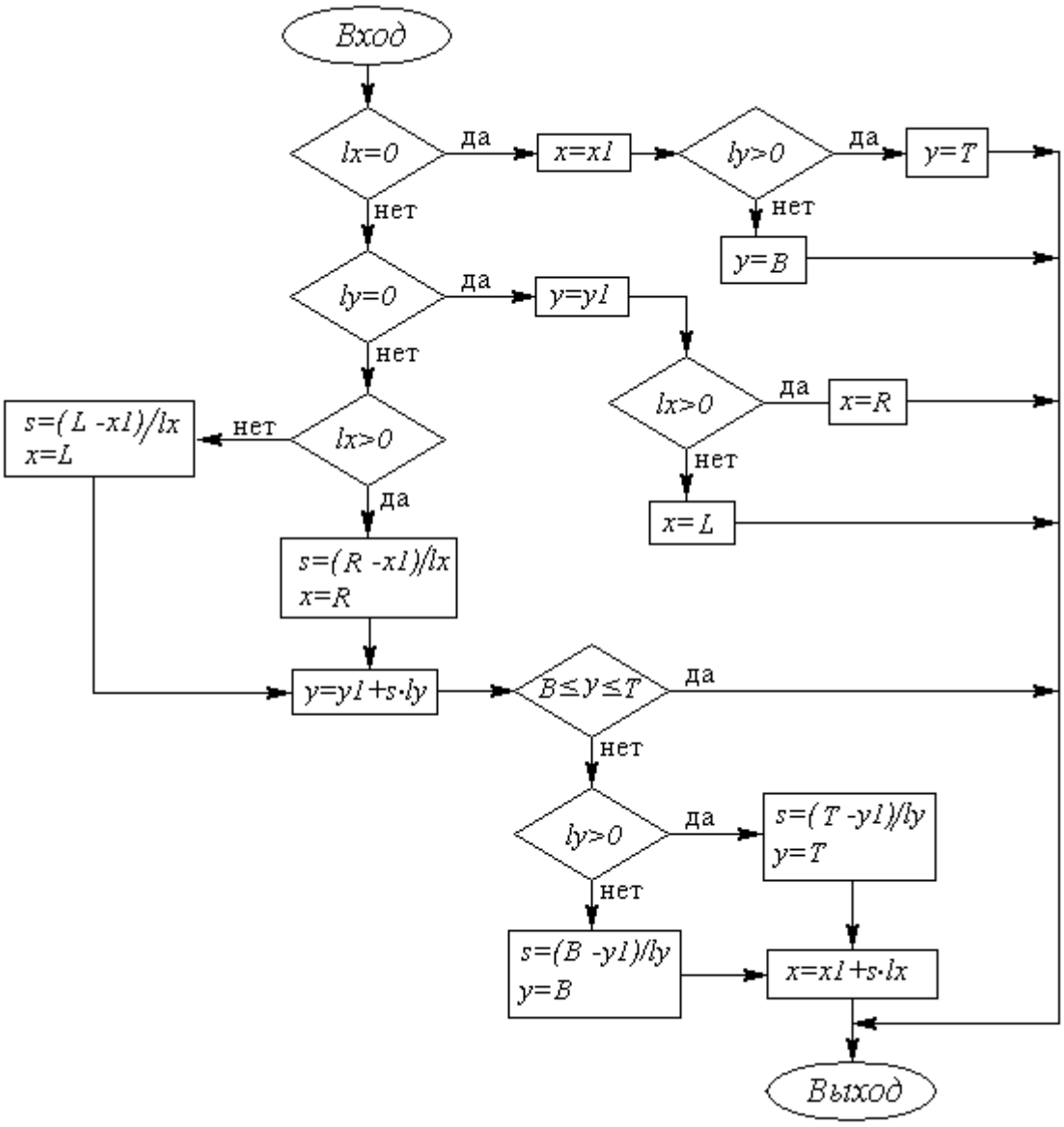


Рис. 5.3. Блок-схема функции Intersec

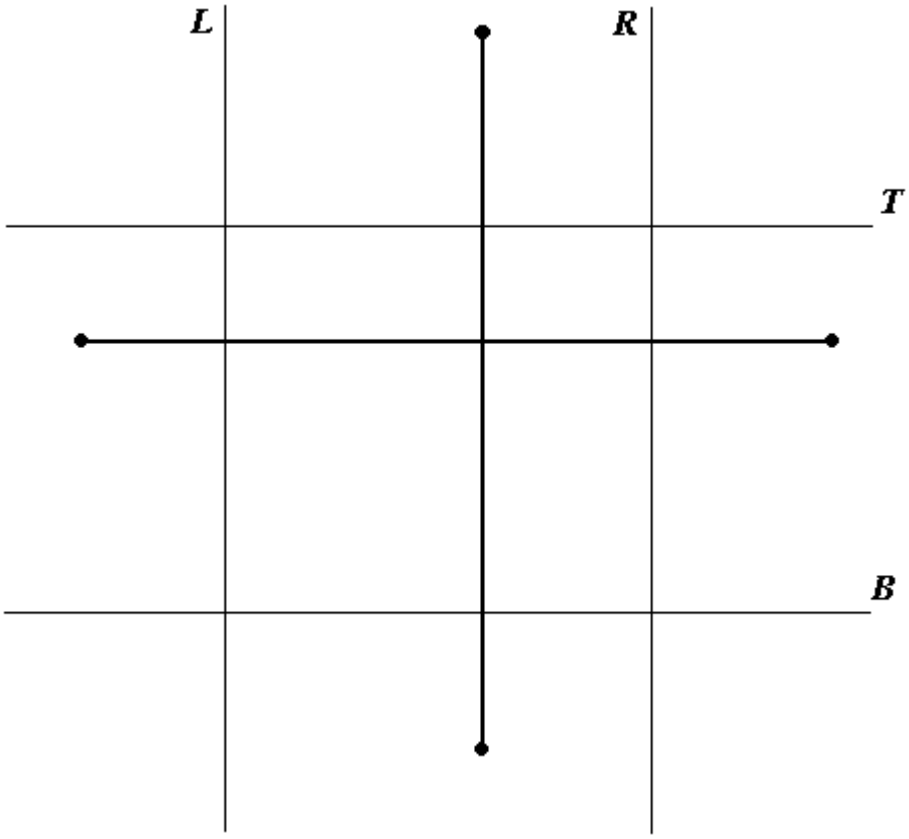
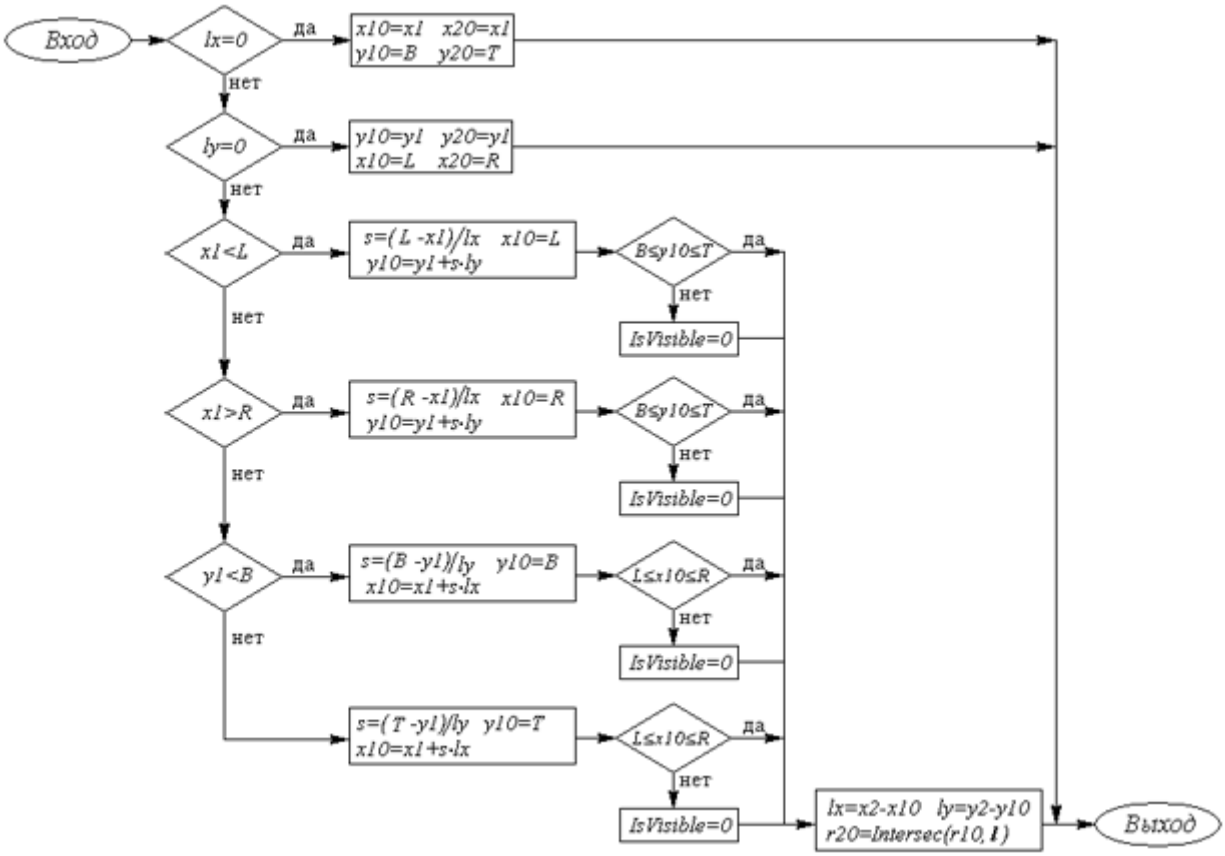


Рис. 5.4. Отрезки параллельны сторонам окна



увеличить изображение

Рис. 5.5. Блок-схема функции Intrsec0

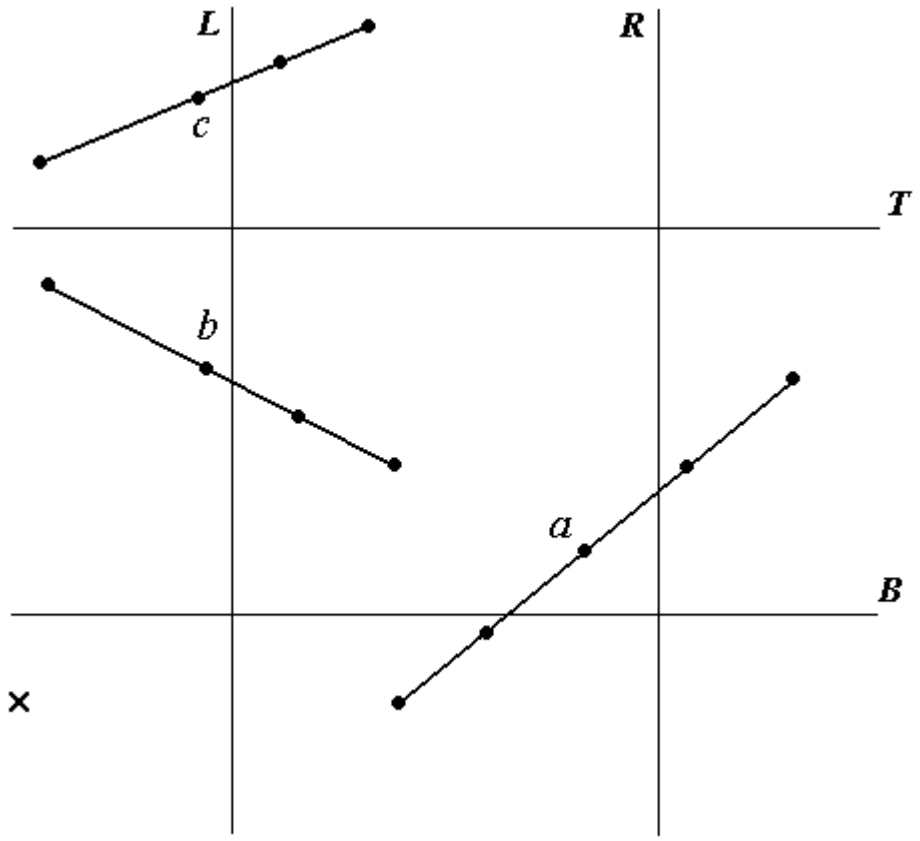


Рис. 5.6. Произвольное расположение отрезков

Отсечение выпуклым многоугольником

Во многих задачах компьютерной графики часто приходится иметь дело с отсечением не только простым прямоугольным окном, но и окном достаточно произвольной геометрии. В частности, такие задачи могут возникнуть при использовании перспективных проекций трехмерных сцен, но не только в этих случаях. Поэтому актуальной является задача отсечения выпуклым многоугольником. Ясно, что простой анализ с помощью кодов Сазерленда-Коэна в такой ситуации неприменим. Здесь нужен надежный и достаточно эффективный алгоритм нахождения точки пересечения двух произвольно ориентированных отрезков, а также алгоритм определения местоположения точки относительно многоугольника (внутри, снаружи или на границе).

Рассмотрим задачу о пересечении отрезка с концами $\overrightarrow{r}_1 = (x_1, y_1),\ \overrightarrow{r}_2 = (x_2, y_2)$ с выпуклым многоугольником, заданным списком ребер. Ребро может быть задано в виде пары точек из множества вершин многоугольника $R = (\overrightarrow{p}, \overrightarrow{q}),\ \overrightarrow{p} = (p_x, p_y),\ \overrightarrow{q} = (q_x, q_y)$ (рис. 5.7). То обстоятельство, что многоугольник выпуклый, является очень существенным: это позволяет использовать достаточно простой алгоритм, использующий внутренние нормали к его сторонам. Под внутренней нормалью понимается вектор, перпендикулярный стороне и направленный внутрь многоугольника. Как и в предыдущем алгоритме, воспользуемся параметрическим уравнением прямой, проходящей через концы отрезка: $\overrightarrow{r} = \overrightarrow{r}_1 + s(\overrightarrow{r}_2 - \overrightarrow{r}_1)$. Если при некотором значении параметра s_0 эта прямая пересекается с прямой, проходящей через точки $\overrightarrow{p}, \overrightarrow{q}$, то вектор, соединяющий произвольную точку ребра с точкой $\overrightarrow{r}_0 = \overrightarrow{r}_1 + s_0(\overrightarrow{r}_2 - \overrightarrow{r}_1)$, будет перпендикулярен вектору нормали. Следовательно, скалярное произведение векторов

$\overrightarrow{n}$ и $\overrightarrow{p} - \overrightarrow{r}_0$ будет равно нулю. Отсюда путем несложных выкладок получаем $s_0 = \frac{((\overrightarrow{p} - \overrightarrow{r}_1) \cdot \overrightarrow{n})}{((\overrightarrow{r}_2 - \overrightarrow{r}_1) \cdot \overrightarrow{n})}$..

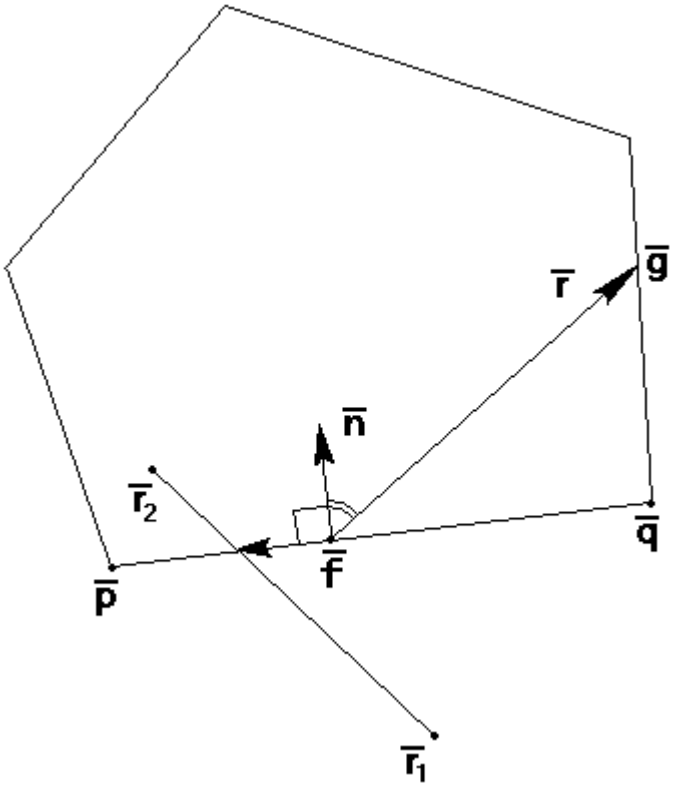


Рис. 5.7. Пересечение отрезка многоугольником

Конечно, использование этой формулы предполагает, что $d \equiv ((\overrightarrow{r}_2 - \overrightarrow{r}_1) \cdot \overrightarrow{n}) \neq 0$, т.е. что отрезок не параллелен стороне многоугольника, но этот случай рассматривается отдельно. Найденная точка принадлежит отрезку при условии $0 \leq s_0 \leq 1$. Условие принадлежности этой точки ребру многоугольника также можно выразить через скалярное произведение, так как векторы $\overrightarrow{p} - \overrightarrow{r}_0$ и $\overrightarrow{r}_0 - \overrightarrow{q}$ в этом случае должны быть одинаково направленными, т.е. $((\overrightarrow{p} - \overrightarrow{r}_0) \cdot (\overrightarrow{r}_0 - \overrightarrow{q})) \geq 0$.

Для каждого отрезка возможны три случая взаимного расположения с многоугольником:

- точек пересечения нет;
- существует одна точка пересечения;
- существуют две точки пересечения.

В каждом из этих вариантов для нахождения пересечения отрезка с окном необходимо уметь определять принадлежность точки выпуклому многоугольнику. Из рис. 5.7 видно, что если для любой точки $\overrightarrow{g}$, принадлежащей многоугольнику (или его границе), и произвольной точки ребра $\overrightarrow{f}$ построить вектор $\overrightarrow{m} = \overrightarrow{g} - \overrightarrow{f}$, то выполняется условие $(\overrightarrow{m} \cdot \overrightarrow{n} \geq 0)$, поскольку угол между векторами не может превышать 90 $\deg$. Таким образом, если данное условие выполняется для всех ребер многоугольника, то точка является внутренней.

Таким образом, алгоритм отсечения отрезка начинается с анализа расположения концов отрезка по отношению к окну. Если обе точки лежат внутри окна, то отрезок полностью видимый, и дальнейший поиск прекращается. Если только одна из точек лежит внутри окна, то имеет место случай II, и предстоит найти одну точку пересечения. И, наконец, если обе точки лежат вне окна, то существуют либо две точки пересечения (отрезок пересекает две границы окна), либо ни одной (отрезок полностью невидим). Впрочем, две точки пересечения могут совпадать (если отрезок проходит через вершину многоугольника), но этот случай в дополнительном анализе не нуждается.

Далее выполняется цикл по всем ребрам многоугольника с целью нахождения точек пересечения. Для каждого ребра перед началом поиска пересечения необходимо проверить, не параллельно ли оно с отрезком. Если это так, то можно вычислить расстояние от одного из концов отрезка до прямой, проходящей через ребро $d = ((\overrightarrow{r}_1 - \overrightarrow{p}) \cdot \overrightarrow{n})$. При $d = 0$отрезок лежит на прямой, и остается определить взаимное расположение концов отрезка и концов ребра, что можно сделать простым покоординатным сравнением. При $d \neq 0$отрезок не имеет общих точек с данным ребром.

Клиппирование многоугольников

От задачи отсечения отрезков можно перейти к более сложной: клиппирование произвольных многоугольников. Если многоугольник невыпуклый, то в результате пересечения даже с прямоугольным окном может получиться несколько не связанных между собой фигур, как это показано на рис. 5.8.

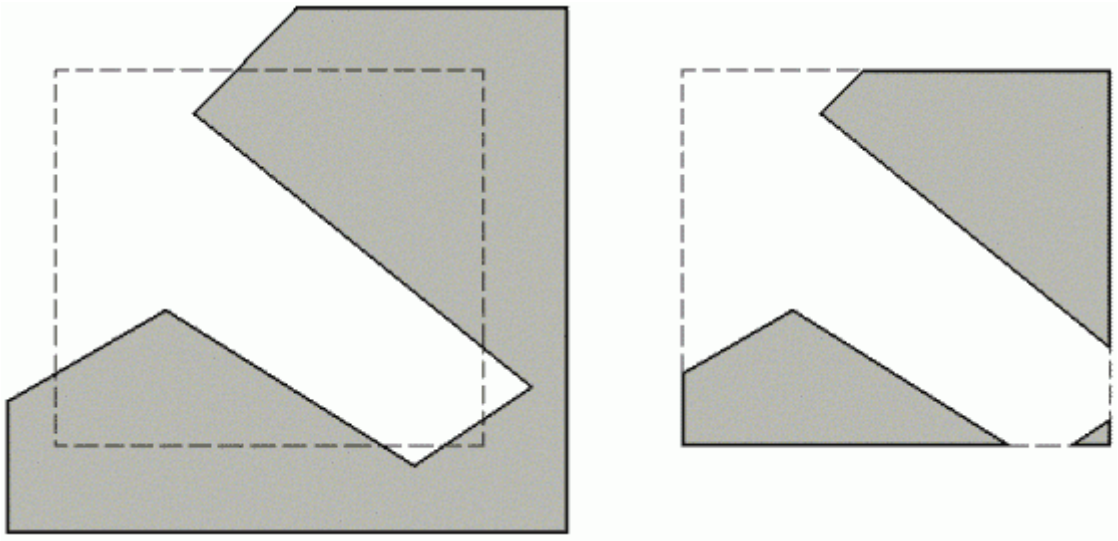
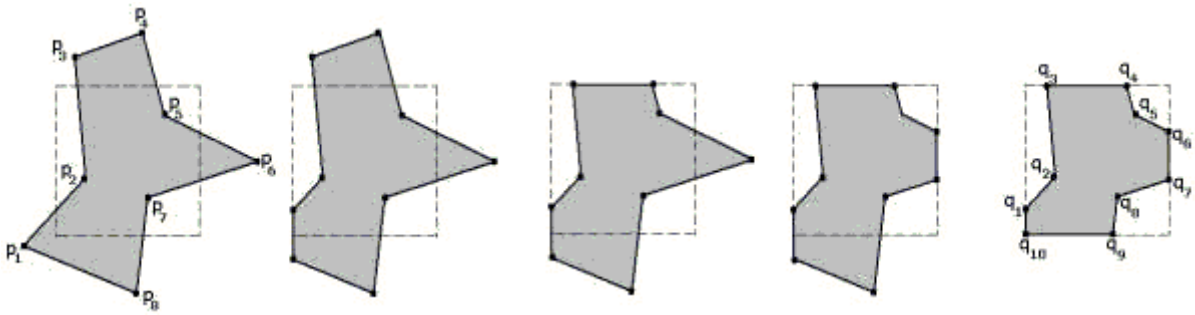


Рис. 5.8. Отсечение невыпуклого многоугольника

Когда стоит задача штрихования замкнутой области одновременно с задачей отсечения, то важно правильно определить принадлежность вновь полученных фигур внутренней или внешней части исходного многоугольника. Пусть исходный многоугольник задан упорядоченным списком вершин $\{\overrightarrow{p}_1, \overrightarrow{p}_2, \ldots, \overrightarrow{p}_n$, соответствующих ребрам $(\overrightarrow{p}_1, \overrightarrow{p}_2), (\overrightarrow{p}_2, \overrightarrow{p}_3), \ldots, (\overrightarrow{p}_{n-1}, \overrightarrow{p}_n), (\overrightarrow{p}_n, \overrightarrow{p}_1)$. Для отсечения такого многоугольника прямоугольным окном можно применить алгоритм, предложенный Сазерлендом и Ходжменом. Идея его заключается в последовательном отсечении части многоугольника прямыми, соответствующими сторонам окна. Результатом его работы является упорядоченный список вершин, лежащих в видимой части окна. На каждом шаге алгоритма образуется некоторая промежуточная фигура, также представленная

упорядоченным списком вершин и ребер. Пример таких последовательных отсечений показан на рис. 5.9. В процессе отсечения последовательно обходится список вершин, причем каждая очередная точка за исключением первой рассматривается как конечная точка ребра, начальной точкой которого является предшествующая точка из списка. Порядок, в котором рассматриваются стороны окна, не имеет значения. В процессе обхода формируется список новых вершин многоугольника.



увеличить изображение

Рис. 5.9. Последовательные шаги клиппирования произвольного многоугольника

На первом шаге для первой вершины в списке определяется ее принадлежность видимой области. Если она видима, то она становится первой точкой первого обрабатываемого ребра и заносится в список новых вершин. Если же она невидима, то в список новых вершин не заносится, но все равно становится первой точкой ребра.

Для анализируемого ребра возможны четыре случая расположения относительно окна.

1. Ребро полностью видимо. Очередная точка заносится в список новых вершин (предыдущая уже должна находиться в этом списке, поскольку ребро полностью видимо).
2. Ребро полностью невидимо. Никаких действий не производится.
3. Ребро выходит из области. Находится точка пересечения ребра со стороной окна и заносится в список новых ребер.
4. Ребро входит в область. Также отыскивается точка пересечения со стороной окна и заносится в список новых вершин. Конечная точка тоже заносится в список новых вершин.

В этом алгоритме постоянно приходится определять видимость точки по отношению к конкретному ребру отсекающего окна. Окно также можно задать в виде упорядоченного списка вершин. Если обход вершин окна осуществляется по часовой стрелке, то его внутренняя область будет расположена по правую сторону от границы.

При этом расположение точки $\overrightarrow{p}$ относительно прямой, которой принадлежит ребро, можно устанавливать различными способами:

1. Выбирается начальная точка $\overrightarrow{s}$ данного ребра и строится вектор $\overrightarrow{r} = \overrightarrow{p} - \overrightarrow{s}$ и вектор внутренней нормали n к границе (ребру) окна. Вычисляется скалярное произведение $d = (\overrightarrow{r} \cdot \overrightarrow{n})$. Если $d \geq 0$, то точка $\overrightarrow{p}$ является видимой.
2. Строится пространственный вектор $\overrightarrow{r}$ (третью координату можно положить равной нулю). Вектор $\overrightarrow{l}$ (также пространственный, лежащий в той же плоскости, что и $\rightarrow$) направлен вдоль ребра (с учетом направления обхода). Вычисляется векторное произведение $\overrightarrow{v} = [\overrightarrow{r} \times \overrightarrow{l}]$. Если координата z у вектора $\overrightarrow{v}$ положительна, то точка $\overrightarrow{p}$ лежит справа от ребра (является видимой).

3. Выписывается каноническое уравнение прямой, проходящей через ребро:

$$f(x, y) + by + c = 0$$

4. Для произвольной внутренней точки окна (x_0, y_0) вычисляется значение $d = f(x_0, y_0)$, а также для точки $\overrightarrow{p} = (x_1, y_1)$ вычисляется $d_1 = f(x_1, y_1)$. Если числа d и d_1 имеют одинаковый знак, то точка $\overrightarrow{p}$ является видимой.

Наиболее просто эти алгоритмы реализуются в случае отсечения прямоугольным окном со сторонами, параллельными осям координат.

Вопросы и упражнения

1. Что такое клиппирование?
2. Если концы отрезков имеют коды 1000 и 0100, сколько сторон окна он может пересекать?
3. При каком значении кода одного из концов отрезка он обязательно будет частично видимым?
4. Если оба конца отрезка лежат вне окна, то при каких кодах концов он может проходить вдоль диагонали окна?
5. Какой из алгоритмов отсечения отрезков эффективнее: приведенный в блок-схеме 5.3 или основанный на делении отрезка пополам?
6. С помощью какого условия можно определить принадлежность точки выпуклому многоугольнику?
7. Будет ли это условие применимо в случае произвольного многоугольника? (подтвердите свой ответ примерами).
8. Какие случаи расположения ребра относительно окна рассматриваются в алгоритме клиппирования произвольного многоугольника?

Лекция 6. Удаление невидимых поверхностей и линий

Исторический экскурс. Методы переборного типа. Метод Z-буфера. Методы удаления нелицевых граней многогранника. Алгоритмы Варнака и Вейлера — Азертона. Методы приоритетов (художника, плавающего горизонта). Метод двоичного разбиения пространства. Алгоритмы построчного сканирования для криволинейных поверхностей. Алгоритм определения видимых поверхностей путем трассировки лучей

Задача удаления невидимых линий и поверхностей является одной из наиболее интересных и сложных в компьютерной графике. Алгоритмы удаления заключаются в определении 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 на рис. 6.1. Рисунок наглядно демонстрирует, что изображение без удаления невидимых линий воспринимается неоднозначно.

Рис. 6.1. Неоднозначность восприятия изображения куба

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Многие из них ориентированы на специализированные приложения. Единого (общего) решения этой задачи, годного для различных случаев, естественно, не существует: для каждого случая выбирается наиболее подходящий метод. Например, для моделирования процессов в реальном времени требуются быстрые алгоритмы, в то время как для формирования сложного реалистического изображения, в котором представлены тени, прозрачность и фактура, учитывающие эффекты отражения и преломления цвета в мельчайших оттенках, фактор времени выполнения уже не так существенен. Подобные алгоритмы работают медленно, и зачастую на вычисления требуется несколько минут или даже часов. 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 такого рода так или иначе включают в себя сортировку, причем главная сортировка ведется по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения или картинной плоскости. 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 в значительной мере зависит от эффективности процесса сортировки.

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают. Алгоритмы, работающие в объектном пространстве, имеют дело с мировой системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные, вообще говоря, лишь погрешностью вычислений. Полученные изображения можно свободно масштабировать. 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.

Мы приведем некоторые из алгоритмов, работающих как в объектном пространстве, так и в пространстве изображения, каждый из которых иллюстрирует одну или несколько основополагающих идей теории алгоритмов удаления невидимых линий и поверхностей.

Удаление нелицевых граней многогранника

Алгоритм Робертса

Этот алгоритм, предложенный в 1963 г., является первой разработкой такого рода и предназначен для удаления невидимых линий при штриховом изображении объектов, составленных из выпуклых многогранников. Он относится к алгоритмам, работающим в объектном пространстве, и очень элегантен с математической точки зрения. В нем очень удачно сочетаются геометрические методы и методы линейного программирования.

Выпуклый многогранник однозначно определяется набором плоскостей, образующих его грани, поэтому исходными данными для алгоритма являются многогранники, заданные списком своих граней. Грани задаются в виде плоскостей, заданных в канонической форме (см. лекцию 3) в объектной системе координат: $ax + by + cz + d = 0$.

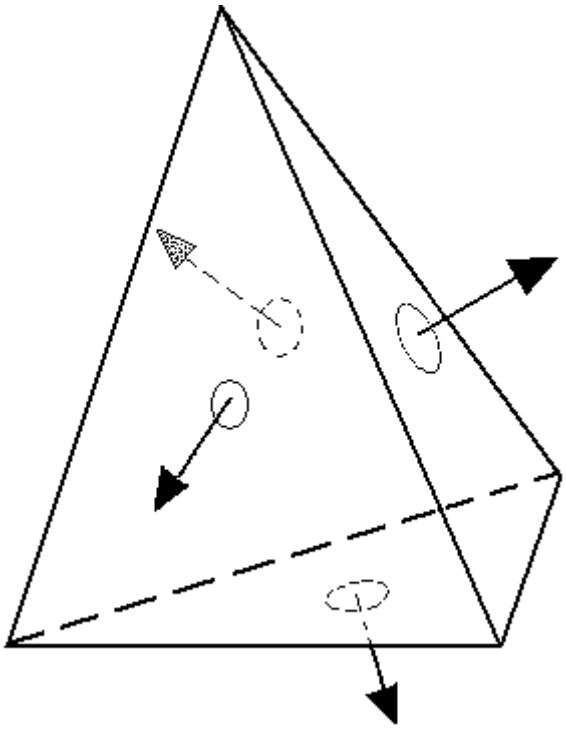


Рис. 6.2. Внешние нормали тетраэдра

Таким образом, каждая плоскость определяется четырехмерным вектором $\overline{P}$, а каждая точка $\overrightarrow{r}$, заданная в однородных координатах, также представляет собой четырехмерный вектор:

$$\overrightarrow{P} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \overrightarrow{r} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Принадлежность точки плоскости можно установить с помощью скалярного произведения, т.е. если $(\overrightarrow{P} \cdot \overrightarrow{r}) = 0$, то точка принадлежит плоскости, если же нет, то знак произведения показывает, по какую сторону от плоскости эта точка находится. В алгоритме Робертса плоскости строятся таким образом, что внутренние точки многогранника лежат в положительной полуплоскости. Это означает, что вектор

(A, B, C) является **внешней нормалью** к многограннику (рис. 6.2). Из векторов плоскостей строится прямоугольная матрица порядка $4 \times n$, которая называется **обобщенной матрицей описания многогранника**:

$$M = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \\ c_1 & c_2 & c_3 & \dots & c_n \\ d_1 & d_2 & d_3 & \dots & d_n \end{pmatrix}.$$

Умножая столбцы матрицы на вектор $\overrightarrow{r}$, получим *n* -мерный вектор, и если все его компоненты неотрицательны, то точка принадлежит многограннику. Это условие будем записывать в виде $(\overrightarrow{r} \cdot M) \geq 0$(имеется в виду умножение вектор-строки на матрицу).

В своем алгоритме Робертс рассматривает только отрезки, являющиеся пересечением граней многогранника.

Из обобщенной матрицы можно получить информацию о том, какие грани многогранника пересекаются в вершинах. Действительно, если вершина $\overrightarrow{v} = (x, y, z, 1)$принадлежит граням $\overrightarrow{P}_1,\ \overrightarrow{P}_2,\ \overrightarrow{P}_3$, то она удовлетворяет уравнениям

$$\left.\begin{aligned} (\overrightarrow{v} \cdot \overrightarrow{P}_1) &= 0 \\ (\overrightarrow{v} \cdot \overrightarrow{P}_2) &= 0 \\ (\overrightarrow{v} \cdot \overrightarrow{P}_3) &= 0 \\ (\overrightarrow{v} \cdot \overrightarrow{e}_4) &= 1 \end{aligned}\right\}, \quad \text{где} \quad \overrightarrow{e}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Эту систему можно записать в матричном виде:

$$\overrightarrow{v} \cdot Q = \overrightarrow{e}_4,$$

где Q - матрица, составленная из вектор-столбцов $\overrightarrow{P}_1, \overrightarrow{P}_2, \overrightarrow{P}_3, \overrightarrow{e}_4$. Значит, координаты вершины определяются соотношением

$$\overrightarrow{v} = \overrightarrow{e}_4 \cdot Q^{(-1)},$$

т.е. они составляют последнюю строку обратной матрицы. А это означает, что если для каких-либо трех плоскостей обратная матрица существует, то плоскости имеют общую вершину.

Алгоритм прежде всего удаляет из каждого многогранника те ребра или грани, которые экранируются самим телом. Робертс использовал для этого простой тест: если одна или обе смежные грани обращены своей внешней поверхностью к наблюдателю, то ребро является видимым. Тест этот выполняется вычислением скалярного произведения координат наблюдателя на вектор внешней нормали грани: если результат отрицательный, то грань видима.

Затем каждое из видимых ребер каждого многогранника сравнивается с каждым из оставшихся многогранников для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Для этого в каждую точку ребра проводится отрезок луча, выходящего из точки расположения наблюдателя. Если отрезок не пересекает ни одного из многогранников, то точка видима. Для решения этой задачи используются параметрические уравнения прямой, содержащей ребро, и луча.

Если заданы концы отрезка $\overrightarrow{r}$ и $\overrightarrow{s}$, а наблюдатель расположен в точке $\overrightarrow{u}$, то отрезок задается уравнением

$$\overrightarrow{v} = \overrightarrow{r} + t \cdot (\overrightarrow{s} - \overrightarrow{r}) \equiv \overrightarrow{r} + t \cdot \overrightarrow{d}, \quad 0 \le t \le 1,$$

а прямая, идущая в точку, соответствующую параметру t, - уравнением

$$\overrightarrow{w} = \overrightarrow{v} + \tau \cdot \overrightarrow{g} = \overrightarrow{r} + t \cdot \overrightarrow{d} + \tau \cdot \overrightarrow{g}.$$

Для определения той части отрезка, которая закрывается каким-либо телом, достаточно найти значения t и τ, при которых произведение вектора $\overrightarrow{w}$ на обобщенную матрицу положительно. Для каждой плоскости $\overrightarrow{P}_i$ записывается неравенство

$$q_i = (\overrightarrow{v} \cdot \overrightarrow{P}_i) + t(\overrightarrow{d} \cdot \overrightarrow{P}_i) + \tau(\overrightarrow{g} \cdot \overrightarrow{P}_i) > 0.$$

Эти условия должны выполняться для всех плоскостей.

Полагая $q_i = 0$, получаем систему уравнений, решения которой дают нам точки "смены видимости" отрезка. Результат можно получить путем совместного решения всевозможных пар уравнений из этой системы. Число всевозможных решений при N плоскостях равно $N \cdot (N-1)/2$.

Так как объем вычислений растет с увеличением числа многоугольников, то желательно по мере возможности сокращать их число, т.е. если мы аппроксимируем некоторую поверхность многогранником, то в качестве граней можно использовать не треугольники, а более сложные многоугольники. При этом, разумеется, встает проблема, как построить такой многоугольник, чтобы он мало отклонялся от плоской фигуры.

Алгоритм Варнока

В отличие от алгоритма Робертса, Варнок в 1968 г. предложил алгоритм, работающий не в объектном пространстве, а в пространстве образа. Он также нацелен на изображение многогранников, а главная идея его основана на гипотезе о способе обработки информации, содержащейся в сцене, глазом и мозгом человека. Эта гипотеза заключается в том, что тратится очень мало времени и усилий на обработку тех областей, которые содержат мало информации. Большая часть времени и труда затрачивается на области с высоким информационным содержимым. Так, например, рассматривая помещение, в котором имеется только картина на стене, мы быстро осматриваем стены, пол и потолок, а затем все внимание сосредоточиваем на картине. В свою очередь, на этой картине, если это портрет, мы бегло отмечаем фон, а затем более внимательно рассматриваем лицо изображенного персонажа, в особенности глаза, губы. Как правило, достаточно детально рассматриваются еще и руки и с чуть меньшим вниманием - одежда.

В алгоритме Варнока и его вариантах делается попытка воспользоваться тем, что большие области изображения однородны. Такое свойство называют **когерентностью**, имея в виду, что смежные области (пиксели) вдоль обеих осей *x* и *y* имеют тенденцию к однородности.

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. В последнем случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом.

Конкретная реализация алгоритма Варнока зависит от метода разбиения окна и от деталей критерия, используемого для того, чтобы решить, является ли содержимое окна достаточно простым. В оригинальной версии алгоритма каждое окно разбивалось на четыре одинаковых подокна. Многоугольник, входящий в изображаемую сцену, по отношению к окну будем называть (рис. 6.3)

- **внешним**, если он целиком находится вне окна;
- **внутренним**, если он целиком расположен внутри окна;
- **пересекающим**, если он пересекает границу окна;
- **охватывающим**, если окно целиком расположено внутри него.

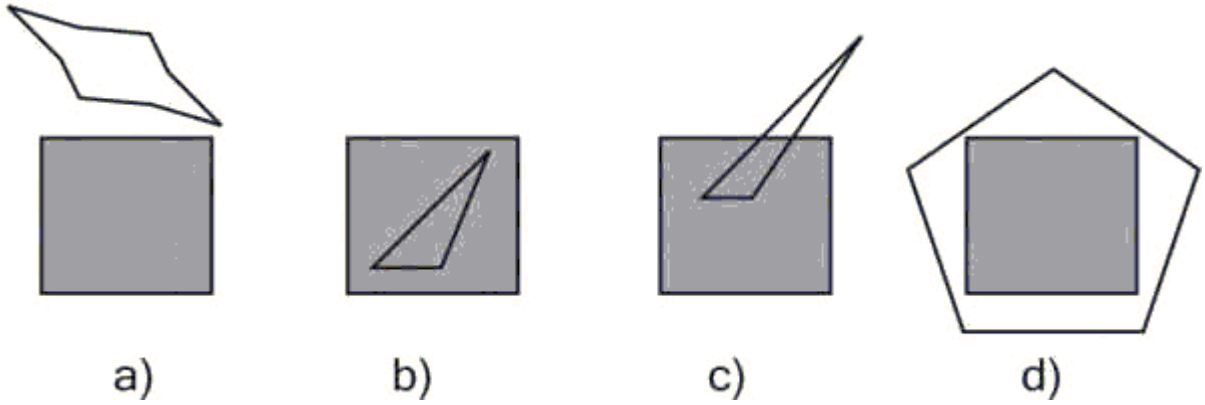


Рис. 6.3. Варианты расположения многоугольника по отношению к окну

Теперь можно в самом общем виде описать алгоритм.

Для каждого окна:

1. Если все многоугольники сцены являются внешними по отношению к окну, то оно пусто; изображается фоновым цветом и дальнейшему разбиению не подлежит.
2. Если только один многоугольник сцены имеет общие точки с окном и является по отношению к нему внутренним, то окно заполняется фоновым цветом, а сам многоугольник заполняется своим цветом.
3. Если только один многоугольник сцены имеет общие точки с окном и является по отношению к нему пересекающим, то окно заполняется фоновым цветом, а часть многоугольника, принадлежащая окну, заполняется цветом многоугольника.
4. Если только один многоугольник охватывает окно и нет других многоугольников, имеющих общие точки с окном, то окно заполняется цветом этого многоугольника.
5. Если существует хотя бы один многоугольник, охватывающий окно, то среди всех таких многоугольников выбирается тот, который расположен ближе всех многоугольников к точке наблюдения, и окно заполняется цветом этого многоугольника.
6. В противном случае производится новое разбиение окна.

Шаги 1–4 рассматривают ситуацию пересечения окна только с одним многоугольником. Они используются для сокращения числа подразбиений. Шаг 5 решает задачу удаления невидимых поверхностей. Многоугольник, находящийся ближе всех к точке наблюдения, экранирует все остальные.

Для реализации алгоритма необходимы функции, определяющие взаимное расположение окна и многоугольника, которые достаточно легко реализуются в случае

прямоугольных окон и выпуклых многоугольников. Для определения, является ли многоугольник охватывающим, внешним или внутренним, можно воспользоваться, например, погружением многоугольника в прямоугольную оболочку. Для определения наличия пересечений можно использовать опорные прямые (так же, как использовались плоскости в алгоритме Робертса). Если же многоугольник невыпуклый, то задача усложняется. Методы решения такого рода задач будут рассмотрены в главе, относящейся к геометрическому поиску.

Следует заметить, что существуют различные реализации алгоритма Варнока. Были предложены варианты оптимизации, использующие предварительную сортировку многоугольников по глубине, т. е. по расстоянию от точки наблюдения, и другие.

Алгоритм Вейлера-Азертона

Вейлер и Азертон попытались оптимизировать алгоритм Варнока в отношении числа выполняемых разбиений, перейдя от прямоугольных разбиений к разбиениям вдоль границ многоугольников (1977). Для этого они использовали ими же разработанный алгоритм отсечения многоугольников. Алгоритм работает в объектном пространстве, и результатом его работы являются многоугольники. В самом общем виде он состоит из четырех шагов.

1. Предварительная сортировка по глубине.
2. Отсечение по границе ближайшего к точке наблюдения многоугольника, называемое сортировкой многоугольников на плоскости.
3. Удаление многоугольников, экранируемых более близкими к точке наблюдения многоугольниками.
4. Если требуется, то рекурсивное разбиение и новая сортировка.

В процессе предварительной сортировки создается список приблизительных приоритетов, причем близость многоугольника к точке наблюдения определяется расстоянием до ближайшей к ней вершины. Затем выполняется отсечение по самому первому из многоугольников. Отсечению подвергаются все многоугольники из списка, причем эта операция выполняется над проекциями многоугольников на картинную плоскость. При этом создаются списки внешних и внутренних фигур. Все попавшие в список внешних не экранируются отсекающим многоугольником. Затем рассматривается список внутренних многоугольников и выполняется сортировка по расстоянию до отсекающего многоугольника. Если все вершины некоторого многоугольника оказываются дальше от наблюдателя, чем самая удаленная из вершин экранирующего, то они невидимы, и тогда они удаляются. После этого работа алгоритма продолжается с внешним списком.

Если какая-то из вершин внутреннего многоугольника оказывается ближе к наблюдателю, чем ближайшая из вершин экранирующего многоугольника, то такой многоугольник является частично видимым. В этом случае предварительный список приоритетов некорректен, и тогда в качестве нового отсекающего многоугольника выбирается именно этот "нарушитель порядка". При этом используется именно исходный многоугольник, а не тот, что получился в результате первого отсечения. Такой подход позволяет минимизировать число разбиений.

Этот алгоритм в дальнейшем был обобщен Кэтмулом (1974) для изображения гладких бикубических поверхностей. Его подход заключался в том, что разбиению подвергалась поверхность. Коротко этот алгоритм можно описать так:

1. Рекурсивно разбивается поверхность до тех пор, пока проекция элемента на плоскость изображения не будет покрывать не больше одного пикселя.
2. Определить атрибуты поверхности в этом пикселе и изобразить его.

Эффективность такого метода, как и алгоритм Варнока, зависит от эффективности разбиений. В дальнейшем этот алгоритм был распространен на сплайновые поверхности.

Метод Z-буфера

Это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом в 1975 г. Работает этот алгоритм в пространстве изображения. Идея Z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов каждого пикселя в пространстве изображения, а Z-буфер предназначен для запоминания глубины (расстояния от картинной плоскости) каждого видимого пикселя в пространстве изображения. Поскольку достаточно распространенным является использование координатной плоскости XOY в качестве картинной плоскости, то глубина равна координате z точки, отсюда и название буфера. В процессе работы значение глубины 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 глубины. 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. В последнее время в связи с быстрым ростом возможностей вычислительной техники этот недостаток становится менее лимитирующим. Но в то время, когда алгоритм еще только появился, приходилось изобретать способы создания буфера как можно большего объема при имеющемся ресурсе памяти.

Например, можно разбивать пространство изображения на 4, 16 или больше прямоугольников или полос. В предельном варианте можно использовать буфер размером в одну строку развертки. Для последнего случая был разработан **алгоритм построчного сканирования**. Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование Z-буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены.

Другой недостаток алгоритма состоит в трудоемкости реализации эффектов, связанных с полупрозрачностью, и ряда других специальных задач, повышающих реалистичность изображения. Поскольку алгоритм заносит пиксели в буфер кадра в произвольном порядке, то довольно сложно получить информацию, которая необходима для методов, основывающихся на предварительном анализе сцены.

В целом алгоритм выглядит так:

1. Заполнить буфер кадра фоновым значением цвета.
2. Заполнить Z -буфер минимальным значением z (глубины) .
3. Преобразовать изображаемые объекты в растровую форму в произвольном порядке.

4. Для каждого объекта:

4.1. Для каждого пикселя (x, y)образа вычислить его глубину $z(x, y)$.

4.2. Сравнить глубину $z(x, y)$со значением глубины, хранящимся в Z-буфере в этой же позиции.

4.3. Если $z(x, y) > Z - \text{буфер}(x, y)$, то занести атрибуты пикселя в буфер кадра и заменить $Z - \text{буфер}(x, y)\text{на}z(x, y)$. В противном случае никаких действий не производить.

Алгоритм, использующий Z-буфер, можно также применять для построения сечений поверхностей. Изменится только оператор сравнения:

$z(x, y) > Z - \text{буфер}(x, y)$ и $z(x, y) = z$ сечения
где z сечения - глубина искомого сечения.

Методы приоритетов (художника, плавающего горизонта)

Здесь мы рассмотрим группу методов, учитывающих специфику изображаемой сцены для удаления невидимых линий и поверхностей.

При изображении сцен со сплошным закрашиванием поверхностей можно воспользоваться **методом художника**: элементы сцены изображаются в последовательности от наиболее удаленных от наблюдателя к более близким. При экранировании одних участков сцены другими невидимые участки просто закрашиваются. Если вычислительная трудоемкость получения изображения для отдельных элементов достаточно высока, то такой алгоритм будет не самым лучшим по эффективности, но зато мы избежим анализа (и вполне возможно, тоже дорогостоящего), позволяющего установить, какие же из элементов изображать не надо в силу их невидимости. Например, при изображении правильного многогранника мы довольно легко можем упорядочить его грани по глубине, но такая сортировка для произвольного многогранника возможна далеко не всегда. Мы рассмотрим применение этого метода на примере изображения поверхности, заданной в виде однозначной функции двух переменных.

Пусть поверхность задана уравнением

$$z = f(x, y), \quad a \le x \le b, \quad c \le y \le d.$$
В качестве картинной плоскости выберем плоскость XOY. В области задания функции на осях координат построим сетку узлов:
$$a = x_0 < x_1 < \ldots < x_{n-1} = b, \quad c = y_0 < y_1 < \ldots < y_{m-1} < y_m = d.$$

Тогда $z_{ij} = f(x_i, y_i)$представляют собой набор "высот" для данной поверхности по отношению к плоскости XOY. Поверхность будем аппроксимировать треугольниками с вершинами в точках $\overrightarrow{r}_{ij} = (x_i, y_i, z_{ij})$так, что каждому прямоугольнику сетки узлов будут соответствовать два треугольника: $tr^1_{i\,j} = \{\overrightarrow{r}_{ij}, \overrightarrow{r}_{i+1\,j}, \overrightarrow{r}_{i+1\,j+1}\}$ и $tr^2_{i\,j} = \{\overrightarrow{r}_{ij}, \overrightarrow{r}_{i\,j+1}, \overrightarrow{r}_{i+1\,j+1}\}$. Для построения наглядного изображения поверхности повернем ее на некоторый угол сначала относительно оси OX, а затем

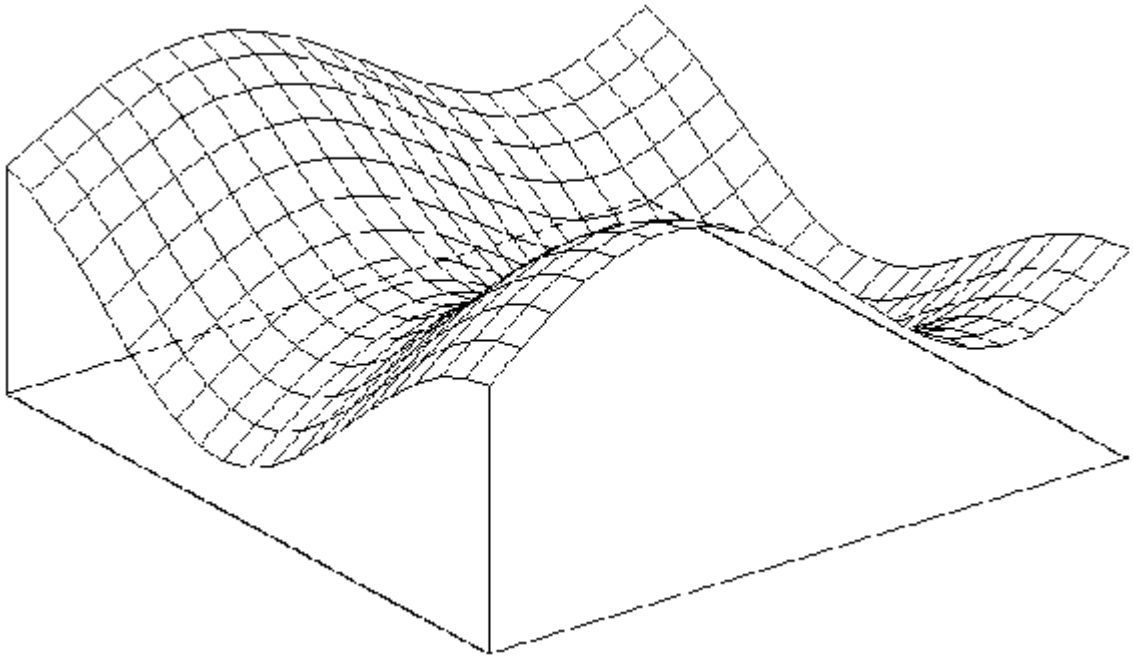
относительно оси OY, причем направление вращения выберем таким образом, что точки, соответствующие углам координатной сетки, расположатся в следующем порядке по удаленности от картинной плоскости: $\overrightarrow{r}_{nm}, \overrightarrow{r}_{n0}, \overrightarrow{r}_{0m}, \overrightarrow{r}_{00}$, т.е. точка $\overrightarrow{r}_{nm}$ окажется наиболее близкой к картинной плоскости (и наиболее удаленной от наблюдателя). Предполагается, что способ закрашивания треугольников уже определен. Тогда процесс изображения поверхности можно коротко записать так:

Для $i = n, \ldots, 1$

Для $j = m, \ldots, 1$

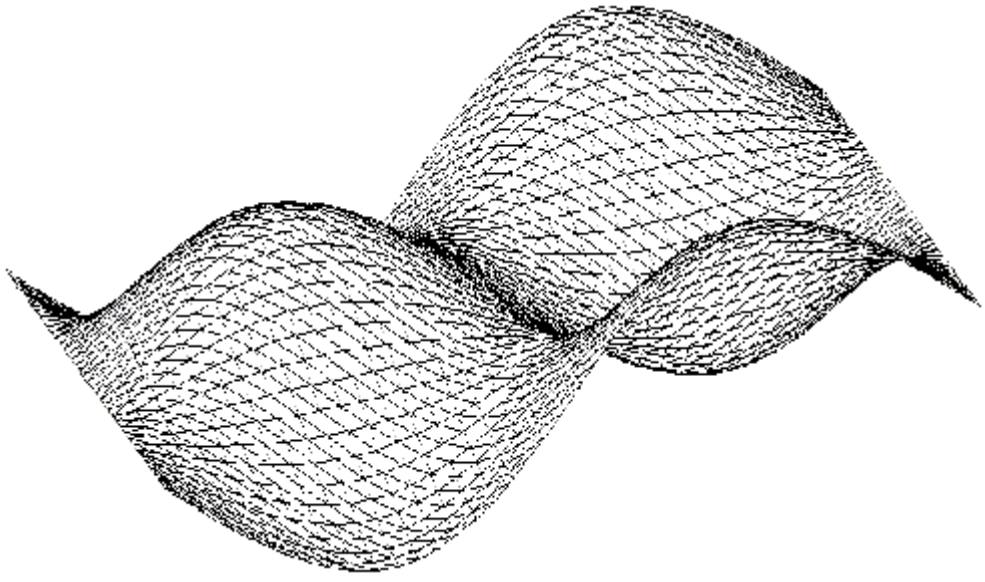
Нарисовать $tr^1_{i\,j}$; нарисовать $tr^2_{i\,j}$.

При такой последовательности вывода изображения мы продвигаемся от самого удаленного треугольника к все более близким, частично закрашивая уже изображенные участки поверхности.



увеличить изображение

Рис. 6.4. Простое каркасное изображение с поверхности



увеличить изображение
Рис. 6.5. Каркасное изображение диагональными ребрами

Алгоритм художника можно применять для полностью закрашенной сцены, а для каркасного изображения, когда объект представляется в виде набора кривых или ломаных линий, он непригоден. Для этого случая предложен еще один метод, весьма эффективный - **метод плавающего горизонта**. Вернемся к предыдущему примеру изображения поверхности. Каркасное изображение получается путем изображения кривых, получаемых при пересечении этой поверхности плоскостями $x = x_i$ и $y = y_i$ (рис. 6.4).

На самом деле мы будем рисовать четырехугольник и одну диагональ. В процессе рисования нам понадобятся два целочисленных массива: $LHor$ (нижний горизонт) и $HHor$ (верхний горизонт) размерностью, соответствующей горизонтальному размеру экрана в пикселях. Они нужны для анализа видимости участков изображаемых отрезков. Сначала мы инициализируем верхний горизонт нулем, а нижний - максимальным значением вертикальной координаты на экране. Каждая выводимая на экран точка может закрывать другие точки, которые "скрываются за горизонтом". По мере рисования нижний горизонт "опускается", а верхний "поднимается", постепенно оставляя все меньше незакрытого пространства. В отличие от метода художника, здесь мы продвигаемся от ближнего угла к дальнему. Теперь опишем алгоритм подробнее.

Функция $segment$ в этом фрагменте предназначена для вывода на экран отрезка прямой, причем в момент инициализации очередного пикселя (i, j) она выполняет следующие действия:

$$\text{Если } (HHor[i] < j),\ \text{то } HHor[i] = j;\ \text{вывести пиксель;}$$
$$\text{Иначе если } (LHor[i] > j),\ \text{то } LHor[i] = j;\ \text{вывести пиксель.}$$

Таким образом, пиксель выводится только в том случае, если он выше верхнего или ниже нижнего горизонта, после чего его координаты уже сами становятся одним из горизонтов. А в целом алгоритм будет выглядеть так:

$$\text{Для } i = 0, \ldots, n-1$$
$$\quad \text{Для } j = 0, \ldots, m-1$$
$$\quad\quad segment(x_i, y_i, x_{i+1}, y_j); segment(x_i, y_i, x_{i+1}, y_{j+1});$$
$$\quad\quad\quad segment(x_i, y_i, x_i, y_{j+1}).$$

На рис. 6.5 приведен пример изображения поверхности с использованием этого алгоритма.

Алгоритмы построчного сканирования для криволинейных поверхностей

Идея построчного сканирования, предложенная в 1967 г. Уайли, Ромни, Эвансом и Эрдалом, заключалась в том, что сцена обрабатывается в порядке прохождения сканирующей прямой. В объектном пространстве это соответствует проведению секущей плоскости, перпендикулярной пространству изображения. Строго говоря, алгоритм работает именно в пространстве изображения, отыскивая точки пересечения сканирующей прямой с ребрами многоугольников, составляющих картину (для случая изображения многогранников). Но при пересечении очередного элемента рисунка выполняется анализ глубины полученной точки и сравнение ее с глубиной других точек на сканирующей плоскости. В некоторых случаях можно построить список ребер, упорядоченный по глубине, но зачастую это невозможно выполнить корректно. Поэтому сканирование сочетают с другими методами.

Один из таких методов мы уже рассматривали - метод Z–буфера, в котором буфер инициализируется заново для каждой сканирующей строки. Другой метод **называют интервальным алгоритмом построчного сканирования**. В нем сканирующая строка разбивается проекциями точек пересечения ребер многоугольников на интервалы, затем в каждом из интервалов выбираются видимые отрезки. В этой ситуации их уже можно отсортировать по глубине. Мы остановимся чуть подробнее на методе построчного сканирования для криволинейных поверхностей.

В описании метода приоритетов поверхность задавалась в виде функции двух переменных, здесь мы будем задавать поверхность параметрическими уравнениями:

$$x = x(u,v), \quad y = y(u,v), \quad z = z(u,v).$$

Пересечение сканирующей плоскости $y = y_1$ с поверхностью дает нам так называемую **линию уровня**, или **изолинию**. Эта кривая может быть неодносвязной, т. е. состоять из нескольких отдельных кривых. Чтобы получить эту кривую, мы должны решить уравнение

$$y(u,v) = y_i$$

и, определив значения параметров u, v, найти точки кривой. Для получения решения можно воспользоваться численными итерационными методами, но это вносит дополнительные проблемы (например, при плохом выборе начального приближения итерационный процесс может не сойтись). Выбор подходящего метода решения лежит вне задач нашего курса, поэтому перейдем к описанию алгоритма, считая, что он уже выбран и надежно работает.

Для каждой сканирующей строки со значением ординаты y_i :

  Для каждого значения абсциссы x_j :

    Для всех решений уравнений $u = u(x_j, y_j),\ v = v(x_j, y_j)$

    вычислить глубину z=z(u,v).

      Определить точку с наименьшим значением глубины

      и изобразить.

Второй шаг этого алгоритма предполагает, что решение отыскивается только для тех элементов поверхности (или группы поверхностей, если речь идет о более сложной сцене, содержащей составные объекты), которые при данном значении ординаты пересекают сканирующую плоскость. Предварительный анализ в некоторых случаях позволяет оптимизировать алгоритм.

Метод двоичного разбиения пространства

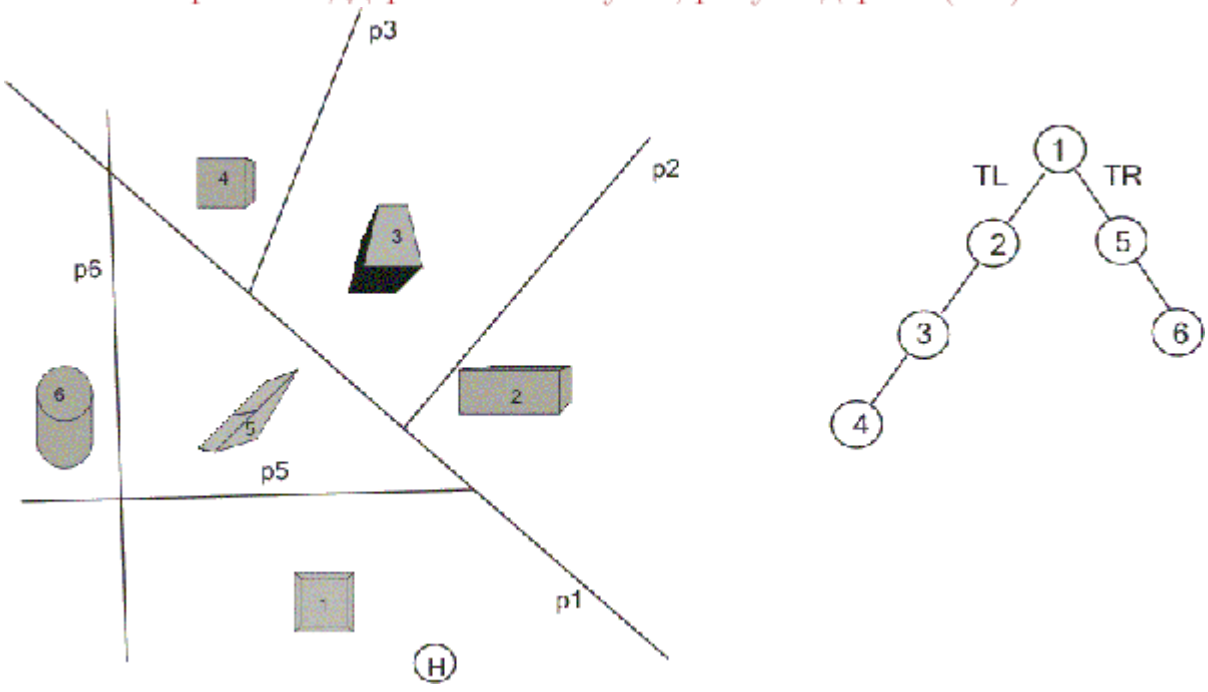
Теперь разберем один способ использования метода художника при изображении пространственных сцен, содержащих несколько объектов или составные объекты. Это так называемый **метод двоичного разбиения пространства плоскостями**.

Плоскости, как обычно, будут задаваться с помощью вектора нормали $\overrightarrow{n}$ и расстояния до начала координат d (с точностью до знака). Пусть изображаемая сцена состоит из набора непересекающихся граней $F_1, F_2, \ldots, F_n$ (они могут иметь общие прямолинейные участки границы). Проведем плоскость P_1, разбивающую все пространство на два полупространства, в одном из которых находится наблюдатель. Предположим, что плоскость при этом не пересекает ни одну из граней (но может содержать участок ее границы). Тогда грани, находящиеся в одном полупространстве с наблюдателем, могут заслонять от него часть граней из второго полупространства, но не наоборот. Это означает, что они должны изображаться позже. Разобьем плоскостью

P_2 второе полупространство и снова определим, какая группа граней из него должна изображаться раньше. Продолжая этот процесс до того уровня, когда все пространство будет разбито плоскостями на секции, в каждой из которых будет находиться только одна грань, мы получим упорядоченный набор граней. Этот порядок можно изобразить в виде двоичного дерева. В контексте рассматриваемого алгоритма это дерево представляет собой структуру данных T, элементами которой являются указатель на грань изображаемой сцены, плоскость, отделяющая эту грань, указатели на левое и правое поддерево TL и TR. Такой элемент называется узлом дерева.

В каждом узле дерева левое поддерево будет содержать грани, отделенные плоскостью, а правое - не отделенные. Рисование сцены осуществляется с помощью рекурсивного алгоритма следующего вида:

Рисуем дерево (T) :
Если наблюдатель находится в положительной полуплоскости, то:
    Если правое поддерево TR не пусто, рисуем дерево (TR).
    Рисуем корневую грань.
    Если левое поддерево TL не пусто, рисуем дерево (TL).
Иначе
    Если левое поддерево TL не пусто, рисуем дерево (TL).
    Рисуем корневую грань.
    Если правое поддерево TR не пусто, рисуем дерево (TR).



увеличить изображение
Рис. 6.6. Разбиение пространства и соответствующее ему дерево

Построение плоскостей и дерева в данном случае осуществляется "вручную". Для эффективности работы алгоритма надо стремиться к тому, чтобы дерево было сбалансированным. Если какие-то грани не удается отделить, то их пересекают плоскостями и рисуют как два объекта. Способ определения, по какую сторону плоскости находится наблюдатель, а по какую - грань, очень прост. Параметр плоскости d для каждой грани будем задавать так, чтобы грань находилась в

положительной полуплоскости. Тогда если при подстановке координат наблюдателя в это уравнение получаем положительное значение, то он находится в одной полуплоскости с гранью, если нет, то в разных.

Алгоритм может применяться не только к многогранникам, но и вообще к любой сцене при условии, что имеется алгоритм изображения составляющих ее объектов. На рис. 6.6 изображена проекция сцены, разбитой вертикальными плоскостями, и соответствующее ей дерево. Положение наблюдателя отмечено кружком с буквой **H**. При этой точке зрения объекты будут изображаться в последовательности 5, 6, 1, 2, 3, 4.

Метод трассировки лучей

Главная идея этого алгоритма была предложена в 1968 г. А.Аппелем, а первая реализация была выполнена в 1971 г.

Наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект, отражается или преломляется согласно законам оптики и затем каким-то путем доходит до глаза наблюдателя. Из огромного множества лучей света, выпущенных источником, лишь небольшая часть дойдет до наблюдателя. Следовательно, отслеживать пути лучей в таком порядке неэффективно с точки зрения вычислений. Аппель предложил отслеживать (трассировать) лучи в обратном направлении, т.е. от наблюдателя к объекту. В первой реализации этого метода трассировка прекращалась, как только луч пересекал поверхность видимого непрозрачного объекта; т.е. луч использовался только для обработки скрытых или видимых поверхностей. Впоследствии были реализованы алгоритмы трассировки лучей с использованием более полных моделей освещения с учетом эффектов отражения одного объекта от поверхности другого, преломления, прозрачности и затенения.

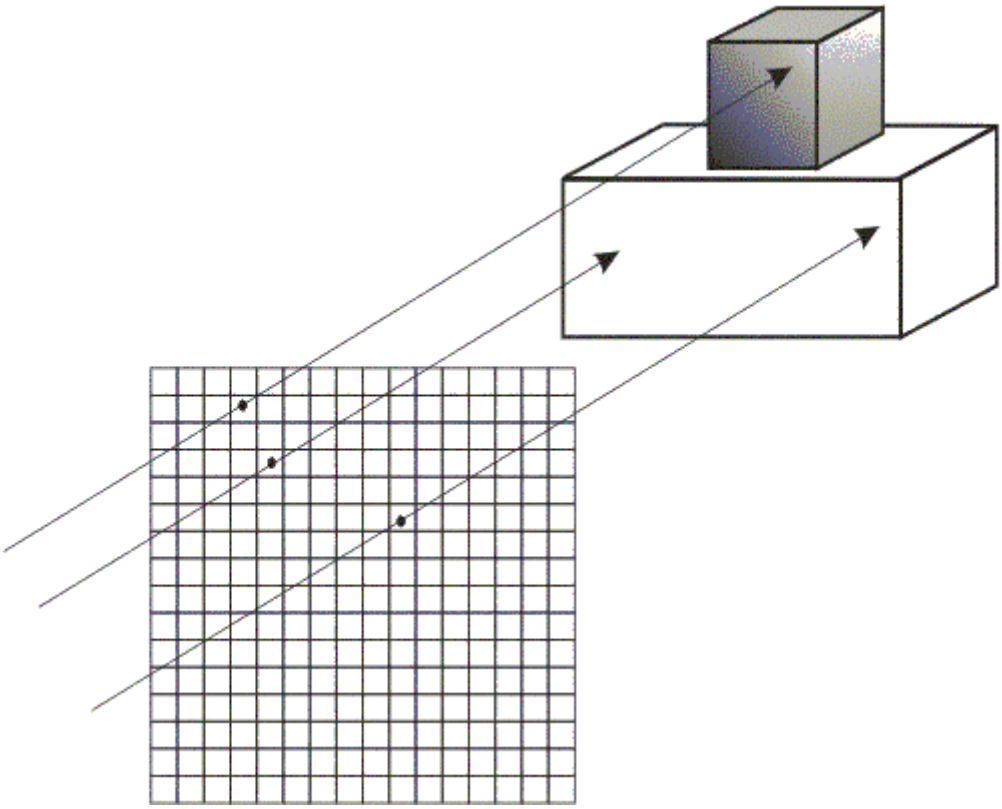


Рис. 6.7. Трассировка параллельными лучами

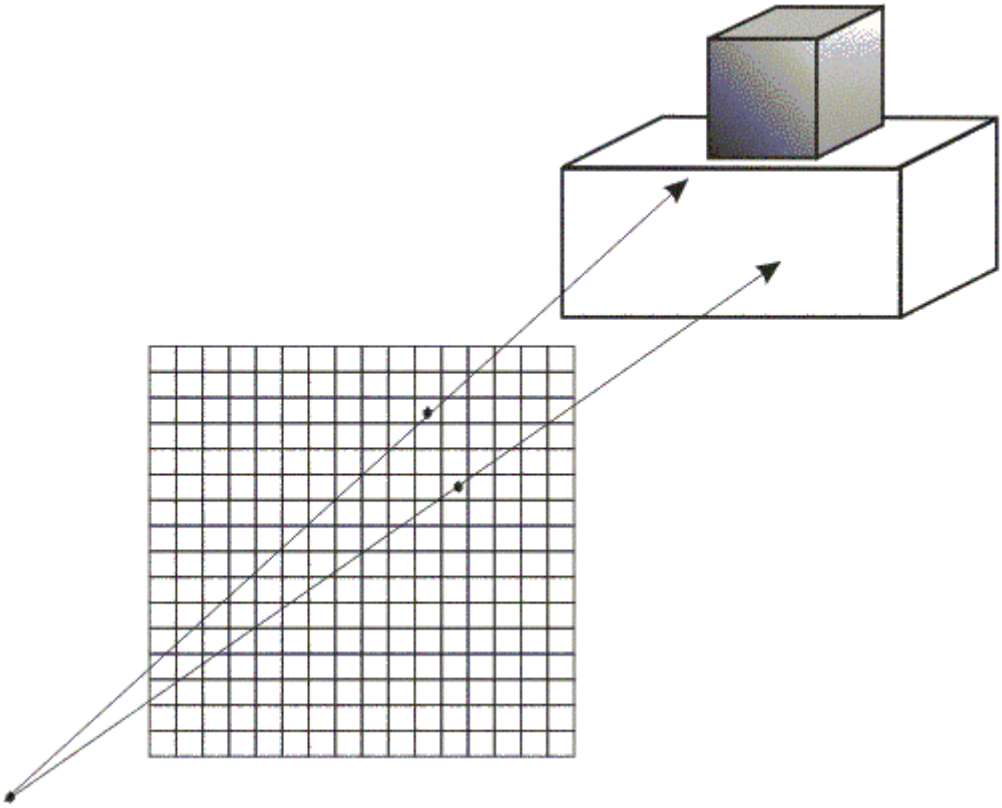


Рис. 6.8. Трассировка с центральной точкой

В этом алгоритме предполагается, что сцена уже преобразована в пространство изображения. Если используется ортографическая проекция, то точка зрения или наблюдатель находится в бесконечности на положительной полуоси OZ. В этом случае все световые лучи, идущие от наблюдателя, параллельны оси (рис. 6.7). Каждый луч проходит через пиксель растра до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Можно получить большое количество пересечений, если рассматривать много объектов. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением z представляет видимую поверхность для данного пикселя. Атрибуты этого объекта используются для определения характеристик пикселя.

Если точка зрения находится не в бесконечности (перспективная проекция), алгоритм трассировки лучей лишь незначительно усложняется. Здесь предполагается, что наблюдатель по-прежнему находится на положительной полуоси OZ. Картинная плоскость, т.е. растр, перпендикулярна оси OZ, как показано на рис. 6.8.

Наиболее важным и трудоемким элементом этого алгоритма является процедура определения пересечений, поскольку эта задача отнимает наибольшую часть времени всей работы алгоритма. Поэтому эффективность методов поиска особенно важна. Объекты сцены могут состоять из набора плоских многоугольников, многогранников или тел, ограниченных замкнутыми параметрическими поверхностями. Для ускорения поиска важно иметь эффективные критерии, позволяющие исключить из процесса заведомо лишние объекты.

Одним из способов сокращения числа пересекаемых объектов является погружение объектов в выпуклую оболочку - сферическую или в форме параллелепипеда. Поиск пересечения с такой оболочкой очень прост, и если луч не пересекает оболочки, то не нужно больше искать пересечений этого объекта с лучом.

Особенно просто выполняется тест на пересечение со сферической оболочкой (в лекции 3 были рассмотрены задачи о пересечении луча со сферой и плоскостью). Несколько большего объема вычислений требует задача о пересечении с прямоугольным параллелепипедом, поскольку необходимо проверить пересечение луча по меньшей мере с тремя бесконечными плоскостями, ограничивающими прямоугольную оболочку. Поскольку точки пересечения могут оказаться вне граней этого параллелепипеда, для каждой из них следует, кроме того, произвести проверку на попадание внутрь. Следовательно, для трех измерений тест с прямоугольной оболочкой оказывается более медленным, чем тест со сферической оболочкой.

После выполнения этих первичных тестов начинается процесс поиска пересечений с объектами, попавшими в список потенциально видимых. При этом задача формирования изображения не исчерпывается нахождением самой точки пересечения: если мы учитываем эффекты отражения и преломления, необходимо отслеживать дальнейший путь луча, для чего, как правило, требуется восстановить нормаль к поверхности, а также определить направление отраженного или преломленного луча. В связи со всеми этими задачами важно выбрать достаточно удобные аппроксимации поверхностей, составляющих сцену. Определение атрибутов пикселя, выводимого в конечном итоге на экран, зависит от выбора модели освещения, о чем более подробно будет рассказано в последующих лекциях.

Алгоритм трассировки лучей для простых непрозрачных поверхностей можно представить следующим образом.

Создать список объектов, содержащий:

- полное описание объекта: тип, поверхность, характеристики, тип оболочки и т.п.;
- описание оболочки: центр и радиус для сферы или шесть значений для параллелепипеда $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$.

Для каждого трассируемого луча:

Выполнить для каждого объекта трехмерный тест на пересечение с оболочкой. Если луч пересекает эту оболочку, то занести объект в список активных объектов.

Если список активных объектов пуст, то изобразить данный пиксель с фоновым значением цвета и продолжать работу. В противном случае для каждого объекта из списка активных объектов:

- Найти пересечения со всеми активными объектами.
- Если список пересечений пуст, то изобразить данный пиксель с фоновым значением цвета.
- В противном случае в списке пресечений найти ближайшее к наблюдателю (с максимальным значением z) и определить атрибуты точки.
- Изобразить данный пиксель, используя найденные атрибуты пересеченного объекта и соответствующую модель освещенности.

В настоящее время алгоритм трассировки, несмотря на вычислительную сложность, стал очень популярен, особенно в тех случаях, когда время формирования изображения не очень существенно, но хочется добиться как можно большей реалистичности изображения.

Вопросы и упражнения

- В чем заключается суть удаления невидимых линий и поверхностей?
- В каком пространстве работает алгоритм Робертса?

- Для каких объектов примеряется алгоритм Робертса?
- Что представляет собой вектор-столбец обобщенной матрицы описания многогранника?
- Как интерпретируется выражение $(\vec{r} \cdot M) \geq 0$(M - обобщенная матрица) в алгоритме Робертса?
- В каком пространстве работает алгоритм Варнока?
- Какие типы расположения многоугольника относительно окна рассматриваются в алгоритме Варнока?
- Который из шести шагов алгоритма решает задачу об удалении невидимых поверхностей?
- В каком пространстве работает алгоритм Вейлера-Азертона?
- В чем принципиальное отличие алгоритма Вейлера-Азертона от алгоритма Варнока?
- Какое обобщение алгоритма Вейлера-Азертона предложил Кэтмул?
- Кем предложен алгоритм Z-буфера?
- В чем недостатки алгоритма Z-буфера?
- На чем основаны методы приоритетов?
- Для какого вида изображения разработан метод художника?
- Для какого вида изображения разработан метод плавающего горизонта?
- Что общего между алгоритмом построчного сканирования и методом Z-буфера?
- В чем состоит идея метода трассировки?
- Какие бывают виды трассировки?
- Какие приемы используются для повышения эффективности алгоритма трассировки?

Лекция 7. Проецирование пространственных сцен

Основные типы проекций. Прямая и перспективная проекция. Специальные картографические проекции. Экзотические проекции земной сферы

Основные типы проекций

В математическом смысле проекции - это преобразования точек пространства размерности *n* в точки пространства размерности меньшей, чем *n*, или, как еще говорят, **на подпространство исходного пространства**. В компьютерной графике рассматриваются преимущественно проекции трехмерного пространства образа на двумерную картинную плоскость. Проекция трехмерного объекта, представленного в виде совокупности точек, строится при помощи прямых проецирующих лучей, которые называются проекторами и которые выходят из центра проекции, проходят через каждую точку объекта и, пересекая картинную плоскость, образуют проекцию.

Определенный таким образом класс проекций называют **плоскими геометрическими проекциями**, поскольку проецирование в этом случае производится на **проекционную плоскость** и в качестве проекторов используются прямые. Существуют и другие проекции, в которых проецирование осуществляется на криволинейные поверхности или же проецирование осуществляется не с помощью прямых (такие проекции используются, например, в картографии).

Следует отметить, что, приводя иллюстрации к данной главе, мы вынуждены использовать те же самые проекции, методы построения которых собираемся описать. Хочется надеяться, что материал из-за этого не будет выглядеть более неясным, чем при отсутствии рисунков.

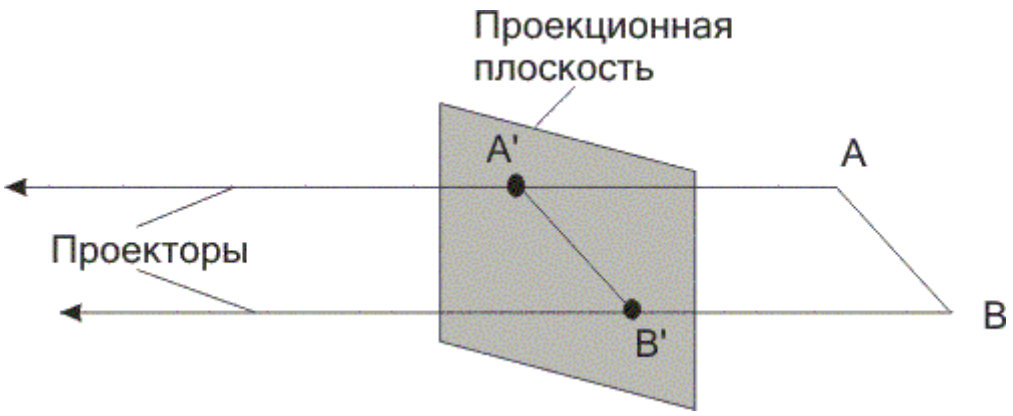
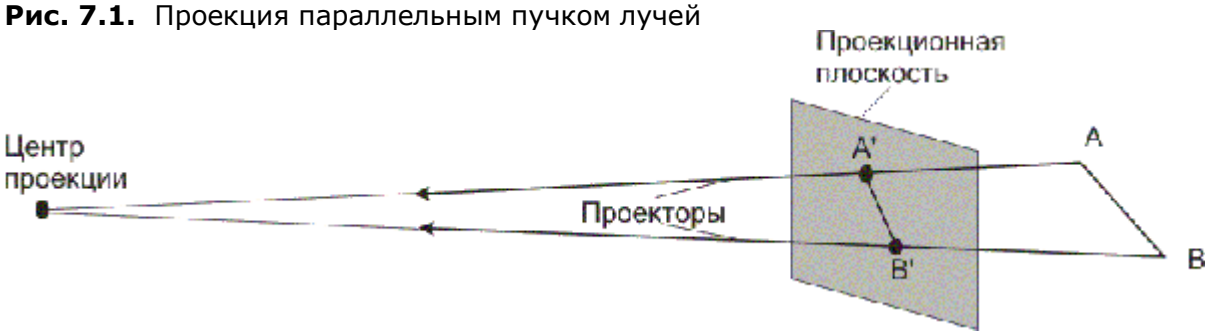


Рис. 7.1. Проекция параллельным пучком лучей



увеличить изображение

Рис. 7.2. Центральная проекция

Плоские геометрические проекции подразделяются на два основных класса: центральные и параллельные. Различие между ними определяется соотношением между центром проекции и проекционной плоскостью. Если расстояние между ними конечно, то проекция будет центральной, если же оно бесконечно, то проекция будет параллельной. Параллельные проекции названы так потому, что центр проекции бесконечно удален и все проекторы параллельны. При описании центральной проекции мы явно задаем ее центр проекции, в то время как, определяя параллельную проекцию, мы указываем направление проецирования. На рис. 7.1 и 7.2 показаны две различные проекции одного и того же отрезка, а также проекторы, проходящие через его конечные точки. Поскольку проекция отрезка сама является отрезком, то достаточно спроецировать одни лишь конечные точки и соединить их.

Центральная проекция порождает визуальный эффект, аналогичный тому, к которому приводят фотографические системы или зрительная система человека, и поэтому используется в случаях, когда желательно достичь определенной степени реалистичности. Этот эффект называется **перспективным укорачиванием**: по мере увеличения расстояния от центра до объекта размер получаемой проекции уменьшается. Это, с другой стороны, означает, что хотя центральная проекция объектов является реалистичной, она оказывается непригодной для представления точной формы и размеров объектов: из проекции нельзя получить информацию об относительных расстояниях; углы сохраняются только на тех гранях объекта, которые параллельны проекционной плоскости; проекции параллельных линий в общем случае не параллельны. Так, при центральной проекции куба в большинстве случаев мы получаем картину, вообще не имеющую параллельных отрезков.

Параллельная проекция порождает менее реалистичное изображение, поскольку отсутствует перспективное укорачивание, хотя при этом могут иметь место различные постоянные укорачивания вдоль каждой из осей. Проекция фиксирует истинные размеры (с точностью до скалярного множителя), и параллельные прямые остаются параллельными. Как и в случае центральной проекции, углы сохраняются только на тех гранях объекта, которые параллельны проекционной плоскости.

Параллельные проекции

Параллельные проекции разделяются на два типа в зависимости от соотношения между направлением проецирования и нормалью к проекционной плоскости. Если эти направления совпадают, т.е. направление проецирования является нормалью к проекционной плоскости, то проекция называется **ортографической**. Если же проекторы не ортогональны к проекционной плоскости, то проекция называется **косоугольной**.

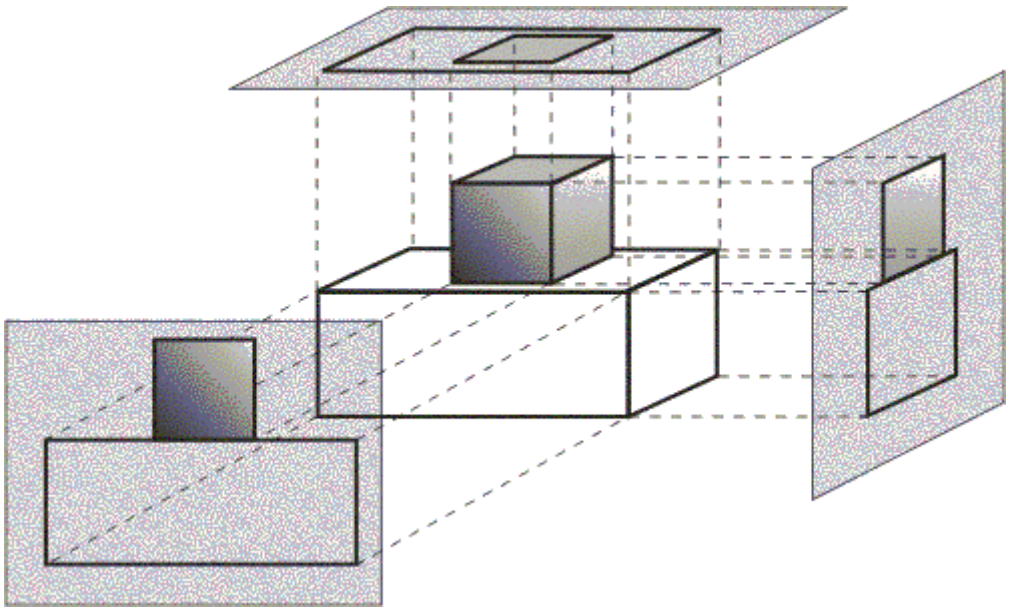
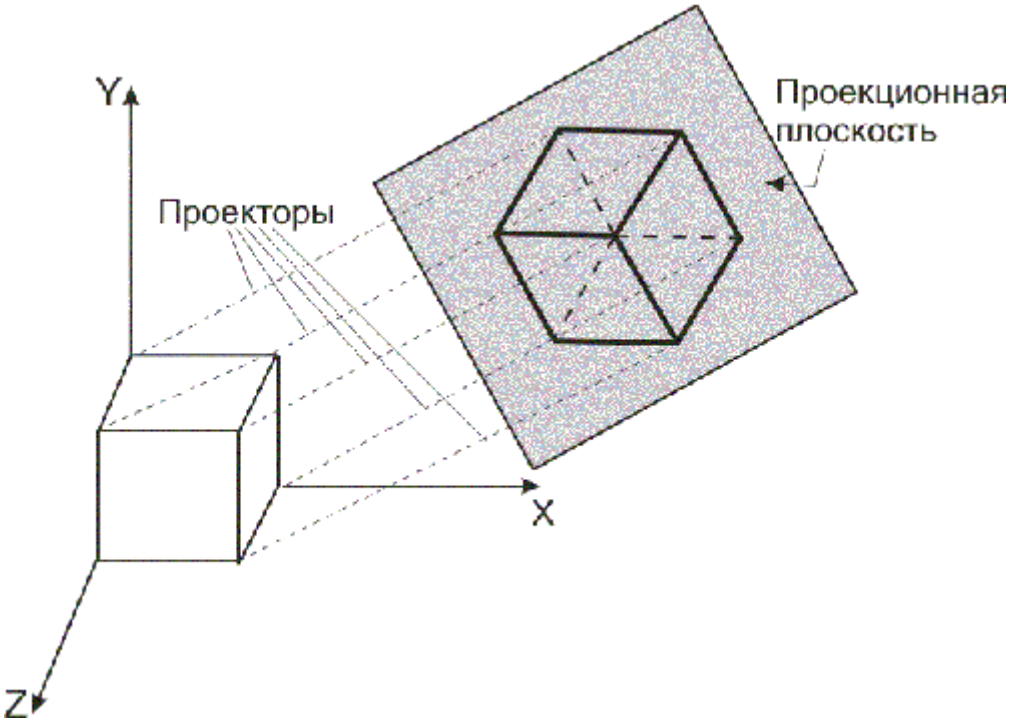


Рис. 7.3. Ортографические проекции

В инженерной графике наиболее широко используемыми видами ортографических проекций являются вид спереди, вид сверху (план) и вид сбоку, в которых проекционная плоскость перпендикулярна главным координатным осям, совпадающим вследствие этого с направлением проецирования (рис. 7.3). Поскольку каждая проекция отображает лишь одну сторону объекта, часто совсем непросто представить себе пространственную структуру проецируемого объекта, даже если рассматривать сразу несколько проекций одного и того же объекта. Но тем не менее такие чертежи позволяют определять реальные размеры объекта.

В случае аксонометрических ортографических проекций используются проекционные плоскости, не перпендикулярные главным координатным осям, поэтому на них изображается сразу несколько сторон объекта, так же как и при центральном проецировании, однако в аксонометрии укорачивание постоянно, тогда как в случае центральной проекции оно связано с расстоянием от центра проекции. При аксонометрическом проецировании сохраняется параллельность прямых, а углы изменяются; расстояния же можно измерить вдоль каждой из главных координатных осей (в общем случае с различными масштабными коэффициентами).



увеличить изображение
Рис. 7.4. Изометрическая проекция

Аксонометрические проекции подразделяются на три группы в соответствии с расположением проекционной плоскости по отношению к осям координат. Если нормаль к проекционной плоскости образует три различных угла с осями, то проекция называется **триметрической** (**триметрией**). Если два из этих углов одинаковы, то получаем **диметрическую** проекцию (**диметрию**). И, наконец, если все три угла равны между собой, то проекция называется **изометрической** (**изометрией**). Изометрическая проекция обладает тем свойством, что все три главные координатные оси одинаково укорачиваются. Поэтому можно проводить измерения вдоль направления осей с одним и тем же масштабом (отсюда название: "изо", что означает "равно", и "метрия" - "измерение"). Кроме того, главные координатные оси проецируются так, что их проекции составляют равные углы друг с другом (рис. 7.4).

Косоугольные проекции также являются параллельными, причем проекционная плоскость перпендикулярна главной координатной оси. Сторона объекта, параллельная этой плоскости, проецируется так, что можно измерять углы и расстояния. Проецирование других сторон объекта также допускает проведение линейных измерений (но не угловых) вдоль главных осей. Мы коснемся только двух наиболее часто используемых косоугольных проекций: проекции **кавалье** (**cavalier**) и **кабине** (**cabinet**). В отечественной практике эти проекции называют **горизонтальной косоугольной изометрией** и **кабинетной проекцией**.

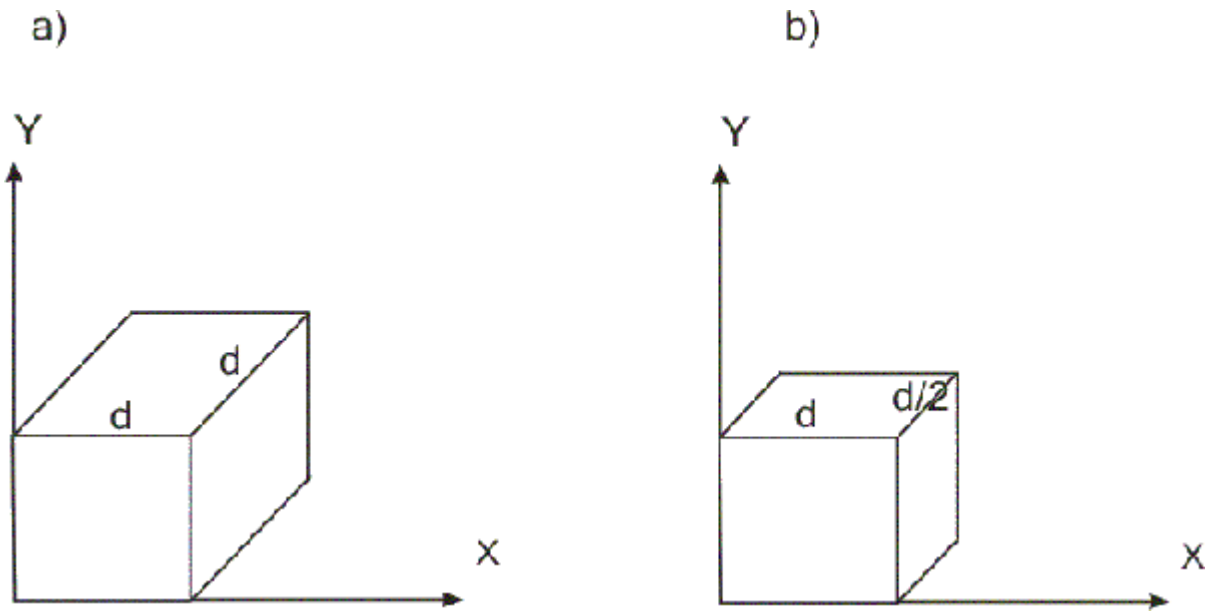


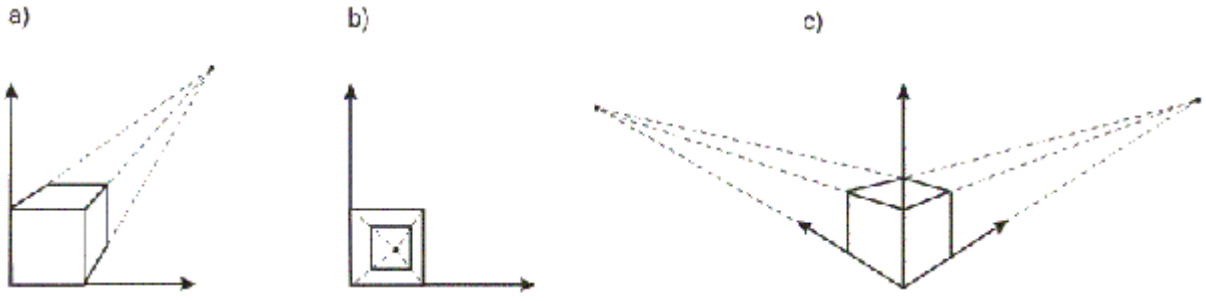
Рис. 7.5. Горизонтальная косоугольная изометрия (а) и кабинетная проекция (б)

В проекции горизонтальной косоугольной изометрии направление проецирования составляет с плоскостью угол 45 $\deg$. В результате проекция отрезка, перпендикулярного проекционной плоскости, имеет ту же длину, что и сам отрезок, т.е. укорачивание отсутствует (рис. 7.5а). Кабинетная проекция имеет направление проецирования, которое составляет с проекционной плоскостью угол $\operatorname{arctg}(1/2)$. При этом отрезки, перпендикулярные проекционной плоскости, после проецирования составляют 1/2 их действительной длины, что более соответствует нашему визуальному опыту, поэтому изображение выглядит более реалистично (рис. 7.5b).

Центральные проекции

Когда пучок проекторов исходит из заданного центра проекции, то параллельные отрезки на плоскости проекции уже не будут параллельными, за исключением случая, когда они лежат в плоскости, параллельной проекционной. При проецировании нескольких параллельных прямых их проекции пересекаются в так называемой **точке схода**. Если совокупность прямых параллельна одной из координатных осей, то их точка схода называется **главной**. Таких точек может быть не больше трех. Например, если проекционная плоскость перпендикулярна оси OZ, то лишь на этой оси будет лежать главная точка схода, поскольку прямые, параллельные как оси OX, так и OY, параллельны также и проекционной плоскости и поэтому не имеют точки схода.

Центральные проекции классифицируются в зависимости от числа главных точек схода, которыми они обладают, а следовательно, и от числа координатных осей, которые пересекает проекционная плоскость. На рис. 7.6 приведены три различные одноточечные проекции куба, причем две из них имеют одну точку схода, а третья - две точки.



увеличить изображение
Рис. 7.6. Одноточечные и двухточечная проекции

Двухточечная центральная проекция широко применяется в архитектурном, инженерном и промышленном проектировании и в рекламных изображениях, в которых вертикальные прямые проецируются как параллельные и, следовательно, не сходятся. Трехточечные центральные проекции почти совсем не используются, во-первых, потому, что их трудно конструировать, а во- вторых, из-за того, что они добавляют мало нового с точки зрения реалистичности по сравнению с двухточечной проекцией.

Математический аппарат

Для выполнения проективных преобразований будем использовать однородные координаты и матрицы преобразований, рассмотренные ранее в лекции 4. Проекция выполняется в системе координат наблюдателя.

Ортогональные проекции

Сначала рассмотрим математическое описание параллельных проекций как более простых. Случай, когда картинная плоскость перпендикулярна оси OZ и задается уравнением $z = 0$(т.е. ортографическая проекция), фактически уже рассматривался в лекции 4, где был приведен вид матриц проекции на координатные плоскости.

Случай аксонометрической проекции сводится к последовательности преобразований, подобно тому как осуществлялся поворот в пространстве относительно произвольной оси. Пусть плоскость задается единичным вектором нормали $\overrightarrow{n} = (n_x, n_y, n_z)$ и расстоянием от начала координат $d \geq 0$. Каноническое уравнение плоскости, таким образом, имеет вид

$$n_x \cdot x + n_y \cdot y + n_z \cdot z - d = 0.$$

Вектор, направленный по нормали от начала координат до пересечения с плоскостью, есть

$$\overrightarrow{N} = d \cdot \overrightarrow{n} = (n_x d, n_y d, n_z d) = (N_x, N_y, N_z).$$

Координаты вектора единичной нормали являются ее направляющими косинусами.

Проецирование в пространстве однородных координат осуществляется следующей последовательностью шагов.

- Сдвиг на вектор $-\overrightarrow{N}$ с помощью матрицы

$$S_1 = \begin{pmatrix} 1 & 0 & 0 & -N_x \\ 0 & 1 & 0 & -N_y \\ 0 & 0 & 1 & -N_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- Поворот, совмещающий направление нормали с направлением оси OZ. Как было показано в лекции 4, этот поворот можно реализовать в виде двух поворотов: первый (относительно оси OZ) переводит нормаль в плоскость

YOZ, а затем - поворот относительно оси OY до совмещения нормали с осью OZ. Соответствующую матрицу вращения, являющуюся произведением двух матриц, обозначим R.

- Проекция на плоскость XOY с помощью матрицы

$$P_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- Поворот с помощью матрицы R^{-1}.
- Сдвиг на вектор $\overrightarrow{N}$ с помощью матрицы S_1^{-1}

Полное преобразование, таким образом, определяется матрицей

$$\mathrm{Pr} = S_1^{-1} \cdot R^{-1} \cdot P_{xy} \cdot R \cdot S_1.$$

Косоугольные проекции

Рассмотрим косоугольную проекцию на плоскость XOY, при которой орт $\overrightarrow{e}_3 = (0,0,1)$ переходит в вектор $\overrightarrow{r}_0 = (a,b,0)$, т.е. направление проекции задается вектором $\overrightarrow{p} = \overrightarrow{r}_0 - \overrightarrow{e}_3 = (a,b,-1)$. Такое преобразование в пространстве однородных координат можно задать с помощью матрицы

$$P = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

В проекции кавалье вектор $\overrightarrow{e}_3$ переходит в вектор $(\cos(\pi/4), \cos(\pi/4), 0)$, а в кабинетной проекции - в вектор $(0.5 \cdot \cos(\pi/4), 0.5\cos(\pi/4), 0)$, причем в обеих проекциях $a = b$.

Центральные проекции

Предположим, что центр проекции находится в точке $\overrightarrow{c} = (c_x, c_y, c_z)$, а картинная плоскость совпадает с плоскостью XOY. Возьмем произвольную точку изображаемого объекта $\overrightarrow{M} = (x,y,z)$ и определим ее проекцию на выбранную плоскость (рис. 7.7).

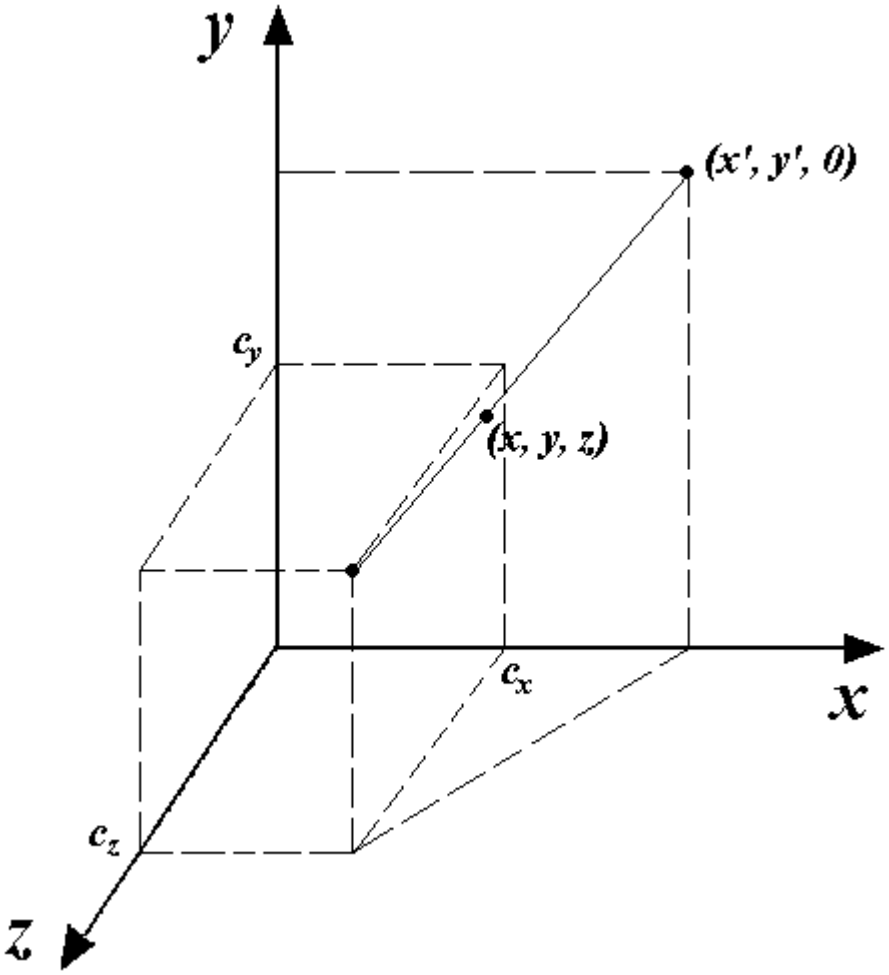


Рис. 7.7. Центральная проекция на плоскость XOY

Прямую, проходящую через точки $\overrightarrow{c}$ и $\overrightarrow{M}$, зададим в параметрическом виде:

$$\overrightarrow{r}(t) = \overrightarrow{c} + t \cdot (\overrightarrow{M} - \overrightarrow{c}) = (c_x + t \cdot (x - c_x), c_y + t \cdot (y - c_y), c_z + \\ + t \cdot (z - c_z)). \quad (7.1)$$

Теперь найдем точку пересечения этой прямой с картинной плоскостью. Она определяется из условия равенства нулю третьей координаты:

$$c_z + t \cdot (z - c_z) = 0,$$

откуда определяем значение параметра t, при котором точка прямой принадлежит координатной плоскости:

$$t^* = \frac{c_z}{c_z - z} = \frac{1}{1 - \frac{z}{c_z}}.$$

Подставляя это значение в формулу (7.1), мы получим координаты проекции точки $\overrightarrow{M}$:

$$x^* = c_x + \frac{x - c_x}{1 - \frac{z}{c_z}}, \quad y^* = c_y + \frac{y - c_y}{1 - \frac{z}{c_z}}. \quad (7.2)$$

Фактором, влияющим на перспективное изменение размеров, является наличие координаты z в знаменателе. Чем ближе оказывается точка к центру проекции, тем больше знаменатель, а соответственно и координаты точки.

Мы будем рассматривать ситуацию, когда центр проекции лежит на оси OZ, а сама ось направлена от наблюдателя к проекционной плоскости, т.е. $c_x = x_y = 0, \quad c_z = -d, \; d > 0$. Тогда формулы (7.2) приобретают вид

$$x^* = \frac{x}{1 + \frac{z}{d}}, \quad y^* = \frac{y}{1 + \frac{z}{d}}. \quad (7.3)$$

В однородных координатах такое преобразование можно записать с помощью двух операций. Сначала умножаем матрицу проективного преобразования P_z на исходную точку и получаем точку в четырехмерном пространстве:

$$P_z \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 + z/d \end{pmatrix}. \quad (7.4)$$

Затем проецируем эту точку в пространство однородных координат путем деления на четвертую компоненту:

$$(x^*, y^*, 0, 1) = (x/p, y/p, 0, 1), \quad p = 1 + z/d.$$

Посмотрим теперь, что происходит с пучком параллельных прямых под действием матрицы проекции. Пусть задан пучок прямых, параллельных вектору $\overrightarrow{v} = (a, b, c)$. Тогда параметрическое уравнение прямой, принадлежащей этому пучку, имеет вид

$$\overrightarrow{r} = (x + at, y + bt, z + ct).$$

Из формулы (7.4) следует, что в результате проецирования получим множество точек

$$\left(x + at, y + bt, 0, 1 + \frac{z + ct}{d} \right).$$

Переходя к однородным координатам и умножив числитель и знаменатель каждой дроби на d, получим точки $\overrightarrow{r}_0$ вида

$$\overrightarrow{r}_0 = \left(d\frac{x + at}{d + z + ct}, d\frac{y + bt}{d + z + ct}, 0, 1 \right).$$

Теперь в каждой компоненте вектора числитель и знаменатель поделим на t:

$$\overrightarrow{r}_0 = \left(d\frac{x/t + a}{(d + z)/t + c}, d\frac{y/t + b}{(d + z)/t + c}, 0, 1 \right).$$

Переходя к пределу при $t \to \infty$, получим точку

$$\overrightarrow{r}_\infty = \left(\frac{da}{c}, \frac{db}{c}, 0, 1\right).$$

Таким образом, получаем, что после проецирования пучок параллельных прямых пересекается в точке схода $\overrightarrow{r}_\infty$. Понятно, что у каждого пучка своя точка схода. Если пучок прямых параллелен плоскости XOY, т.е. $c = 0$, то точка схода оказывается на бесконечности, а значит, прямые остаются параллельными.

Для построения перспективной проекции с несколькими точками схода используется матрица перспективного преобразования без проецирования:

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/d_1 & 1/d_2 & 1/d_3 & 1 \end{pmatrix}.$$

Теперь точки пространства сначала подвергаются перспективному преобразованию, а затем осуществляется проекция.

Определим точки схода для прямых, параллельных осям координат. Для прямых $\overrightarrow{r} = (x, y, z + ct)$ результатом проективного преобразования будет множество точек $(x/f, y/f, (z + ct)/f, 1)$, где $f = x/d_1 + y/d_2 + (z + ct)/d_3 + 1$. При $t \to \infty$ получим точку с координатами $(0, 0, d_3, 1)$. При проекции на плоскость XOY получим точку $(0, 0)$. Пучок прямых $\overrightarrow{r} = (x + ct, y, z)$ перейдет в $((x + ct)/f_1, y/f_1, z/f_1, 1),\ f_1 = (x + ct)/d_1 + y/d_2 + z/d_3 + 1$, а точкой схода для него будет $(d_1, 0, 0, 1)$, которая при проецировании перейдет в точку, лежащую на оси OX $(d_1, 0)$. Аналогично для пучка прямых, параллельных оси OY, получим точку схода на оси OY $(0, d_2)$. Эти три точки на плоскости являются главными точками схода.

Специальные картографические проекции. Экзотические проекции земной сферы

С развитием торговли и путешествий приобрела большую важность довольно непростая геометрическая задача: как перенести на плоскость часть земной поверхности, чтобы расстояния между любыми двумя точками на ней остались неискаженными? Различные ученые в течение многих веков пытались разрешить эту проблему по заказам своих правительств, крупных коммерсантов или просто путешественников, для которых такая карта была бы настоящей находкой, так как существенно облегчила бы навигацию, как морскую, так и воздушную.

В целом эта задача оказалась неразрешимой. Поверхность цилиндра или конуса можно без искажений перенести на плоскость (такие поверхности называются **развертывающимися**), отобразить же на плоскость поверхность сферы, сохранив расстояние между любыми двумя точками, невозможно. Дело в том, что даже малую область сферической поверхности (в отличие от цилиндра или конуса) невозможно развернуть на плоскости без трещин, складок или искажений. Любая плоская карта Земли или какой-то ее части непременно будет искажать какие-либо свойства. Поэтому в настоящее время так необходимы карты с минимальными (еще лучше нулевыми) искажениями тех свойств, для передачи которых предназначается карта. Желательно,

чтобы и другие свойства деформировались как можно меньше. Всего существует четыре основных типа искажений:

- искажение длин (линии, одинаковые на поверхности Земли, изображаются на карте отрезками разной длины);
- искажение углов (углы на карте между взятыми направлениями не равны горизонтальным углам между теми же направлениями на поверхности земного эллипсоида);
- искажение форм (форма участка или занятой объектом территории на карте отлична от их формы на поверхности Земли);
- искажение площадей (связано с масштабом площади: при постоянстве величины масштаба площади по всей поверхности карты искажения площадей на ней нет).

Помимо классических карт, большая часть которых была разработана в средние века, фантазия ученых предоставляла навигаторам весьма необычные способы проекции земной поверхности, но прежде чем описать способы составления необычных карт, рассмотрим некоторые классические методы картографии.

Центр проекции может быть произвольным по отношению к проецируемой сфере; таким образом, существует бесконечное множество всевозможных различных проекций. Если проводить лучи из некоторой точки, взятой на прямой, проходящей через центр шара перпендикулярно некоторой плоскости, то получим на этой плоскости перспективную проекцию. Рассмотрим некоторые из этих проекций, наиболее полезные с точки зрения картографии.

Стереографическая проекция

Важное свойство любой карты - сохранение углов (угол между любыми двумя линиями на карте должен быть таким же, как угол между прообразами этих линий на земной поверхности). Сохранение углов особенно важно для мореплавания и аэронавтики, так как оно означает, что наблюдаемый угол между любыми двумя ориентирами равен углу, измеряемому на карте с помощью транспортира. Кроме того, на такой карте остаются неизменными и площади малых областей. Карты, сохраняющие углы, называются **конформными**. Проще всего построить конформную карту с помощью **стереографической** проекции.

На рис. 7.8 показано, как поверхность сферы в точке X проецируется из точки A (принадлежащей сфере) на плоскость, касательную к сфере в диаметрально противоположной точке (**антипод** точки A). Проекция называется экваториальной, полярной или косой в зависимости от того, где находятся антиподы: на экваторе, полюсах или в какой-нибудь другой точке земной поверхности соответственно. К сожалению, конформность вызывает искажение масштаба, возрастающее с увеличением расстояния от центра карты.

Рис. 7.8. Три проекции

Обозначим за долготу и дополнение до широты точки на сфере буквами λ и θ соответственно, $0 \le \lambda \le 360^\circ,\ 0 \le \theta \le 180^\circ$, а через x и y - координаты проекций этой точки в некоторой декартовой системе координат, заданной на плоскости проекции. Соответствующие формулы проецирования для Северного полушария имеют следующий вид:

$$x = \mathrm{tg}\,\theta/2\cos(\lambda - 135^\circ);$$
$$y = \mathrm{tg}\,\theta/2\sin(\lambda - 135^\circ)$$

Здесь и в дальнейшем радиус считается равным единице. Ось X направлена вдоль меридиана 135 deg.

Гномоническая проекция

Отображение точки Xиз центра земного шара Bна плоскость карты в точку B' порождает **гномоническую** проекцию (рис. 7.8). Проекция получила такое название, так как она напоминает конструкцию солнечных часов с гномоном. Любая дуга большого круга на поверхности земного шара переходит в прямую на гномонической карте. **Большим кругом** называется окружность на сфере, плоскость которой проходит через ее центр. Такая карта не обладает конформностью, но навигаторы ценят ее за одно важное свойство, отсутствующее у всех других проекций сферы на плоскость: прямая между любыми двумя точками на гномонической карте является геодезической, или кратчайшей дугой между этими двумя точками, и соответствует дуге большого круга на поверхности Земли.

Ортографическая проекция

Если центр проекции находится в бесконечности (все проецирующие лучи параллельны), то проекция будет ортографической (рис. 7.8). Например, глядя на Луну с Земли, наблюдатель видит Луну практически в ортографической проекции. У края ортографической карты расстояния сильно искажены. Ортографическая карта не сохраняет ни площадей, ни углов, но, выполненная достаточно искусно, создает сильную иллюзию шарообразной Земли. Карты, начерченные с точки зрения наблюдателя, находящегося над земной поверхностью, не точны в передаче многих ее свойств, но наиболее верно соответствуют нашему зрительному восприятию сферы.

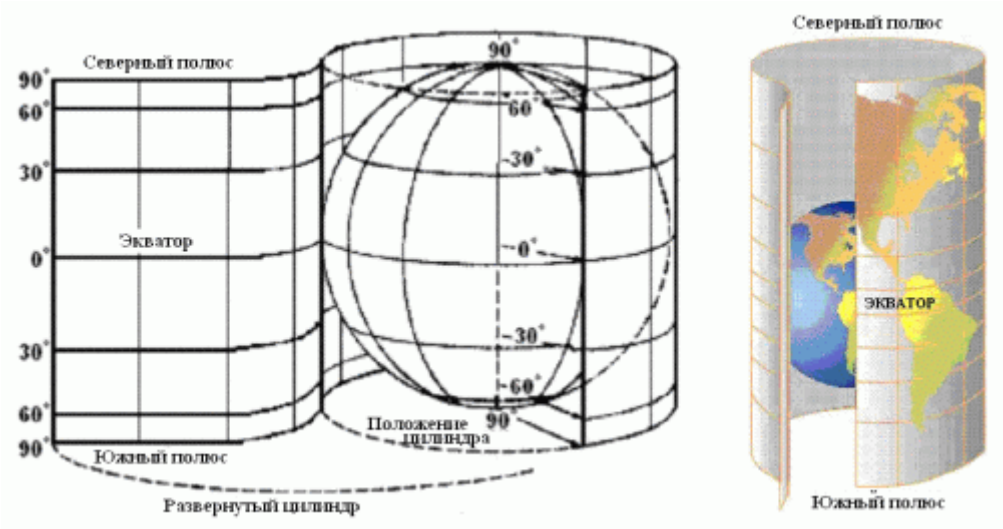
Эта проекция получается при проецировании на плоскость, касательную к сфере в центре изображаемого явления (λ_0, θ_0), с помощью лучей, перпендикулярных этой плоскости. Формулы этой проекции следующие:

$$x = \sin\theta \sin(\lambda - \lambda_0)$$
$$y = \sin\theta_0 \cos\theta - \cos\theta_0 \sin\theta \cos(\lambda - lambda_0)$$

Проекции на цилиндр

Поверхность сферы также можно проецировать на цилиндры и конусы, "надетые" на сферу. После построения цилиндрической или конической проекции поверхность разрезается и развертывается на плоскость.

Лучи, проецирующие земной шар на цилиндр, выбираются так, чтобы они были параллельны плоскости, высекающей окружность, по которой сфера и цилиндр соприкасаются (рис. 7.9). Если цилиндр касается Земли вдоль экватора, то все меридианы и параллели на карте переходят в прямые, пересекающиеся под прямыми углами.



увеличить изображение

Рис. 7.9. Метод цилиндрической проекции с сохранением площадей

Цилиндрическая карта не всегда обладает конформностью и может сильно искажать расстояния и форму областей. Отметим, что ни одна карта не может одновременно быть конформной и сохранять площади. Было предложено огромное число других проекций, сохраняющих площадь; в современных атласах чаще всего встречаются сохраняющие площади карты, построенные с помощью цилиндрической проекции, которая была предложена Карлом Б.Мольвейде в 1805 г.

Проекция Меркатора

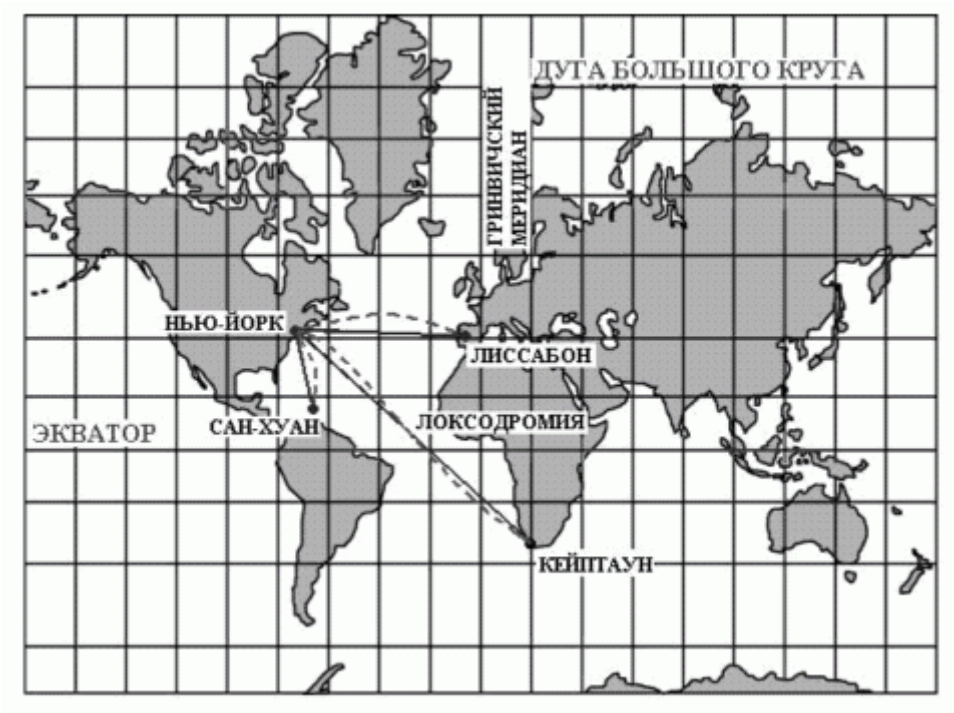
В XVI веке фламандский картограф Герхард Меркатор создал знаменитую цилиндрическую проекцию, обладающую свойством конформности. Конформность в проекции Меркатора достигается за счет растягивания цилиндра за полюсы, при этом в верхней и нижней части этого цилиндра масштаб становится очень искаженным. Несмотря на это данная проекция обладает одним замечательным свойством, очень нужным для навигаторов: прямая, проведенная через любые две точки на карте, является локсодромой, или линией постоянного румба. Локсодрома на сфере или

какой-либо другой поверхности вращения пересекает все меридианы под постоянным углом (рис. 7.10).

Проекция Меркатора задается следующими формулами:

$$x = \begin{cases} \lambda/180^\circ \text{ при } 0^\circ \le \lambda \le 180^\circ, \\ \lambda/180^\circ - 2 \text{ при } 180^\circ \le \lambda \le 360^\circ \end{cases}$$

$$y = \ln(\mathrm{tg}(90^\circ - 0.5\theta))/\pi.$$

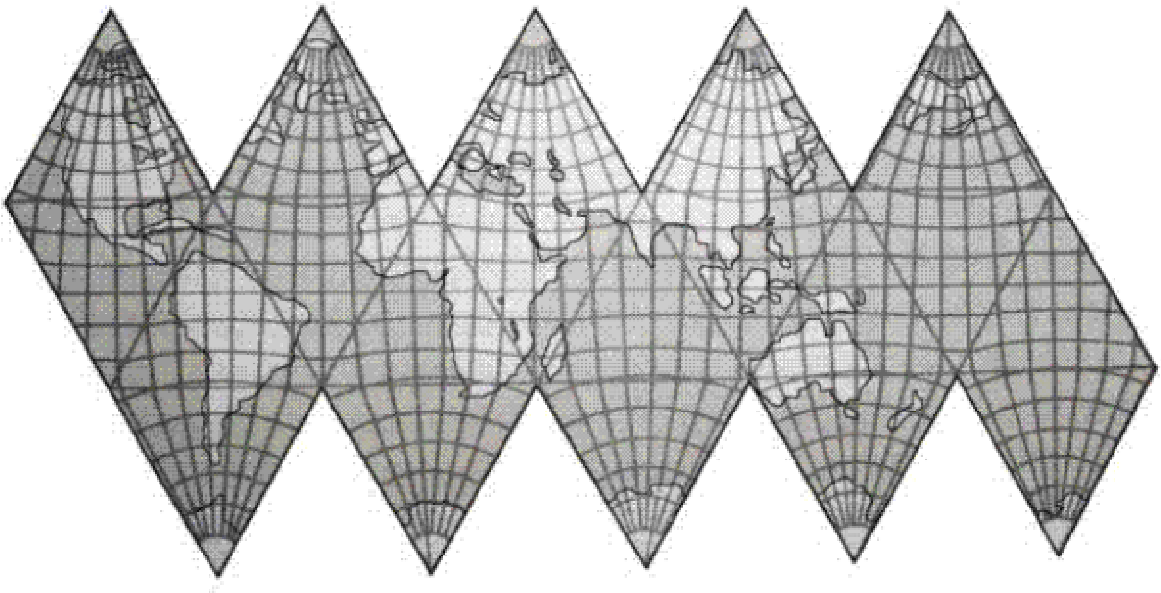


увеличить изображение

Рис. 7.10. Конформная проекция Меркатора. На карту нанесены локсодромы из Нью-Йорка

Проекции на многогранник

Будем называть **разрезанной** карту мира, спроецированную на тот или иной узор из каких-либо многоугольников. После складывания этих фрагментов образуется карта с разрывами любой части земного шара. Одну такую конформную карту составил философ и математик Ч.Пирс. Земная поверхность спроецирована на этой карте на восемь равнобедренных треугольников, которые можно рассматривать как грани октаэдра, сплющиваемого до тех пор, пока длина его пространственной диагонали не обратится в нуль. Вершинам нулевой диагонали на карте Пирса соответствуют северный и южный полюсы.



увеличить изображение
Рис. 7.11. "Линкаглобус" И. Фишера, складывающийся в икосаэдр

Примерно в то же время аналогичная идея пришла в голову выдающемуся экономисту из Йельского университета Ирвингу Фишеру: он задумал осуществить гномоническую проекцию поверхности Земли на 20 треугольных гранях икосаэдра (рис. 7.11). Икосаэдр является наиболее близким к идеалу многогранником для разрезания "на карты".

Необычные проекции

Картографы придумывали самые разнообразные виды проекций, порой удивительно выглядящие и при этом обладающие довольно неплохими свойствами. Одну из таких карт в форме кардиоиды (сердца) придумал Иоганн Вернер. Эта карта сохраняет площади. Она пользовалась широкой известностью в XVI в., но сейчас ею уже давно не пользуются. В отчете о картографических курьезах, написанном для внутреннего пользования фирмы "Лаборатории Белла", математик Эдгар Н. Гильберт пишет: "...незаслуженно забыта. Сильно искаженные части карты лежат далеко от основных масс суши. Искривленные параллели придают карте приятную иллюзию округлости... Параллели представляют собой дуги окружностей, расположенные на одинаковом расстоянии друг от друга, с центром в северном полюсе. Меридианы, проведенные для указа¬ния расстояний вдоль параллелей, такие же как на сфере".

Скопление материков на карте Вернера отражает неравномерное распределение суши по поверхности Земли. Тихий океан столь велик, что если смотреть на Землю из точки, расположенной над проливом Ла-Манш, то взгляду откроется около 80% всей суши, а противопо¬ложное полушарие будет почти сплошь покрыто водой

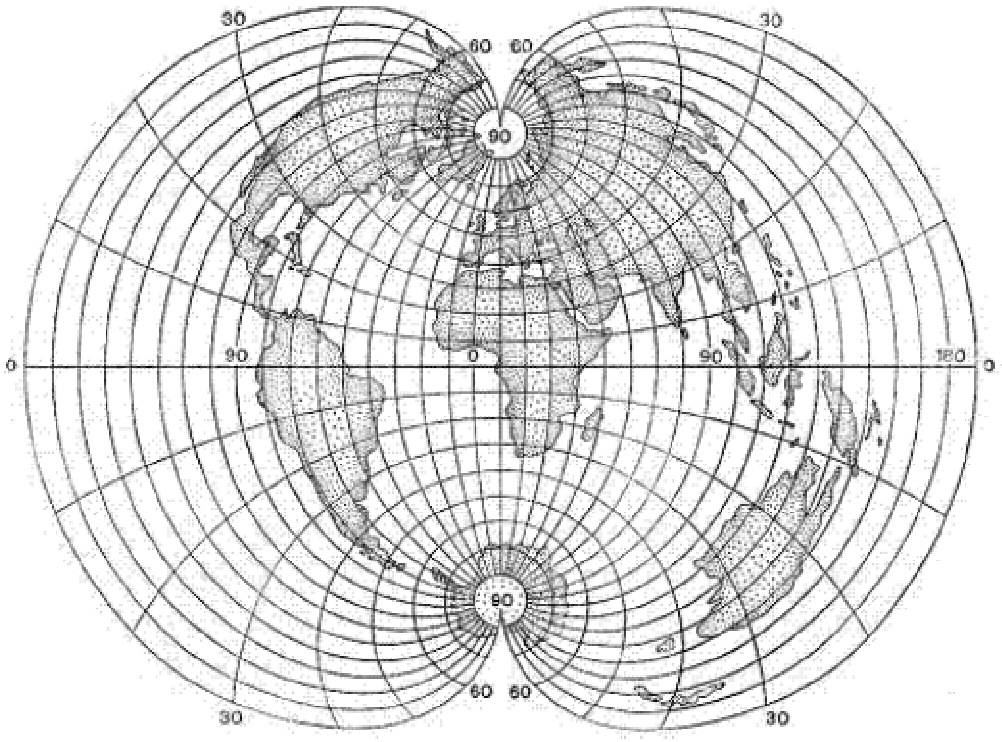


Рис. 7.12. Поликоническая картографическая проекция

Довольно необычна поликоническая картографическая проекция - картографическая проекция, строящаяся с помощью ряда конусов, касательных к земному эллипсоиду (шару). В поликонической картографической проекции параллели нормальной сетки - дуги эксцентрических окружностей, осевой меридиан - прямая, на которой расположены центры параллелей, а остальные меридианы - кривые (рис. 7.12).

Вопросы и упражнения

1. Назовите два основных вида проекций, определяемых типом пучка лучей.
2. Назовите четыре вида параллельных проекций.
3. Сколько шагов в алгоритме ортогональной проекции на произвольную плоскость?
4. Какой вид имеет матрица косоугольной проекции на плоскость XOY, переводящей вектор $(0,0,1)$ в вектор $(x,y,0)$?
5. Напишите формулы преобразования координат при центральной проекции на плоскость XOY с центром в точке $(0,0,c)$. Как выглядит матрица такой проекции в однородной системе координат?
6. Что такое перспективное укорачивание?
7. Что такое точка схода?
8. Как реализуется проекция с тремя точками схода?
9. Каким свойством обладает конформная проекция?
10. Каким свойством обладает цилиндрическая проекция?
11. В чем ценность проекции Меркатора?
12. Какой многогранник наиболее удобен для построения разрезанных карт?

Лекция 8. Растровое преобразование графических примитивов

Алгоритмы Брезенхема растровой дискретизации отрезка. Алгоритмы Брезенхема растровой дискретизации окружности и эллипса. Алгоритмы заполнения внутренних областей

Экран растрового дисплея можно рассматривать как матрицу дискретных элементов, или пикселей. Процесс определения пикселей, наилучшим образом аппроксимирующих некоторую геометрическую фигуру, называется разложением в растр, или построением растрового образа фигуры. Построчная визуализация растрового образа называется растровой разверткой данной фигуры.

Алгоритм Брезенхема растровой дискретизации отрезка

При построении растрового образа отрезка необходимо, прежде всего, установить критерии "хорошей" аппроксимации. Первое требование состоит в том, что отрезок должен начинаться и кончаться в заданных точках и при этом выглядеть сплошным и прямым (при достаточно высоком разрешении дисплея этого можно добиться). Кроме того, яркость вдоль отрезка должна быть одинаковой и не зависеть от наклона отрезка и его длины. Это требование выполнить сложнее, поскольку горизонтальные и вертикальные отрезки всегда будут ярче наклонных, а постоянная яркость вдоль отрезка опять же достигается на вертикальных, горизонтальных и наклоненных под углом в 45 $\deg$ линиях. И, наконец, алгоритм должен работать быстро. Для этого необходимо по возможности исключить операции с вещественными числами. С целью ускорения работы алгоритма можно также реализовать его на аппаратном уровне.

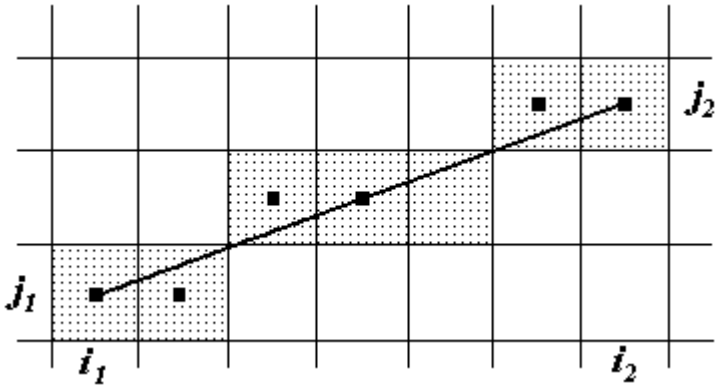


Рис. 8.1. Растровый образ отрезка

В большинстве алгоритмов используется пошаговый метод изображения, т.е. для нахождения координат очередной точки растрового образа наращивается значение одной из координат на единицу растра и вычисляется приращение другой координаты.

Задача состоит в построении отрезка, соединяющего на экране точки с координатами $(i_1, j_1), (i_2, j_2)$ (будем считать, что $i_1 \neq i_2$). Для построения отрезка прямой на плоскости с вещественными координатами можно воспользоваться уравнением прямой, проходящей через две заданные точки, которое имеет вид

$$j = j_1 + k(i - i_1), \quad k = (j_2 - j_1)/(i_2 - i_1).$$

Теперь, считая, что $0 \leq k \leq 1$, (i, j) - координаты текущей точки растрового образа, а y - точное значение координаты точки отрезка, можно построить следующую точку:

$$i' = i + 1, \quad j' = y + k$$

Следует заметить, что целочисленная координата j изменится только в том случае, если y превысит величину $j+0.5$(j' есть ближайшее к y целое число, полученное в результате операции округления). Приведенный пример включает операции с вещественными числами, которые выполняются существенно медленнее, чем соответствующие целочисленные операции, а при построении растрового образа отрезка желателен алгоритм, по возможности обращающийся только к целочисленной арифметике. Кроме того, алгоритм должен работать при любом взаимном расположении концов отрезка.

Алгоритм Брезенхема построения растрового образа отрезка был изначально разработан для графопостроителей, но он полностью подходит и для растровых дисплеев. В процессе работы в зависимости от углового коэффициента отрезка наращивается на единицу либо i , либо j , а изменение другой координаты зависит от расстояния между действительным положением точки и ближайшей точкой растра (смещения). Алгоритм построен так, что анализируется лишь знак этого смещения.

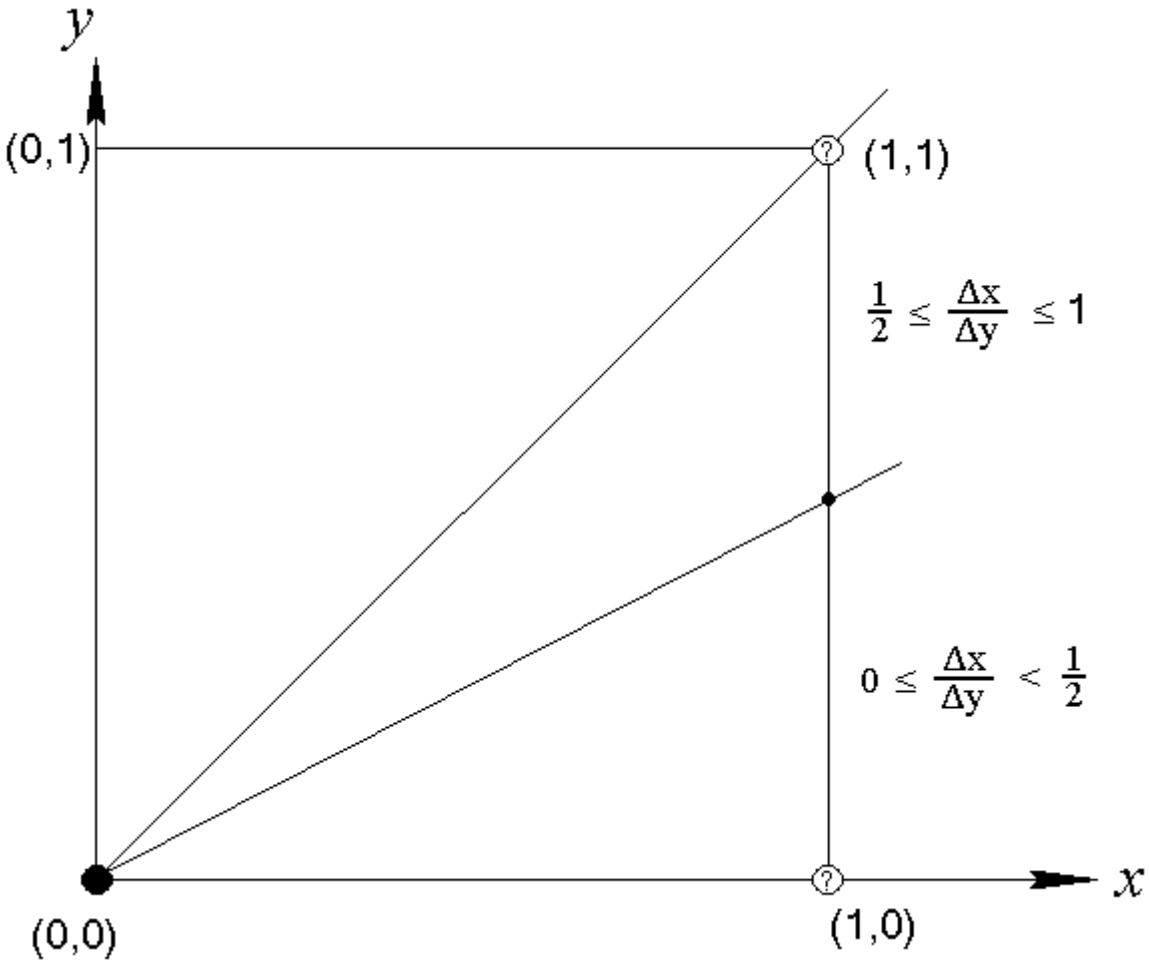


Рис. 8.2. Связь углового коэффициента с выбором пикселя

На рис. 8.2 это иллюстрируется для отрезка с угловым коэффициентом, лежащим в диапазоне от нуля до единицы. Из рисунка можно заметить, что если угловой коэффициент $k \ge \frac{1}{2}$, то при выходе из точки $(0,0)$пересечение с прямой $x=1$будет ближе к прямой $y=1$, чем к прямой $y=0$. Следовательно, точка растра $(1,1)$лучше аппроксимирует прохождение отрезка, чем точка $(1,0)$. При $k < \frac{1}{2}$верно обратное.

На рис. 8.3 показано, каким образом строятся точки растра для отрезка с тангенсом угла наклона $3/8$, а на рис. 8.4 - график смещения. В начале построения смещение полагается равным $k - 1/2$, а затем на каждом шаге оно наращивается на величину k, и если при этом вертикальная координата точки растра увеличивается на единицу, то смещение в свою очередь уменьшается на единицу.

На рис. 8.5 приведена блок-схема алгоритма для случая $i_2 > i_1, \quad j_2 > j_1, \quad k > 0$. Нетрудно понять, как от этого алгоритма перейти к целочисленному: достаточно вместо величины смещения e перейти к величине $\tilde{e} = 2\Delta i \cdot e$.

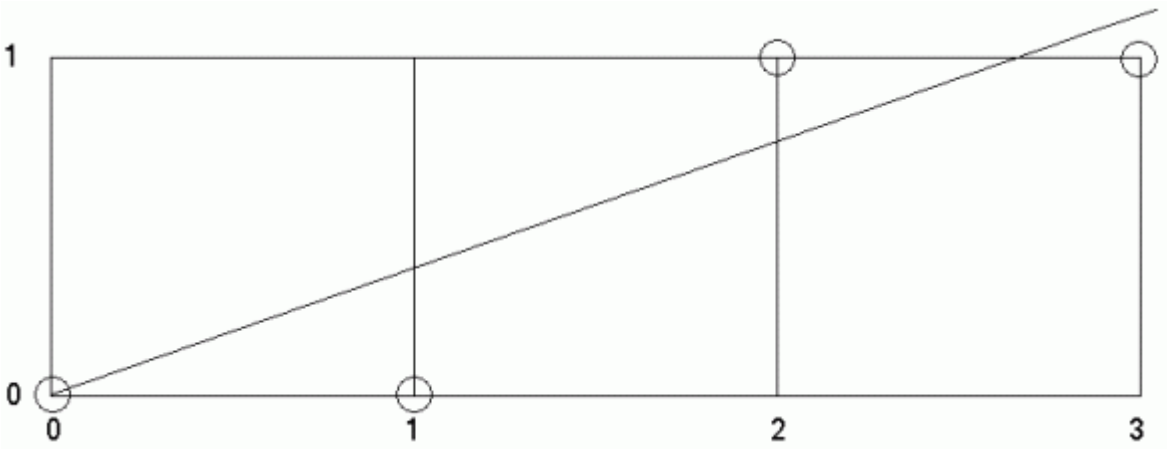


Рис. 8.3. Пиксели, принадлежащие развертке отрезка

Рис. 8.4. График изменения отклонения

Приведем общий алгоритм Брезенхема, который учитывает все возможные случаи направления отрезка, рассматриваемого как вектор на координатной плоскости (на рис. 8.6 выделены четыре области и указаны особенности алгоритма в каждой из них).

В описании алгоритма используются следующие функции:

```
swap (a, b): обмен значений переменных a, b;
abs (a): абсолютное значение a;
sign (a): 0, если a= 0, 1, если a>0, -1, если a<0;
point (i, j) - инициализация точки (i, j).
```

Предполагается, что концы отрезка $(i_1, j_1), (i_2, j_2)$ не совпадают и что все используемые переменные являются целыми.

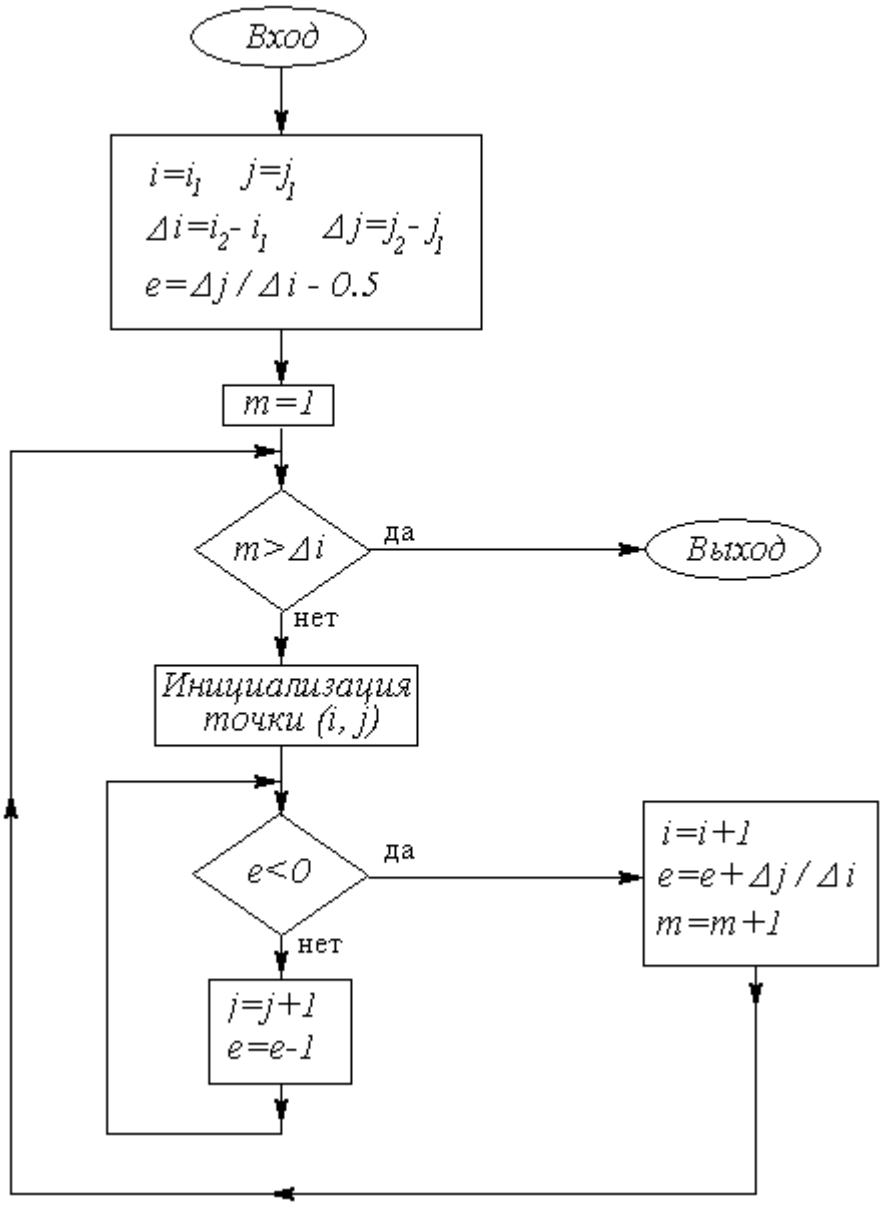


Рис. 8.5. Блок-схема одной ветви алгоритма Брезенхема

```
i=i1;
j=j1;
di=i2-i1;
dj=j2-j1;
s1=sign(i2-i1);
s2=sign(j2-j1);
di=abs(di);
dj=abs(dj);
if (dj>di)
{
 swap(di,dj); c=1;
}
else c=0;
e=2*dj-di;     // Инициализация смещения
// Основной цикл
for (l=0; l<di; l++)
{
       point(i,j);
       while (e>=0)
       {
              if (c==1) i=i+s1;
```

```
            else j=j+s2;
            e=e-2*di;
        }
        if (c==1) j=j+s2; else i=i+s1;
        e=e+2*dj;
        }
```

I : *увеличение j на 1*
II : *увеличение i на 1*
III : *уменьшение j на 1*
IV : *уменьшение i на 1*

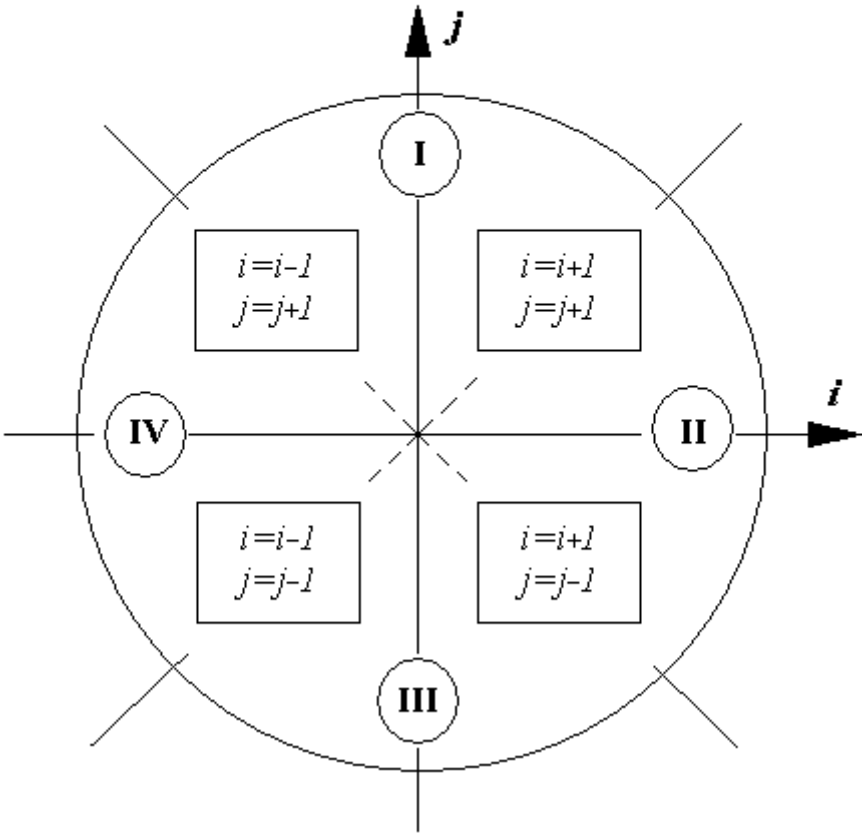


Рис. 8.6. Четыре возможных направления отрезка

Алгоритмы Брезенхема растровой дискретизации окружности и эллипса

Алгоритм изображения окружности несколько сложнее, чем построение отрезка. Мы рассмотрим его для случая окружности радиуса r с центром в начале координат. Перенесение его на случай произвольного центра не составляет труда. При построении растровой развертки окружности можно воспользоваться ее симметрией относительно координатных осей и прямых $y = \pm x$. Необходимо сгенерировать лишь одну восьмую часть окружности, а остальные ее части можно получить путем отображений симметрии. За основу можно взять часть окружности от 0 до 45 $\deg$ в направлении по часовой стрелке с исходной точкой построения $(r, 0)$. В этом случае координата окружности x является монотонно убывающей функцией координаты y.

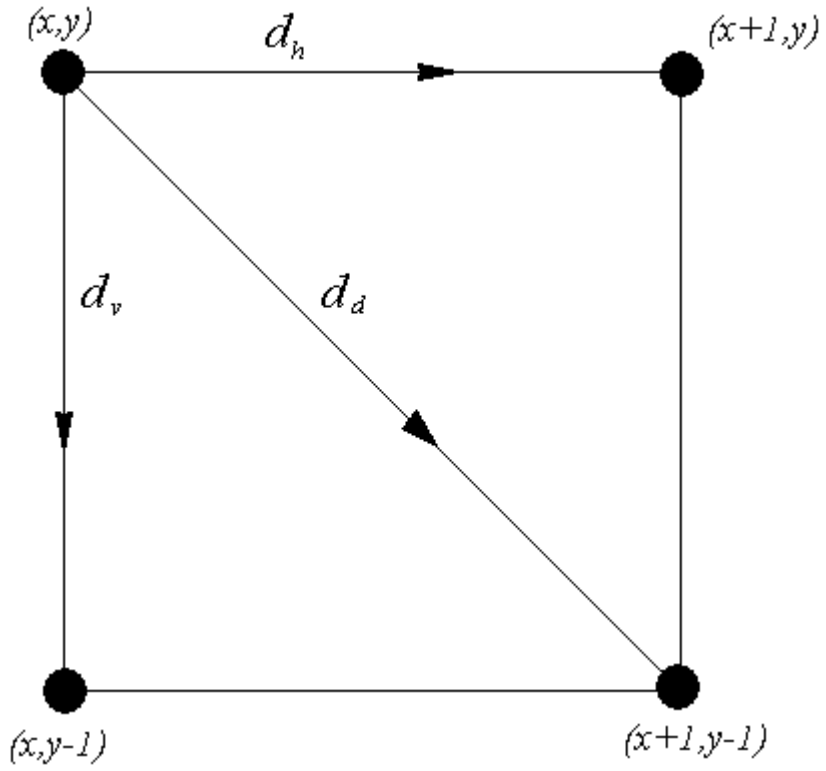


Рис. 8.7. Ближайший пиксель при движении по окружности

При выбранном направлении движения по окружности имеется только три возможности для расположения ближайшего пикселя: на единицу вправо, на единицу вниз и по диагонали вниз (рис. 8.7). Выбор варианта можно осуществить, вычислив расстояния до этих точек и выбрав минимальное из них:

$$d_h = |s_h|, \quad d_v = |s_v|, \quad d_d = |s_d|,$$
$$s_h = (x+1)^2 + y^2 - r^2, \quad s_v = x^2 + (y-1)^2 - r^2,$$
$$s_d = (x+1)^2 + (y-1)^2 - r^2.$$

Алгоритм можно упростить, перейдя к анализу знаков величин s_h, s_v, s_d. При $s_d < 0$ диагональная точка лежит внутри окружности, поэтому ближайшими точками могут быть только диагональная и правая. Теперь достаточно проанализировать знак выражения $\Delta = d_v - d_d$. Если $\Delta \le 0$, выбираем горизонтальный шаг, в противном случае - диагональный. Если же $s_d > 0$, то определяем знак $\Delta^1 = d_d - d_v$, и если $\Delta^1 \le 0$, выбираем диагональный шаг, в противном случае - вертикальный. Затем вычисляется новое значение s_d, причем желательно минимизировать вычисления не только этой величины, но и величин Δ, Δ^1 на каждом шаге алгоритма. Путем несложных преобразований можно получить для первого шага алгоритма, что $\Delta = 2(s_d + y) - 1, \quad \Delta^1 = 2(s_d + x) - 1$.

После перехода в точку $(x', y'),\ x' = x+1,\ y' = y+1$ по диагонали новое значение s_d вычисляется по формуле $s'_d = s_d + 2x' - 2y' + 2$, при горизонтальном переходе $(x' = x+1, y' = y) \quad s'_d = s_d + 2x' + 1$, при вертикальном $(x' = x, y' = y-1)$ - $s'_d = s_d - 2y' + 1$.

Таким образом, алгоритм рисования этой части окружности можно считать полностью описанным (блок-схема его приведена на рис. 8.8). Все оставшиеся ее части строятся параллельно: после получения очередной точки (x', y') можно инициализировать еще семь точек с координатами
$(-x', y'),\ (-x', -y'),\ (x', -y'),\ (y', x'),\ (y', -x'),\ (-y', -x'),\ (-y', x')$.

Для построения растровой развертки эллипса с осями, параллельными осям координат, и радиусами a, b воспользуемся каноническим уравнением

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

которое перепишем в виде

$$f(x, y) \equiv b^2x^2 + a^2y^2 - a^2b^2 = 0.$$

В отличие от окружности, для которой было достаточно построить одну восьмую ее часть, а затем воспользоваться свойствами симметрии, эллипс имеет только две оси симметрии, поэтому придется строить одну четверть всей фигуры. За основу возьмем дугу, лежащую между точками $(0, b)$ и $(a, 0)$ в первом квадранте координатной плоскости.

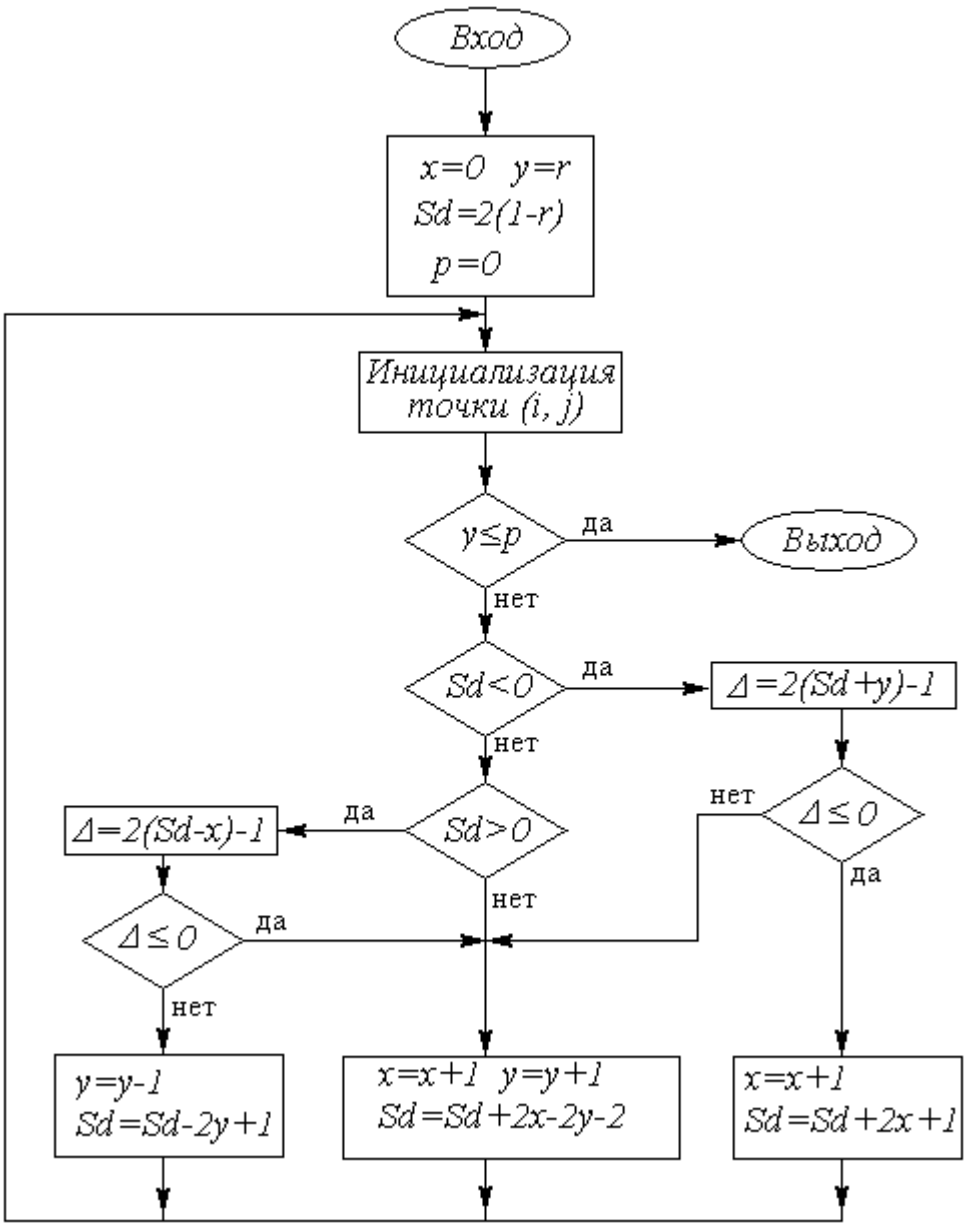


Рис. 8.8. Блок-схема построения восьмой части окружности

В каждой точке (x, y) эллипса существует вектор нормали, задаваемый градиентом функции f. Дугу разобьем на две части: первая - с углом между нормалью и горизонтальной осью больше 45 $\deg$ (тангенс больше 1) и вторая - с углом, меньшим 45 $\deg$ (рис. 8.9). Движение вдоль дуги будем осуществлять в направлении по часовой стрелке, начиная с точки $(0, b)$. Вдоль всей дуги координата является монотонно убывающей функцией от x, но в первой части она убывает медленнее, чем растет аргумент, а во второй - быстрее. Поэтому при построении растрового образа в первой части будем увеличивать x на единицу и искать соответствующее значение y, а во второй - сначала уменьшать значение y на единицу и определять соответствующее значение x.

Направление нормали соответствует вектору

$$grad(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2b^2x, 2a^2y).$$

Отсюда находим тангенс угла наклона вектора нормали: $t = a^2 y / b^2 x$. Приравнивая его единице, получаем, что координаты точки деления дуги на вышеуказанные части удовлетворяют равенству $b^2 x = a^2 y$. Поэтому критерием того, что мы переходим ко второй области в целочисленных координатах, будет соотношение $a^2(y - 1/2) \le b^2(x + 1)$, или, переходя к целочисленным операциям, $a^2(2y - 1) \le 2b^2(x + 1)$.

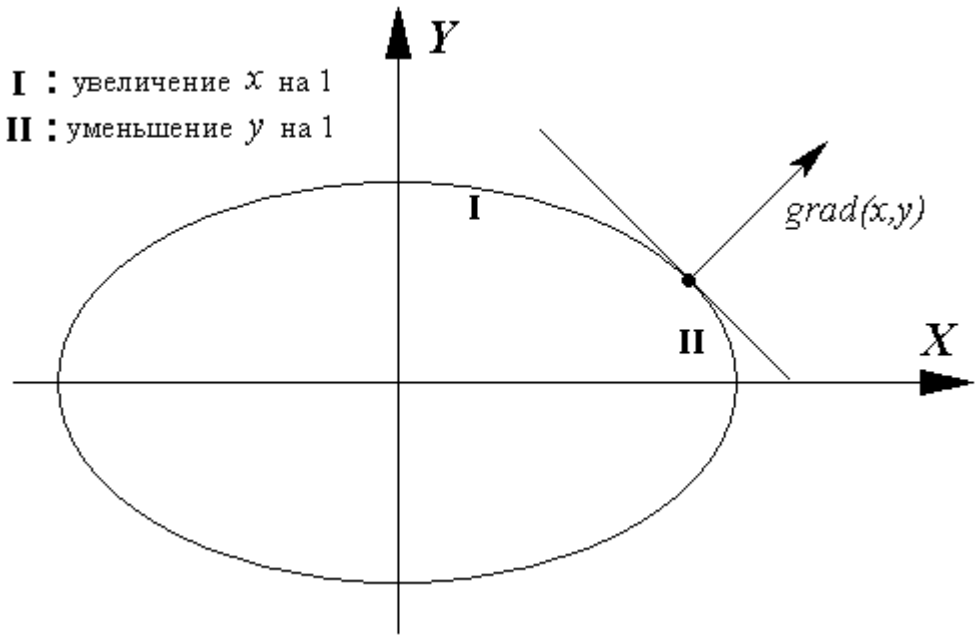


Рис. 8.9. Две области на участке эллипса

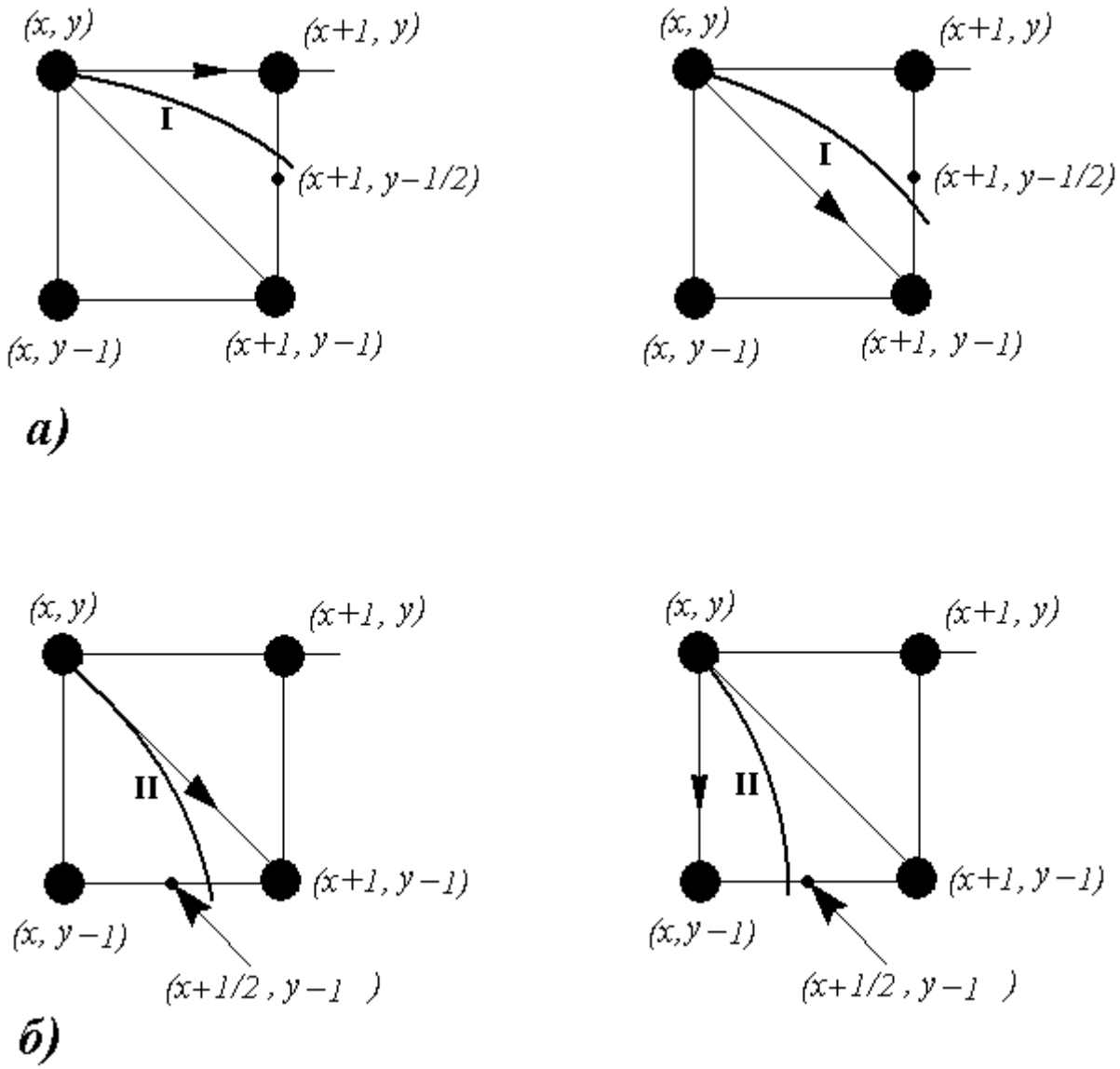


Рис. 8.10. Схема перехода в первой и второй областях дуги эллипса

При перемещении вдоль первого участка дуги мы из каждой точки переходим либо по горизонтали, либо по диагонали, и критерий такого перехода напоминает тот, который использовался при построении растрового образа окружности. Находясь в точке (x, y), мы будем вычислять значение $\Delta = f(x+1, y-\frac{1}{2})$. Если это значение меньше нуля, то дополнительная точка $(x+1, y-\frac{1}{2})$ лежит внутри эллипса, следовательно, ближайшая точка растра есть $(x+1, y)$, в противном случае это точка $(x+1, y-1)$ (рис. 8.10а).

На втором участке дуги возможен переход либо по диагонали, либо по вертикали, поэтому здесь сначала значение координаты y уменьшается на единицу, затем вычисляется $\Delta = f(x+\frac{1}{2}, y-1)$ и направление перехода выбирается аналогично предыдущему случаю (рис. 8.10б).

Остается оптимизировать вычисление параметра Δ, умножив его на 4 и представив в виде функции координат точки. Тогда для первой половины дуги имеем

$$\widetilde{\Delta}(x,y) \equiv 4\Delta(x,y) = 4b^2(x+1)^2 + a^2(2y-1)^2 - 4a^2b^2,$$
$$\widetilde{\Delta}(x+1,y) = \widetilde{\Delta}(x,y) + 4b^2(2x+3),$$
$$\widetilde{\Delta}(x+1,y-1) = \widetilde{\Delta}(x,y) + 4b^2(2x+3) - 8a^2(y-1).$$

Для второй половины дуги получим

$$\widetilde{\Delta}(x,y) \equiv 4\Delta(x,y) = b^2(2x+1)^2 + 4a^2(y-1)^2 - 4a^2b^2,$$
$$\widetilde{\Delta}(x+1,y) = \widetilde{\Delta}(x,y) + 8b^2(x+1),$$
$$\widetilde{\Delta}(x+1,y-1) = \widetilde{\Delta}(x,y) + 8b^2(x+1) - 4a^2(2y+3).$$

Все оставшиеся дуги эллипса строятся параллельно: после получения очередной точки (x', y'), можно инициализировать еще три точки с координатами $(-x', y'),\ (-x', -y'),\ (x', -y')$. Блок-схему не приводим ввиду прозрачности алгоритма.

Алгоритмы заполнения областей

Для заполнения областей, ограниченных замкнутой линией, применяются два основных подхода: затравочное заполнение и растровая развертка.

Методы первого типа исходят из того, что задана некоторая точка (затравка) внутри контура и задан критерий принадлежности точки границе области (например, задан цвет границы). В алгоритмах ищут точки, соседние с затравочной и расположенные внутри контура. Если обнаружена соседняя точка, принадлежащая внутренней области контура, то она становится затравочной и поиск продолжается рекурсивно.

Методы растровой развертки основаны на сканировании строк растра и определении, лежит ли точка внутри заданного контура области. Сканирование осуществляется чаще всего "сверху вниз", а алгоритм определения принадлежности точки заданной области зависит от вида ее границы.

Сначала рассмотрим простой алгоритм заполнения с затравкой с использованием стека. Под стеком в данном случае мы будем понимать массив, в который можно последовательно помещать значения и последовательно извлекать, причем извлекаются элементы не в порядке поступления, а наоборот: по принципу "первым пришел - последним ушел" ("first in - last out"). Алгоритм заполнения выглядит следующим образом:

```
Поместить затравочный пиксель в стек
Пока стек не пуст:
       Извлечь пиксель из стека
       Инициализировать пиксель
       Для каждого из четырех соседних пикселей:
              Проверить, является ли он граничным и был ли он
инициализирован
              Если нет, то поместить пиксель в стек
```

Алгоритм можно модифицировать таким образом, что соседними будут считаться восемь пикселей (добавляются элементы, расположенные в диагональном направлении).

Методы растровой развертки рассмотрим сначала в применении к заполнению многоугольников. Простейший метод построения состоит в том, чтобы для каждого пикселя растра проверить его принадлежность внутренности многоугольника. Но такой перебор слишком неэкономичен, поскольку фигура может занимать лишь

незначительную часть экрана, а геометрический поиск - задача трудоемкая, сопряженная с длинными вычислениями. Алгоритм станет более эффективным, если предварительно выявить минимальный прямоугольник, в который погружен контур многоугольника, но и этого может оказаться недостаточно.

В случае, когда многоугольник, ограничивающий область, задан списком вершин и ребер (ребро определяется как пара вершин), можно предложить еще более экономный метод. Для каждой сканирующей строки определяются точки пересечения с ребрами многогранника, которые затем упорядочиваются по координате x. Определение того, какой интервал между парами пересечений есть внутренний для многогранника, а какой нет, является достаточно простой логической задачей. При этом если сканирующая строка проходит через вершину многогранника, то это пересечение должно быть учтено дважды в случае, когда вершина является точкой локального минимума или максимума. Для поиска пересечений сканирующей строки с ребрами можно использовать алгоритм Брезенхема построения растрового образа отрезка.

В заключение в качестве примера приведем алгоритм закраски внутренней области треугольника, основанный на составлении полного упорядоченного списка всех отрезков, составляющих этот треугольник. Для записи горизонтальных координат концов этих отрезков будем использовать два массива X_{min} и X_{max} размерностью, равной числу пикселей растра по вертикали (рис. 8.11).

Построение начинается с инициализации массивов X_{min} и X_{max}: массив X_{max} заполняется нулями, а массив X_{min} - числом N, равным числу пикселей растра по горизонтали. Затем определяем значения j_{min}, J_{max}, ограничивающие треугольник в вертикальном направлении. Теперь, используя модифицированный алгоритм Брезенхема, занесем границы отрезков в массивы X_{min} и X_{max}. Для этого всякий раз при переходе к очередному пикселю при формировании отрезка вместо его инициализации будем сравнивать его координату i с содержимым j-й ячейки массивов. Если $X_{min}[j] > i$, то записываем координату i в массив X_{min}. Аналогично при условии $X_{max}[j] < i$ координату i записываем в массив X_{max}.

Если теперь последовательно применить алгоритм Брезенхема ко всем трем сторонам треугольника, то мы получим нужным образом заполненные массивы границ. Остается только проинициализировать пиксели внутри отрезков $\{(X_{min[j],j}),\ (X_{max}[j], j)\}$.

Этот алгоритм можно легко распространить на случай произвольного выпуклого многоугольника.

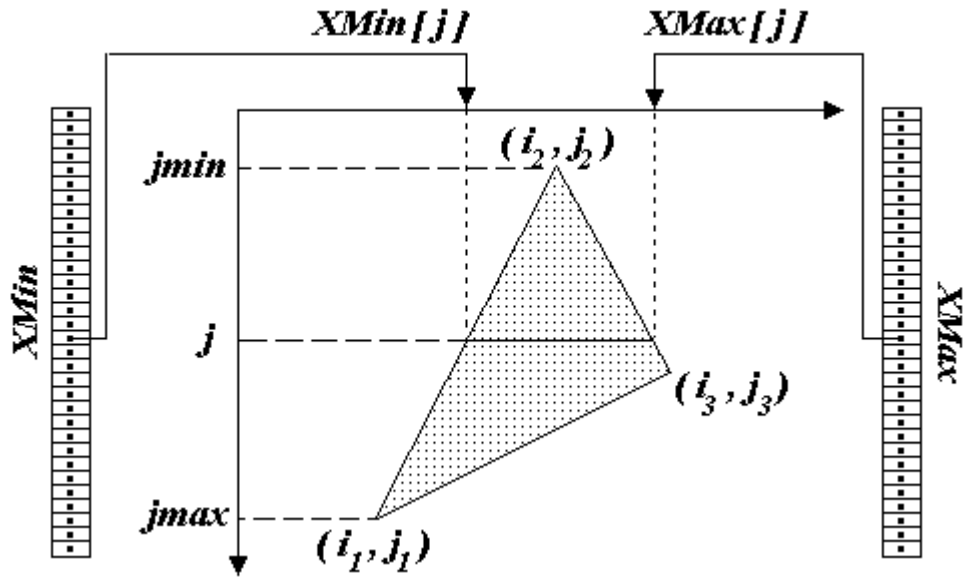


Рис. 8.11. Схема построения растровой развертки треугольника

Вопросы и упражнения

1. Что такое разложение в растр?
2. Какова математическая основа растрового разложения в алгоритме Брезенхема?
3. По какому критерию инициализируется пиксель в этом алгоритме?
4. Чем отличаются ветви алгоритма при углах наклона <45 $\deg$ и >45 $\deg$?
5. Какую часть окружности достаточно построить, чтобы затем путем отражений получить окружность целиком?
6. Какую часть эллипса достаточно построить, чтобы затем путем отражений получить эллипс целиком?
7. Назовите два типа алгоритмов заполнения областей.
8. Какая структура данных используется в алгоритмах с затравкой?
9. Какие данные используются при построении растровой развертки треугольника?

Лекция 9. Закрашивание. Рендеринг полигональных моделей

Модели освещения. Закраска граней: плоское закрашивание, метод Гуро, метод Фонга. Устранение ступенчатости (антиэлайзинг)

Визуальное восприятие объектов окружающей действительности представляет собой сложный процесс, имеющий как физические, так и психологические аспекты. Во второй главе мы уже обсуждали некоторые особенности восприятия света и цвета глазом человека. К тому, что уже было сказано о спектральной чувствительности глаза, надо добавить еще несколько моментов.

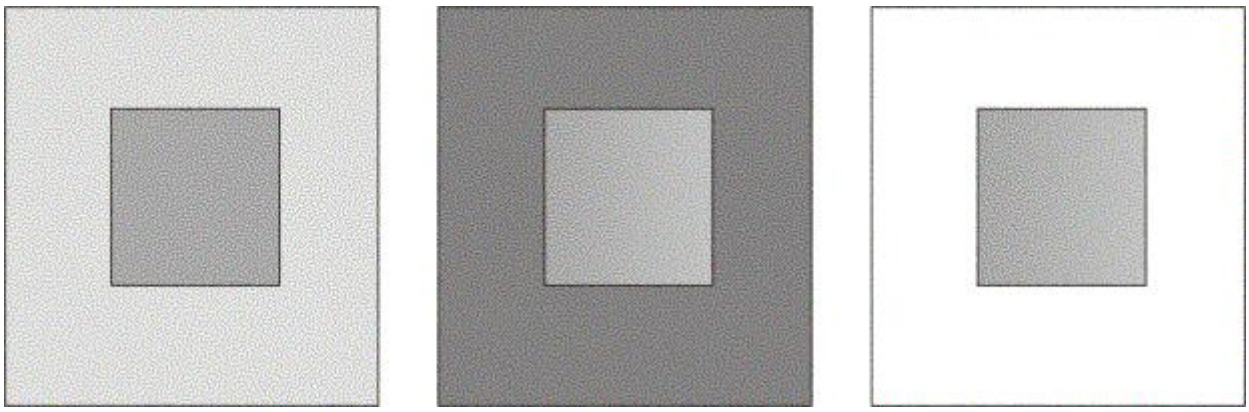


Рис. 9.1. Эффекты восприятия изображения

Глаз адаптируется к средней яркости рассматриваемой сцены, поэтому при смене фона изменяется восприятие сцены. Например, однородно окрашенная область на более темном фоне будет казаться более яркой, чем на светлом. Кроме того, она будет восприниматься как более обширная (рис. 9.1).

Еще одна особенность восприятия заключается в том, что граница равномерно освещенной области кажется более яркой по сравнению с внутренними частями. Это явление было обнаружено Эрнстом Махом, поэтому оно получило название **эффекта полос Маха**. Такие особенности необходимо учитывать, если мы стремимся к созданию реалистических изображений сцен.

При формировании изображения сцен, содержащих зеркальные и полупрозрачные поверхности, следует использовать законы геометрической оптики, преломляющие свойства материалов, эффекты смешения цветов и т.д.

Простая модель освещения

Объекты окружающего пространства становятся видимыми для глаза благодаря световой энергии, которая может излучаться поверхностью предмета, отражаться или проходить сквозь нее. В свою очередь, отражение света от поверхности зависит от физических свойств материала, из которого она изготовлена, а также от характера и расположения источника света. Яркость (или интенсивность) освещения зависит от энергии светового потока, которая обуславливается, во- первых, мощностью источника света, а во-вторых, отражающими и пропускающими свойствами объекта.

Сначала мы рассмотрим модель освещения, учитывающую только отражение. Свойства отраженного света зависят главным образом от направления лучей и характеристик отражающей поверхности.

Отражение может быть двух видов: **диффузное** и **зеркальное**. Первое из них возникает в ситуации, когда свет как бы проникает под поверхность объекта, поглощается, а потом равномерно излучается во всех направлениях. Поверхность в этом случае рассматривается как идеальный рассеиватель. При этом возникает эффект матового света, а видимая освещенность того или иного участка поверхности не зависит от положения наблюдателя. Зеркальное отражение, наоборот, происходит от внешней поверхности, интенсивность его неоднородна, поэтому видимый максимум освещенности зависит от положения глаза наблюдателя.

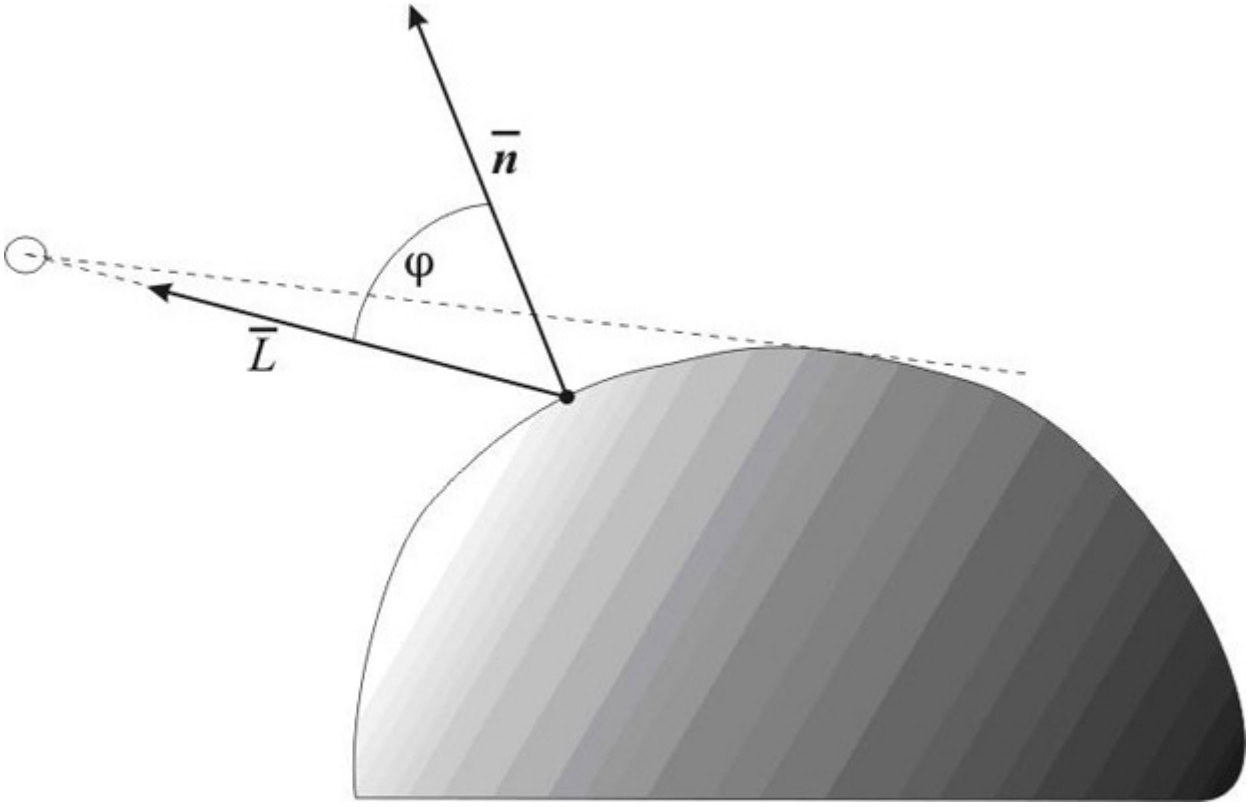


Рис. 9.2. Освещение точечным источником

Свет точечного источника отражается от поверхности рассеивателя по закону Ламберта: интенсивность отражения пропорциональна косинусу угла между внешней нормалью к поверхности и направлением к источнику света (рис. 9.2). Если I_S - интенсивность источника света, φ - угол между вектором внешней нормали к поверхности и направлением к источнику света, то интенсивность отраженного света определяется формулой

$$I = \begin{cases} I_S \cos\varphi \text{ при } 0 \le \varphi \le \pi/2 \\ 0 \text{ в противном случае} \end{cases} \quad (9.1)$$

При таком расчете интенсивности получится очень контрастная картина, т.к. участки поверхности, на которые лучи от источника не попадают напрямую, останутся абсолютно черными. Для повышения реалистичности необходимо учитывать рассеивание света в окружающем пространстве. Поэтому вводится **фоновая освещенность**, зависящая от интенсивности рассеянного света I_F, и интенсивность отраженного света определяется выражением

$$I = \begin{cases} I_F k_F + k_s I_S \cos\varphi \text{ при } 0 \le \varphi \le \pi/2 \\ I_F k_F \text{ в противном случае} \end{cases} \quad (9.2)$$

где k_F - коэффициент диффузного отражения рассеянного света, k_S - коэффициент диффузного отражения падающего света, $0 \le k_S \le 1,\ 0 \le k_F \le 1.$

В описанной модели пока никак не учитывалась удаленность источника света от поверхности, поэтому по освещенности двух объектов нельзя судить об их взаимном расположении в пространстве. Если мы хотим получить перспективное изображение, то необходимо включить затухание интенсивности с расстоянием. Обычно интенсивность света обратно пропорциональна квадрату расстояния от источника. В качестве расстояния до источника в случае перспективного преобразования можно взять расстояние до центра проекции, и если он достаточно удален, то изображение будет достаточно адекватным. Но если этот центр расположен близко к объекту, то квадрат расстояния меняется очень быстро, и в этом случае лучше использовать линейное затухание. В этом случае интенсивность отраженного света от непосредственно освещенных участков поверхности будет задаваться формулой

$$I = I_F k_F + \frac{k_S I_S \cos\varphi}{d + C} \quad (9.3)$$

где d - расстояние до центра проекции, а C - произвольная постоянная. Если центр проекции находится на бесконечности, т. е. при параллельном проецировании, то в качестве d можно взять расстояние до объекта, наиболее близкого к наблюдателю.

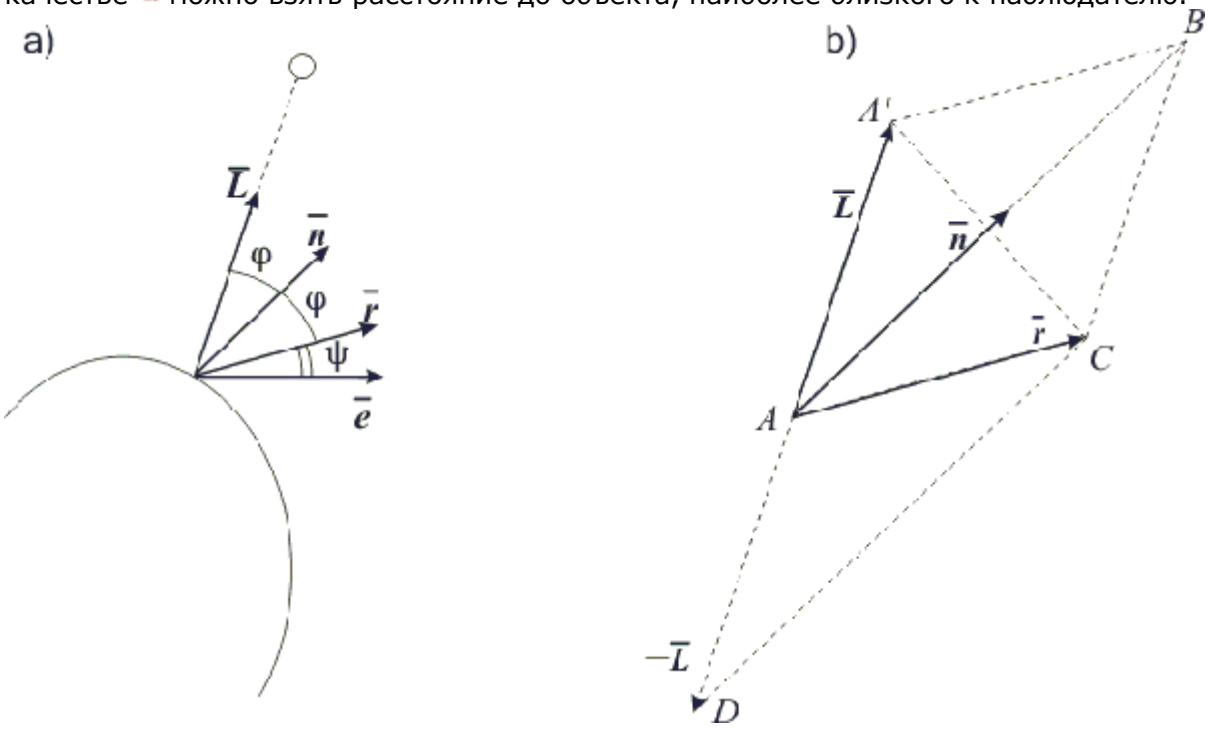
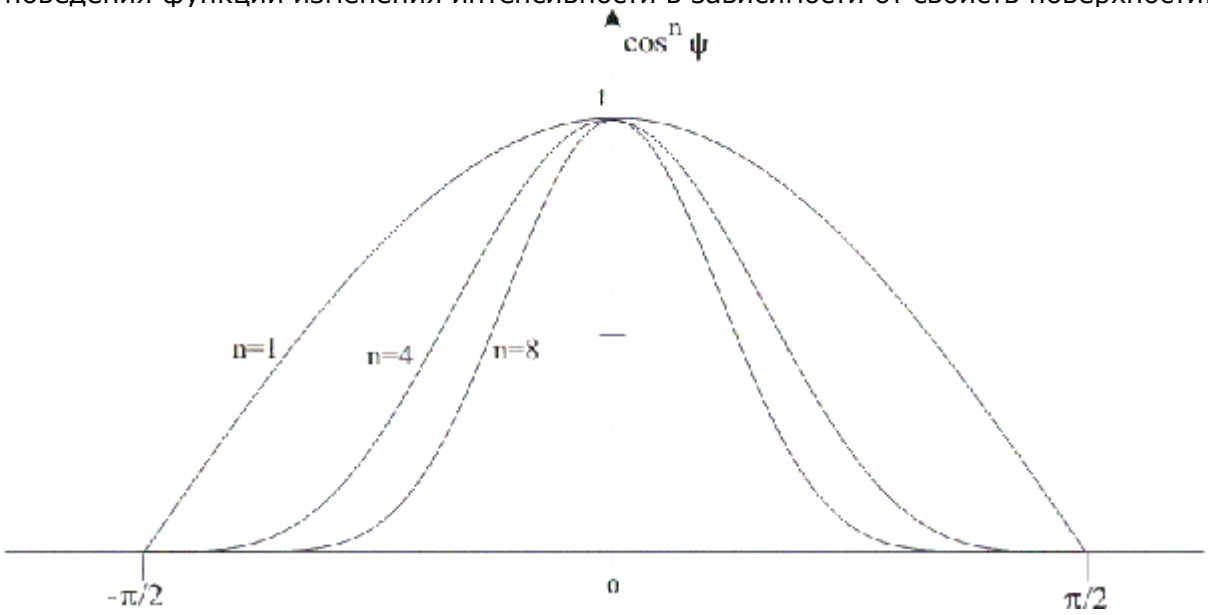


Рис. 9.3. Зеркальное отражение

В отличие от диффузного, зеркальное отражение является направленным. Идеальное зеркало отражает лучи по принципу "отраженный и падающий лучи лежат в одной плоскости, причем угол падения равен углу отражения" (имеется в виду угол между направлением луча и нормалью к поверхности). Если поверхность не идеально зеркальная, то лучи отражаются в различных направлениях, но с разной интенсивностью, а функция изменения интенсивности имеет четко выраженный максимум. Поскольку физические свойства зеркального отражения довольно сложны, то в компьютерной графике используется эмпирическая модель Фонга. Суть ее заключается в том, что для глаза наблюдателя интенсивность зеркально отраженного луча зависит от угла между идеально отраженным лучом и направлением к наблюдателю (рис. 9.3а). Кроме того, поскольку зеркальное отражение зависит еще и от длины волны, это также будем учитывать в формуле для вычисления интенсивности. Модель Фонга описывается соотношением

$$I_Z = \omega(\varphi, \lambda) \cdot I_S \cos^n \psi \quad (9.4)$$

где $\omega(\varphi,\lambda)$ - функция отражения, λ - длина волны. Степень, в которую возводится косинус угла, влияет на размеры светового блика, наблюдаемого зрителем. Графики этой функции приведены на рис. 9.4, и они как раз являются характерными кривыми поведения функции изменения интенсивности в зависимости от свойств поверхности.



увеличить изображение
Рис. 9.4. Зеркальное отражение

Теперь модель освещенности, учитывающую зеркальное и диффузное отражения, можно описать формулой

$$I = I_F k_F + \frac{I_S(k_s \cos\varphi + \omega(\varphi,\lambda)\cdot\cos^n\psi)}{d+C}. \quad (9.5)$$

Используя единичные векторы $\overrightarrow{L}$ (направление к источнику) и $\overrightarrow{n}$ (внешняя нормаль), косинус угла φ можно вычислить через скалярное произведение: $\cos\varphi = (\overrightarrow{L}\cdot\overrightarrow{n})$. Для расчета интенсивности зеркального отражения сначала надо определить отраженный вектор $\overrightarrow{r}$. Из рис. 9.3b видно, что $\overrightarrow{r} = |AB|\cdot\overrightarrow{n} - \overrightarrow{L}$. С другой стороны AB является диагональю ромба $AA'BC$, поэтому $|AB| = 2\cdot(\overrightarrow{L}\cdot\overrightarrow{n})$. Учитывая все эти соотношения, получаем формулу

$$I = I_F k_F + \frac{I_S\{k_s(\overrightarrow{L}\cdot\overrightarrow{n}) + \omega(\varphi,\lambda)\cdot[2\cdot(\overrightarrow{L}\cdot\overrightarrow{n})\cdot(\overrightarrow{e}\cdot\overrightarrow{n}) - (\overrightarrow{e}\cdot\overrightarrow{L})]^n\}}{d+C}. \quad (9.6)$$

В алгоритмах закрашивания с использованием цветовых моделей интенсивность рассчитывается для каждого из базовых цветов, поскольку изменение интенсивности при зеркальном отражении зависит от длины волны.

Закраска граней

Плоское закрашивание

Если предположить, что источник света находится на бесконечности, то лучи света, падающие на поверхность, параллельны между собой. Если к этому добавить условие, что наблюдатель находится в бесконечно удаленной точке, то эффектом ослабления

света с увеличением расстояния от источника также можно пренебречь. Кроме того, такое положение наблюдателя означает еще и то, что векторы, направленные от разных точек поверхности к наблюдателю, также будут параллельны. При выполнении всех этих условий, как следует из формулы (9.6), плоская грань во всех точках имеет одинаковую интенсивность освещения, поэтому она закрашивается одним цветом. Такое закрашивание называется **плоским**.

Если мы аппроксимируем некоторую гладкую поверхность многогранником, то при плоском закрашивании неизбежно проявятся ребра, поскольку соседние грани с различными направлениями нормалей имеют разный цвет. Эффект полос Маха дополнительно усиливает этот недостаток. Для его устранения при использовании этого способа закрашивания можно лишь увеличить число граней многогранника, что приводит к увеличению вычислительной сложности алгоритма.

Закраска методом Гуро

Один из способов устранения дискретности интенсивностей закрашивания был предложен Гуро. Его метод заключается в том, что используются не нормали к плоским граням, а нормали к аппроксимируемой поверхности, построенные в вершинах многогранника. После этого вычисляются интенсивности в вершинах, а затем во всех внутренних точках многоугольника выполняется билинейная интерполяция интенсивности.

Метод сочетается с алгоритмом построчного сканирования. После того как грань отображена на плоскость изображения, для каждой сканирующей строки определяются ее точки пересечения с ребрами. В этих точках интенсивность вычисляется с помощью линейной интерполяции интенсивностей в вершинах ребра. Затем для всех внутренних точек многоугольника, лежащих на сканирующей строке, также вычисляется интенсивность методом линейной интерполяции двух полученных значений. На рис. 9.5 показан плоский многоугольник с вычисленными значениями интенсивностей в вершинах.

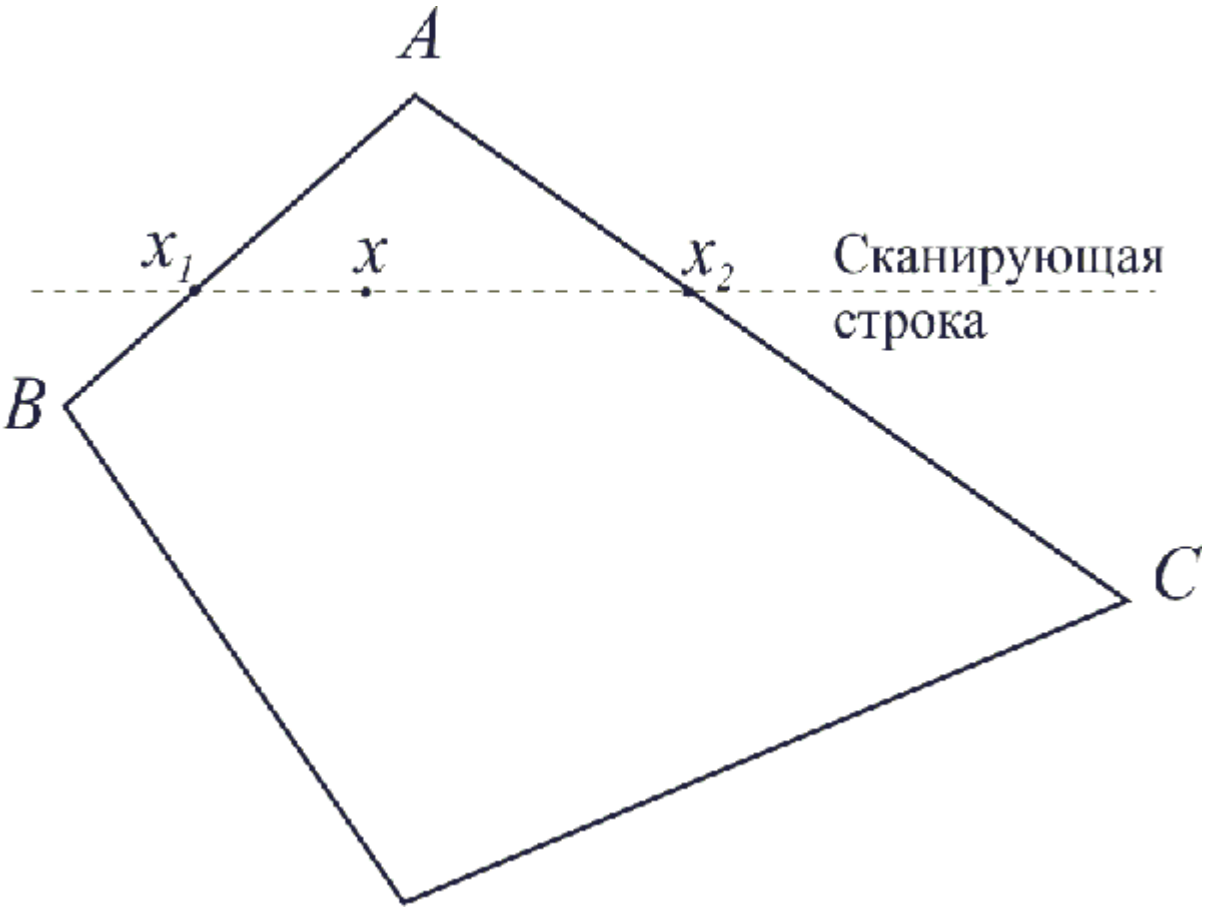


Рис. 9.5. Интерполяция интенсивности

Пусть I_A, I_B, I_C - интенсивности в вершинах A, B, C, x_A, x_B, x_C - горизонтальные координаты этих точек. Тогда в точках пересечения сканирующей строки с ребрами многоугольника интенсивности можно вычислить по формулам интерполяции:

$$\begin{aligned} I_1 &= t_1 I_A + (1-t_1) I_B, \quad t_1 = \frac{x_1 - x_B}{x_A - x_B}, \\ I_2 &= t_2 I_A + (1-t_2) I_C, \quad t_2 = \frac{x_2 - x_C}{x_A - x_C}. \end{aligned} \quad (9.7)$$

После этого интенсивность в точке x получаем путем интерполяции значений на концах отрезка:

$$I = t I_1 + (1-t) I_2, \quad t = \frac{x_2 - x}{x_2 - x_1} \quad (9.8)$$

К недостаткам метода Гуро следует отнести то, что он хорошо работает только с диффузной моделью отражения. Форма бликов на поверхности и их расположение не могут быть адекватно воспроизведены при интерполяции на многоугольниках. Кроме того, есть проблема построения нормалей к поверхности. В алгоритме Гуро нормаль в вершине многогранника вычисляется путем усреднения нормалей к граням, примыкающим к этой вершине. Такое построение сильно зависит от характера разбиения.

Закраска методом Фонга

Фонг предложил вместо интерполяции интенсивностей произвести интерполяцию вектора нормали к поверхности на сканирующей строке. Этот метод требует больших вычислительных затрат, поскольку формулы интерполяции (9.6)–(9.7) применяются к трем компонентам вектора нормали, но зато дает лучшую аппроксимацию кривизны

поверхности. Поэтому зеркальные свойства поверхности воспроизводятся гораздо лучше.

Нормали к поверхности в вершинах многогранника вычисляются так же, как и в методе Гуро. А затем выполняется билинейная интерполяция в сочетании с построчным сканированием. После построения вектора нормали в очередной точке вычисляется интенсивность.

увеличить изображение
Рис. 9.6. Три способа закрашивания

Этот метод позволяет устранить ряд недостатков метода Гуро, но не все. В частности, эффект полос Маха в отдельных случаях в методе Фонга бывает даже сильнее, хотя в подавляющем большинстве случаев аппроксимация Фонга дает лучшие результаты. На рис. 9.6 приведены результаты закрашивания поверхности вращения, аппроксимированной многогранником, который составлен из треугольных граней: a) - плоское закрашивание, b) - закрашивание по методу Гуро, c) - закрашивание по методу Фонга. Первый из вариантов дает изображение ребристой поверхности с очень контрастными переходами от одной грани к другой. Вторая модель дает более гладкое изображение, но в районе бликов отчетливо наблюдаются линии ребер, хотя и сглаженные. Третий вариант получился наиболее гладким, зеркальные блики имеют достаточно реалистичную форму.

Более сложные модели освещения

Когда мы рассматривали алгоритмы удаления невидимых линий, предполагалось, что сцена включает только непрозрачные объекты. В простой модели освещения тоже речь шла о непрозрачных поверхностях. Теперь можно усложнить задачу, включив в модель не только отражение света, но и преломление.

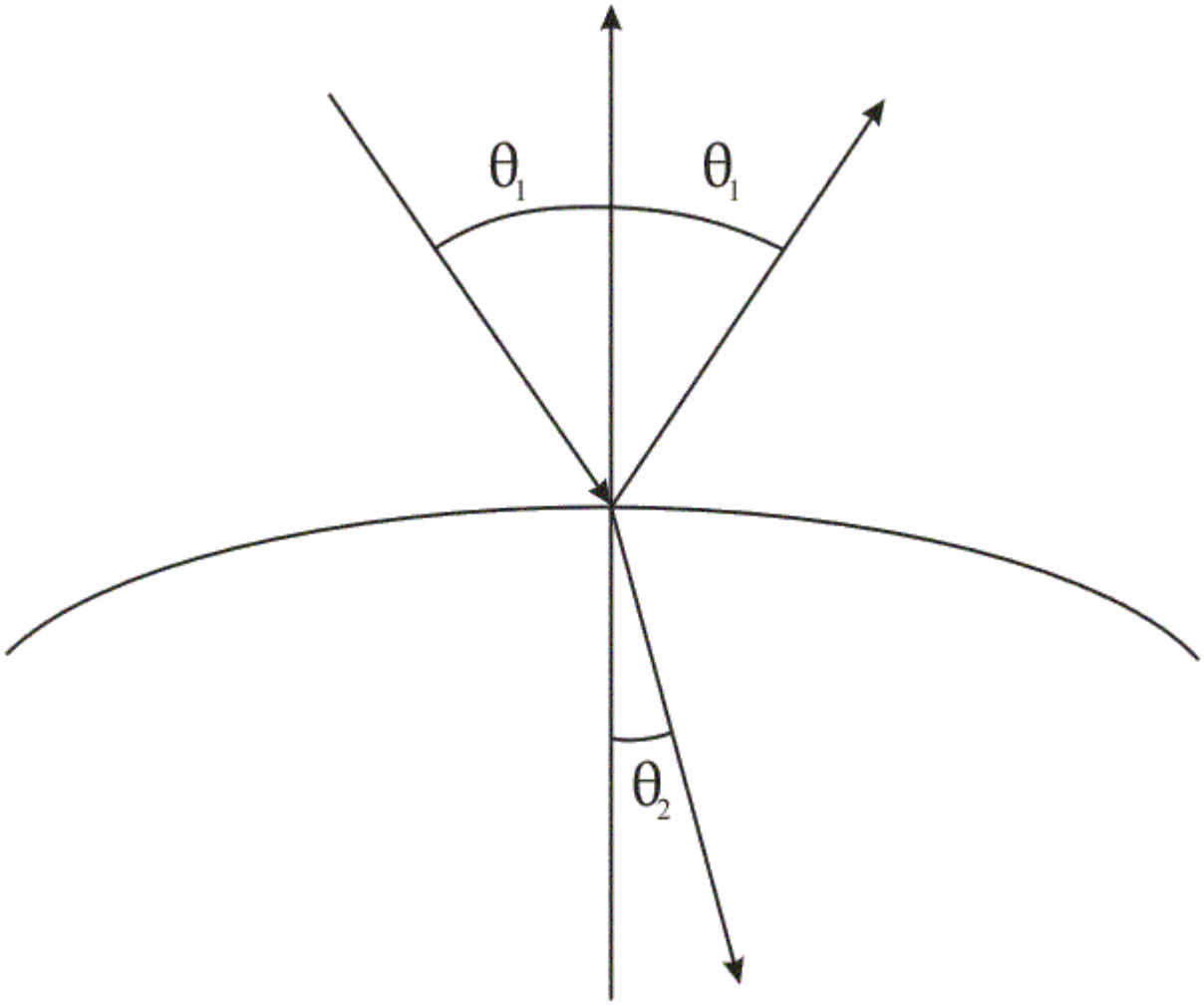


Рис. 9.7. Преломленный и отраженный лучи

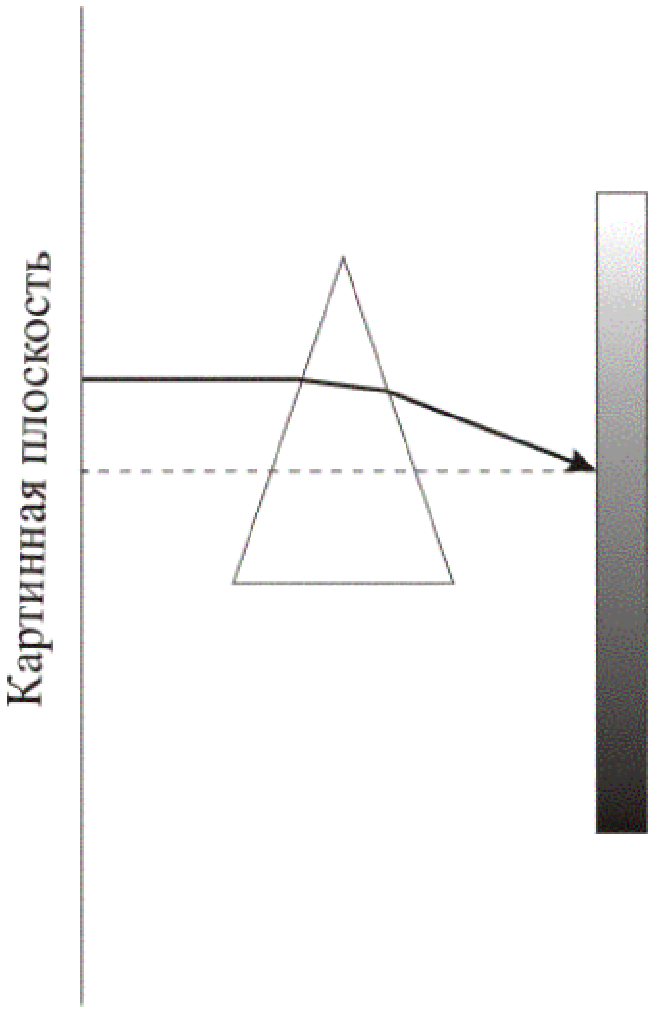


Рис. 9.8. Преломление в призме

При переходе луча из одной среды в другую его направление изменяется согласно закону Синеллиуса: преломленный луч лежит в плоскости, образуемой нормалью к плоскости и падающим лучом, а углы, образуемые лучами с нормалью, связаны формулой

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2,$$

где η_1, η_2 - показатели преломления двух сред (рис. 9.7). Пропускание света также может быть диффузным (если часть энергии света рассеивается средой) или направленным. В первом случае мы имеем дело с полупрозрачными телами, которые изменяют окраску видимых сквозь них объектов. Во втором случае тело является прозрачным, и оно визуально обнаруживается только благодаря искажениям объектов за счет преломления лучей.

При наличии в пространственной сцене прозрачных или полупрозрачных объектов надо учитывать, что изображение других объектов будет отличаться от обычной проекции на картинную плоскость (рис. 9.8). Эти эффекты хорошо знакомы всем, кто сталкивался с различными линзами. Для построения изображения таких сцен целесообразно использовать алгоритмы с обратной трассировкой лучей.

Для изображения полупрозрачных поверхностей без учета преломления можно ввести так называемый коэффициент прозрачности κ, который позволяет смешивать интенсивности для видимой поверхности и той, что расположена за ней:

$$I = \kappa I_1 + (1-\kappa) I_2, \quad 0 \le \kappa \le 1$$

При $\kappa = 1$ поверхность непрозрачна, при $\kappa = 0$ - полностью прозрачна. Для полупрозрачных тел необходимо учитывать их объемную структуру.

Методы построения изображений сцен с прозрачными и полупрозрачными объектами будут более подробно рассмотрены в следующей лекции.

Устранение ступенчатости (антиэлайзинг)

При построении растрового образа линий (см. лекцию 8) мы сталкиваемся с эффектом ступенчатости, связанным с дискретизацией непрерывного объекта. Искажение идеального образа происходит потому, что из всего множества точек мы выбираем только те, которые оказываются ближе всего к центру элемента растра, и инициализируем этот элемент.

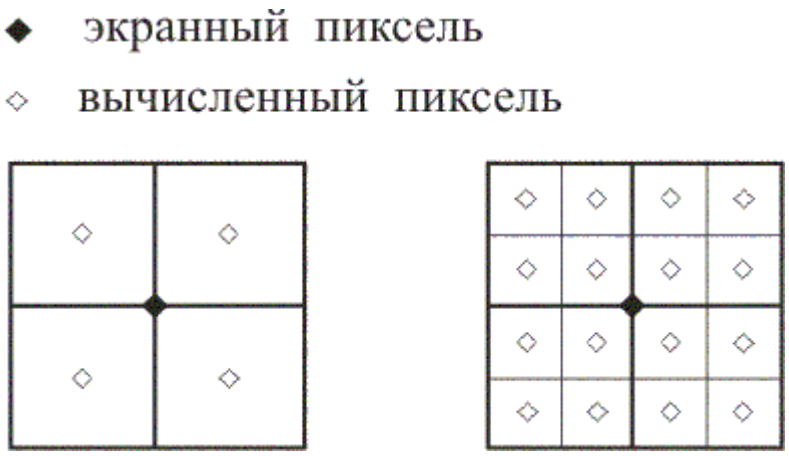
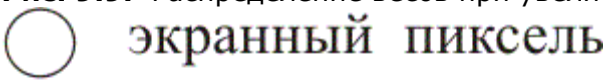


Рис. 9.9. Распределение весов при увеличении разрешения в 4 раза



1	2	3	4	3	2	1
2	4	6	8	6	4	2
3	6	9	12	9	6	3
4	8	12	16	12	8	4
3	6	9	12	9	6	3
2	4	6	8	6	4	2
1	2	3	4	3	2	1

Рис. 9.10. Распределение весов при увеличении разрешения в 16 раз

Для предотвращения сильных искажений в этом случае можно, во- первых, повышать разрешение растра, что позволяет отображать всё более мелкие детали объектов. Но у этого подхода есть свои чисто физические ограничения. Второй подход заключается в том, что растр рассчитывается с более высоким разрешением, а изображается с более низким - путем усреднения атрибутов пикселей первого более детального растра с определенными весами. Если веса одинаковы, то мы получаем равномерное

усреднение, как показано на рис. 9.9. Лучших результатов можно достигнуть, если использовать разные веса у пикселей первого растра. На рис. 9.10 показано распределение весов при детализации пикселя экранного растра.

Другой метод устранения ступенчатости состоит в том, чтобы рассматривать пиксель не как точку, а как некоторую конечную область. В алгоритмах построения растровой развертки пиксель считается принадлежащим области закрашивания, если его центр находился внутри идеального образа области. Если рисунок черно-белый, то устранить эффект ступенчатости растра практически невозможно. Но при наличии оттенков полутонов можно задать интенсивность цвета пикселя в зависимости от площади его пересечения с областью.

Рассмотрим применение этого метода на примере раскраски многоугольника. Ребро многоугольника строится с использованием алгоритма Брезенхема, описанного в лекции 8. Здесь в этот алгоритм будут внесены изменения, включающие параметр максимального числа уровней интенсивностей. Определяя принадлежность пикселя многоугольнику, мы будем использовать в качестве ошибки e долю площади, принадлежащей идеальной фигуре (рис. 9.11).

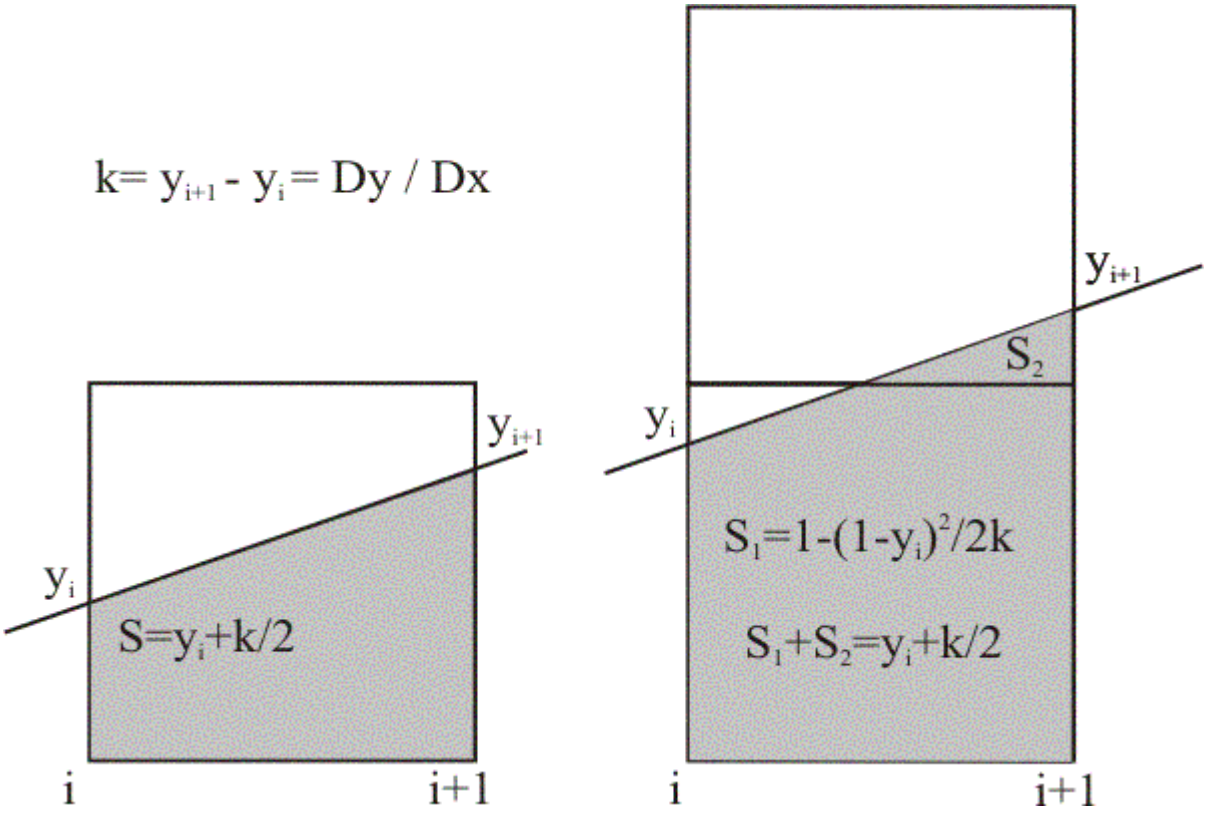


Рис. 9.11. Отсекаемая отрезком площадь пикселя

Рассмотрим опять случай, когда отрезок направлен в положительный квадрант координатной плоскости под углом, меньшим $\pi/4$. Идеальный отрезок при заданном значении целочисленной координаты i может пересекать один или два пикселя. В предыдущей версии алгоритма выбирался пиксель, центр которого располагался ближе к отрезку. Теперь интенсивность для обоих пикселей будет задаваться в зависимости от **степени близости** каждого из них. Инициализация пикселя будет использовать интенсивность в качестве параметра. Предполагается, что отрезок начинается с угла первого пикселя, исходя из чего и задается начальная интенсивность. Блок-схема алгоритма приведена на рис. 9.12.

Устранение эффекта ступенчатости с математической точки зрения является задачей **сглаживания**. Приведенный здесь алгоритм, использующий площади пересечения растра и идеального образа, можно описать с помощью операции свертки функции.

Сначала дадим необходимые определения. **Сверткой** функции $f(x)$ называется интеграл вида

$$C(\xi) = \int_{-\infty}^{+\infty} K(\xi - t) f(t) dt. \quad (9.9)$$

Функция $K(x)$ называется **ядром свертки**. В качестве ядра свертки обычно используется либо функция с конечным носителем (т.е. отличная от нуля лишь на некотором конечном интервале), либо быстро убывающая на бесконечности функция (это может являться необходимым условием существования интеграла).

Рассмотрим в качестве свертываемой функции и ядра следующие функции:

$$f(x) = \begin{cases} x \text{ при } 0 \le x \le 1 \\ 0 \text{ при } x < 0 \text{ и } x > 1 \end{cases}, \qquad K(x) = \begin{cases} 1 \text{ при } 0 \le x \le 1 \\ 0 \text{ при } x < 0 \text{ и } x > 1 \end{cases}.$$

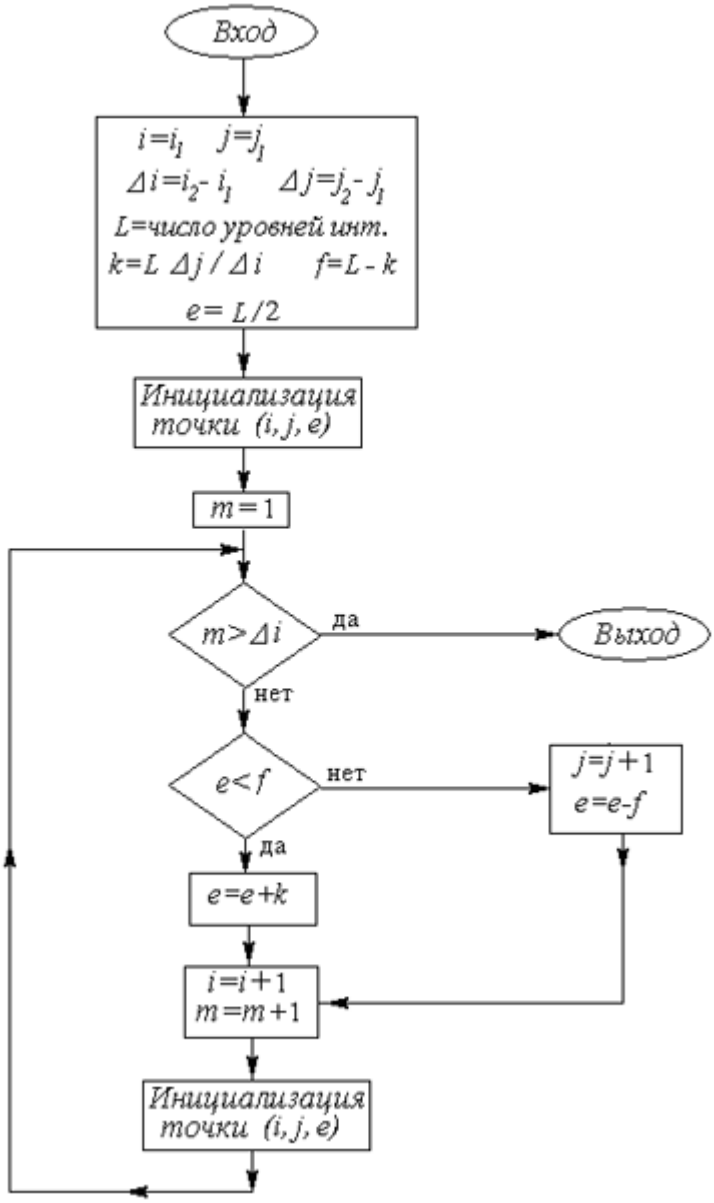


Рис. 9.12. Блок-схема модифицированного алгоритма Брезенхема

Тогда, в силу того, что подынтегральное выражение обращается в ноль при $\xi - t < 0$ и при $\xi - t > 0$, получаем

$$C(\xi) = \int_{\xi-1}^{\xi} f(t)dt.$$

Учитывая вид функции $f(x)$, получаем, что свертка будет отлична от нуля только на интервале $0 < \xi < 2$. Значения свертки в некоторых точках приведены в таблице 9.1.

Таблица 9.1. Значения свертки в узлах

ξ	0	1/2	1	3/2	2
$C(\xi)$	0	1/8	1/2	3/8	0

Очевидно, что наша свертка дает площадь пересечения треугольника, образованного свертываемой функцией с квадратом, основание которого есть отрезок $[\xi, xi+1]$ на оси OX .

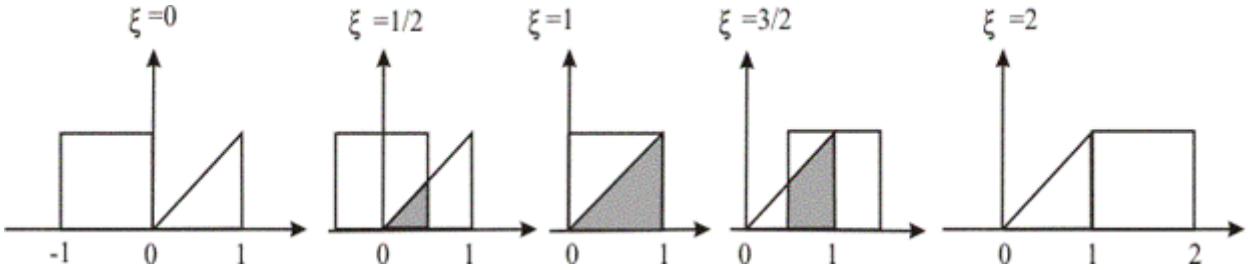
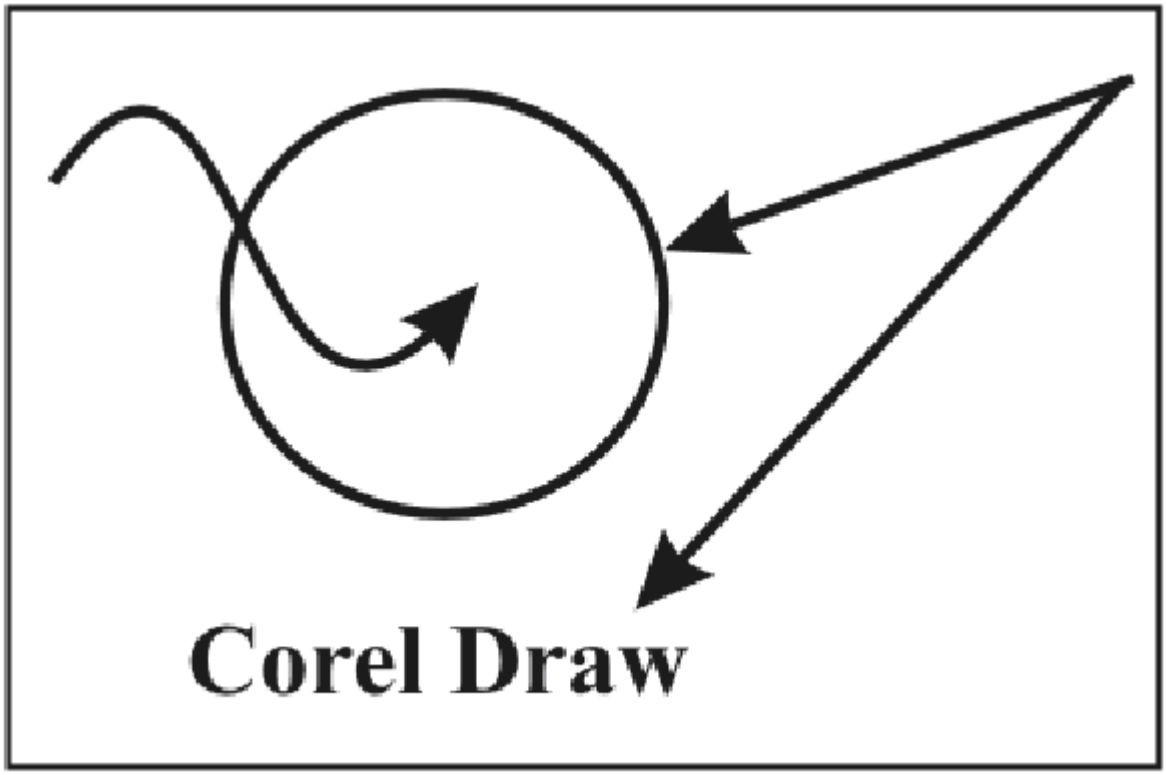


Рис. 9.13. Фигуры, соответствующие значениям свертки из таблицы 9.1

На рис. 9.13 приведен вид пересечения для всех пяти случаев из таблицы 9.1. Если сравнить эти результаты с рис. 9.11, то видно, что значения свертки при $k \leq 1$ дают площадь той части пикселя, что находится внутри многоугольника (если считать $y_i = 0$), а при $k > 1$ - сумму площадей двух пересекаемых пикселей.

В заключение проиллюстрируем результат применения алгоритма устранения ступенчатости на примере изображения, полученного с помощью программы Corel Draw. Эта программа представляет собой развитый графический редактор, позволяющий строить объекты векторной графики. На рис. 9.14 показано изображение простых графических примитивов, предварительно переведенное в растровую форму, на котором при большом увеличении заметно сглаживание с применением оттенков серого цвета.



увеличить изображение

Рис. 9.14. Сглаженные изображения

Вопросы и упражнения

1. Что такое эффект полос Маха?
2. Чем отличается диффузное отражение от зеркального?
3. От чего зависит интенсивность освещения точки поверхности при диффузном отражении?
4. От чего зависит интенсивность освещения точки поверхности при зеркальном отражении?
5. Какие параметры учитывает модель зеркального отражения, предложенная Фонгом?
6. Меняется ли интенсивность освещения при плоском закрашивании грани?
7. Какой параметр интерполируется при закрашивании методом Гуро?
8. Какой параметр интерполируется при закрашивании методом Фонга?
9. В чем состоит идея алгоритмов антиэлайзинга, основанных на уровне детализации растра?
10. Какой параметр используется в алгоритме антиэлайзинга, учитывающем размеры пикселя?

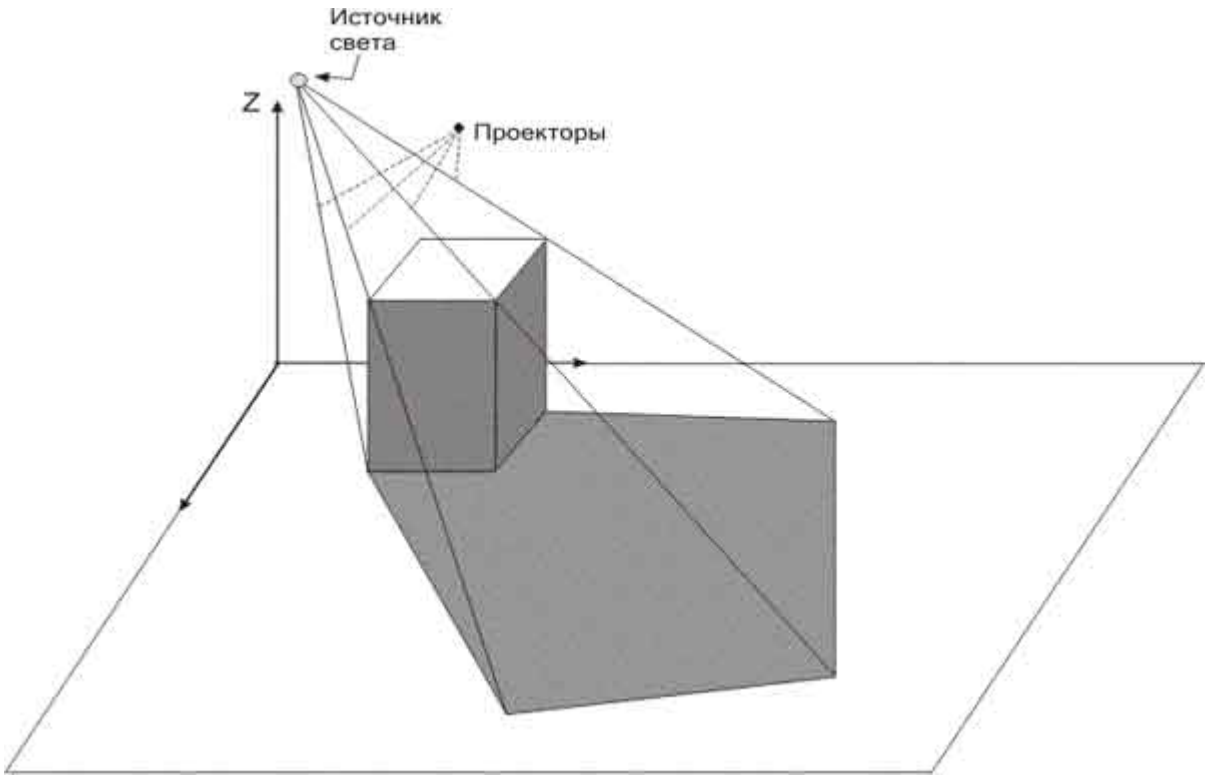
Лекция 10. Визуализация пространственных реалистических сцен

Свето-теневой анализ. Метод излучательности. Глобальная модель освещения с трассировкой лучей. Алгоритм обратной трассировки

Свето-теневой анализ

Тени возникают как результат экранирования источника света предметами, составляющими сцену. При этом существуют полные тени и полутени, что связано с физической природой света: полная тень образуется в центральной части затенения, а полутень - вблизи ее границ. В нашем курсе мы будем касаться только полной тени, хотя и существуют технологии изображения полутеней, сопровождающиеся большим объемом вычислений.

Если считать, что наблюдатель находится в одной точке с источником света, то тени в наблюдаемой сцене не возникают: затеняемая область является невидимой. Во всех прочих случаях тени видны. Источник света может находиться на бесконечности или на конечном расстоянии, причем во втором случае он может оказаться в поле зрения наблюдателя.



увеличить изображение

Рис. 10.1. Образование теней при конечном расстоянии до источника света

Для бесконечно удаленного источника света тени на картинной плоскости получаются в результате параллельной проекции объектов, а для близкого источника - центральной проекции. Объекты и их части становятся невидимыми, если они попадают в область тени, поэтому при построении изображения задача об удалении невидимых областей решается дважды: относительно наблюдателя в процессе проецирования и относительно источника света. При определении расположения теней строятся проекции невидимых с позиции источника света граней (неосвещенных) на картинную плоскость, в результате чего получаются теневые многоугольники. Эти многоугольники строятся для всех объектов сцены и заносятся в список. Отметим, что теневые многоугольники не зависят от положения наблюдателя, поэтому при осмотре сцен с различных точек зрения они строятся только один раз. Неосвещенные области определяются теми же методами, которые были описаны ранее в лекции 7, а для построения проекций используются методы, некоторые из которых описаны в лекции 8. В частности, можно применять матрицы проекций в однородной системе координат.

Впервые идея совмещенного анализа видимости и затененности была предложена в 1968 г. Аппелем. В качестве примера рассмотрим один алгоритм на основе построчного сканирования, состоящий из двух основных этапов.

I. Анализ сцены по отношению к источнику света. Для всех многоугольников, полученных в результате проецирования сцены, определяются неосвещенные (затененные) участки и теневые многоугольники (проекционные тени), причем многоугольники образуют пронумерованный список. Для этих многоугольников формируется матрица $A = (a_{ij})$, позволяющая определить, отбрасывает ли многоугольник тень и какие из многоугольников он может закрывать. Если для некоторых значений $i, j \quad a_{ij} = 1$, то это означает, что многоугольник с номером i может отбрасывать тень на многоугольник с номером j.

Таким образом, на этом этапе основным является вопрос об эффективном алгоритме построения такой матрицы. Если проекция включает N многоугольников, то

необходимо рассмотреть "взаимоотношения" $N \cdot (N-1)$ пар многоугольников. Сократить перебор можно за счет погружения объектов в прямоугольные или сферические оболочки или путем использования сортировки по глубине.

II. Анализ сцены по отношению к наблюдателю. Выполняются два процесса сканирования. Первый - для определения отрезков, видимых с позиции наблюдателя (о нем рассказывалось раньше в лекции 7). Второй - для определения пересечений отрезков с теневыми многоугольниками из списка. Рекурсивный алгоритм второго сканирования состоит из четырех основных шагов.

Для каждого видимого отрезка

1. Если нет ни одного теневого многоугольника, то отрезок изображается с основной интенсивностью.
2. Если многоугольник, содержащий видимый отрезок, не пересекается с теневыми многоугольниками, то отрезок изображается с основной интенсивностью.
3. Если отрезок полностью закрывается некоторыми теневыми многоугольниками, то интенсивность его изображения определяется с учетом затенения всеми этими многоугольниками.
4. Если несколько теневых многоугольников частично закрывают отрезок, то он разбивается на ряд отрезков точками пересечения с теневыми многоугольниками.

Этот алгоритм предполагает, что затенение не абсолютное, т.е. затененные участки все-таки являются видимыми, только их освещенность падает в зависимости от количества и освещенности затеняющих многоугольников. При полном затенении в третьем пункте алгоритма отрезок становится полностью невидимым, а в четвертом дальнейшему анализу подвергаются только незатененные отрезки.

Способы расчета интенсивности при неполном затенении могут быть различны. В этом случае все затененные многоугольники имеют свою интенсивность в зависимости от выбранной модели освещенности. При этом можно учитывать расстояние затененного участка от поверхности, отбрасывающей тень.

Еще один алгоритм, часто применяемый при построении теней, носит название метода теневого буфера. Он строится на основе метода Z-буфера, описанного в лекции 7. Теневой буфер - это тот же Z-буфер, только с точки зрения источника света. Таким образом, используются два буфера: один - для расстояния от картинной плоскости до точек изображаемой сцены, а другой - для расстояний от этих же точек до источника света. Алгоритм позволяет изображать сцены с полным затенением и сводится к двум основным этапам:

1. Сцена рассматривается из точки расположения источника света в соответствующей системе координат. Итогом построения является полностью заполненный теневой буфер.
2. Сцена рассматривается с точки зрения наблюдателя, применяется обычный метод Z-буфера с небольшим дополнением. Если точка (x, y, z) является видимой в этой системе координат, то вычисляются ее координаты в системе, связанной с источником света - (x', y', z'), затем проверяется, является ли точка видимой с этой позиции. Для этого значение z' сравнивается со значением, содержащимся в теневом буфере для этой точки, и в случае видимости значение интенсивности заносится в буфер кадра в точке (x, y).

Оба приведенных алгоритма работают в пространстве изображения, т. е. имеют дело с проекциями на плоскость и некоторой дополнительной информацией о точках сцены, соответствующих этим проекциям. Существуют алгоритмы, работающие в трехмерном

объектном пространстве. В частности, для построения теней используются модификации алгоритма Вейлера-Азертона, описанного в лекции 7. Модификация заключается в том, что, как и в случае теневого буфера, задача удаления невидимых граней решается сначала с позиции источника света, а затем полученная информация об объектах используется при построении изображения с позиции наблюдателя. В общих чертах шаги алгоритма можно описать так:

1. Определяются грани, видимые из точки расположения источника света. С целью повышения эффективности запоминается информация только о видимых гранях. Поскольку анализ выполняется в системе координат, связанной с источником света, то полученные видимые многоугольники затем заново приводятся к исходной системе координат. Многоугольники связываются с гранями, которым они принадлежат (в результате затенения одна грань может содержать несколько многоугольников).
2. Сцена обрабатывается из положения наблюдателя. При изображении видимой грани учитываются только те многоугольники, которые входят в список, полученный на первом этапе, т.е. грань рассматривается как совокупность таких многоугольников.

При наличии нескольких источников света количество освещенных участков естественным образом увеличивается.

Метод излучательности

В этой лекции уже говорилось, что освещенность поверхности определяется собственным излучением тела и отраженными лучами, падающими от других тел (источников). Модель излучательности включает оба эти фактора и основана на уравнениях энергетического баланса. При этом выполняемые расчеты учитывают только взаимное расположение элементов сцены и не зависят от положения наблюдателя.

Представим сцену из N элементов (участков поверхностей). Освещенность будем моделировать как количество энергии, излучаемое поверхностью. Для каждого элемента это количество энергии складывается из собственной энергии (E_k) и отраженной доли энергии, полученной от других объектов. Предполагается, что для каждой пары элементов с номерами i, j можно определить, какая доля энергии одного попадает на другой (w_{ij}). Пусть α_i - коэффициент отражения энергии $i-$м элементом. Тогда полная энергия, излучаемая этим элементом, будет определяться уравнением $U_i = E_i + \alpha_i \sum_{j=1}^{N} w_{ij} U_j$.

Таким образом, мы получаем систему уравнений для нахождения значений U_i, которая в матричном виде выглядит следующим образом:

$$(I - W) \cdot U = E$$

где I - единичная матрица, U и T - векторы излучаемой и собственной энергий, а матрица W состоит из элементов $(\alpha_i w_{ij})$. Поскольку часть излучения элемента может не попадать ни на один из оставшихся, то

$$\sum_{i=1}^{N} w_{ij} \leq 1,$$

а это условие в сочетании с тем, что $\alpha_i < 1$ (отражение не является полным), приводит к тому, что матрица системы имеет так называемое **диагональное преобладание**, т.е. диагональный элемент по абсолютной величине больше, чем сумма остальных

элементов строки. В таком случае система уравнений имеет решение, которое можно найти с помощью численных методов.

Итак, шаги алгоритма изображения сцены сводятся к следующим:

1. Сцена разбивается на отдельные участки, для каждого из которых определяются значения $E_i, \alpha_i, w_{ij}, \quad j = 1, 2, \ldots, N$.
2. Находятся значения U_i для каждой из трех основных компонент цвета.
3. Для выбранной точки наблюдения строится проекция с удалением невидимых граней и осуществляется закрашивание, использующее значения U_i для задания интенсивности. При этом могут использоваться какие-либо алгоритмы, позволяющие сгладить изображение.

Сложным моментом в модели излучательности является расчет коэффициентов w_{ij}.

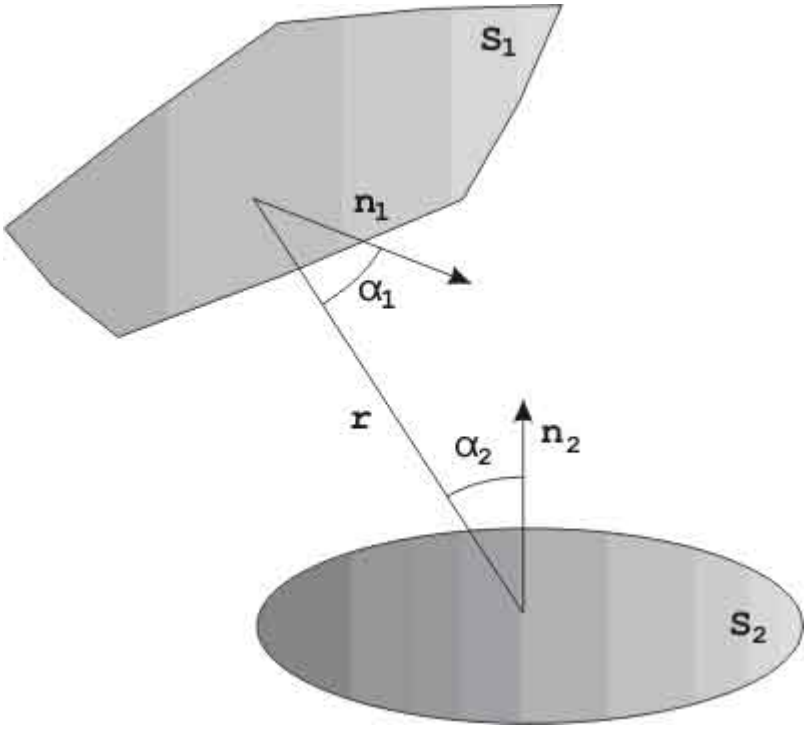


Рис. 10.2. Два элемента сцены

Рассмотрим один пример. Пусть имеется два элемента сцены S_1 и S_1 (рис. 10.2). Поскольку используется диффузная модель освещения, то доля энергии малого участка dS_1 с нормалью $\overrightarrow{n}_1$, излучаемая под углом α_1 к этой нормали, пропорциональна косинусу угла. Следовательно, в направлении элементарного участка dS_2 уходит доля энергии, пропорциональная косинусу угла между $\overrightarrow{n}_1$ и отрезком, который соединяет эти участки. Соответственно, получаемая вторым участком доля этой энергии будет пропорциональна косинусу угла между нормалью $\overrightarrow{n}_2$ и этим же отрезком. Итак, доля энергии, получаемая элементом dS_2 от элемента , dS_1 - $dw_{21} = \cos(\alpha_1) \cdot \cos(\alpha_2)/\pi r^2$, где r - расстояние между элементами. Кроме того, необходимо учесть, что излучаемая элементарным участком энергия равномерно распределена по всем направлениям. И, наконец, в каждой сцене одни объекты могут частично экранировать другие, поэтому надо ввести коэффициент, определяющий степень видимости объекта с позиции другого. Далее полученное выражение интегрируется по S_1 и S_2, что также может быть сложной задачей.

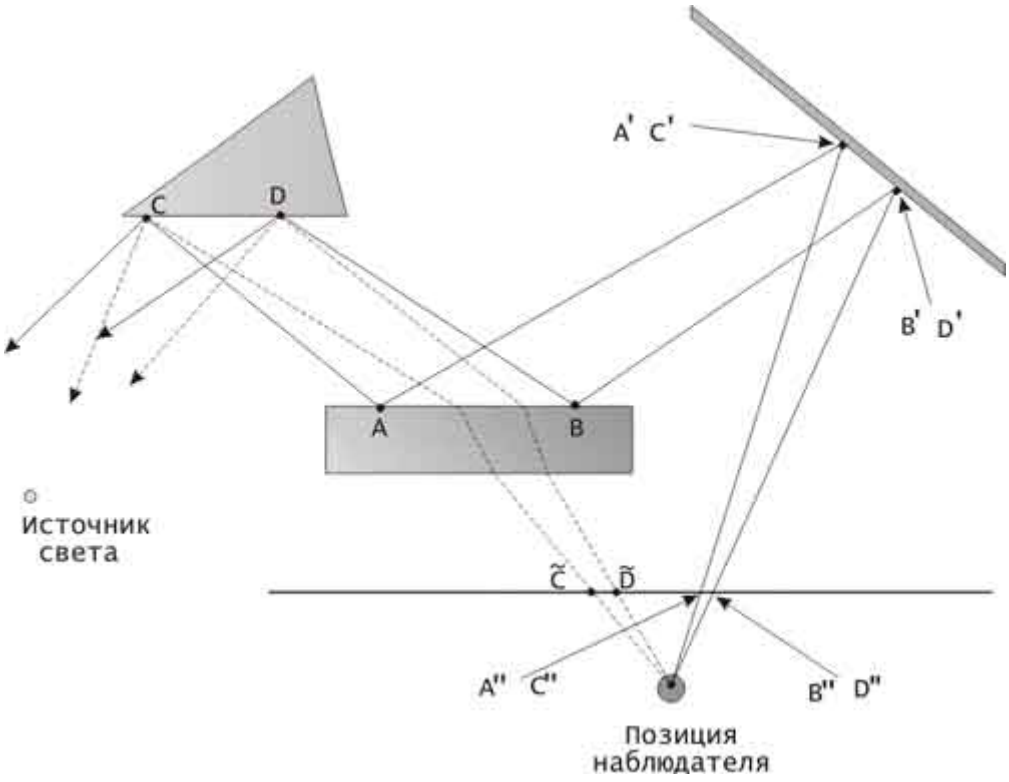
Отсюда видно, насколько трудоемкой может оказаться процедура вычисления коэффициентов w_{ij}. Поэтому, как правило, используются приближенные методы их вычисления. В частности, можно рассматривать поверхности объектов как многогранники, тогда элементами сцены будут плоские многоугольники, для которых формулы несколько упрощаются.

Глобальная модель освещения с трассировкой лучей

Мы уже касались ранее понятия трассировки лучей при описании алгоритмов удаления невидимых граней. Теперь рассмотрим аналогичную процедуру в применении к моделям освещения. В предыдущей главе были описаны модели освещенности от некоторого источника света без учета того, что сами объекты сцены освещают друг друга посредством отраженных лучей. Метод излучательности, разработанный для диффузной модели освещенности, уже учитывает этот фактор.

Глобальная модель освещенности способна воспроизводить эффекты зеркального отражения и преломления лучей (прозрачность и полупрозрачность), а также затенение. Она является составной частью алгоритма удаления невидимых поверхностей методом трассировки.

Если рассмотреть сцену, содержащую в числе прочих зеркальные и полупрозрачные поверхности (рис. 10.3), то изображение будет включать, во-первых, проекции самих объектов, освещенных одним или несколькими источниками света. В некоторых своих частях эти объекты будут искажены за счет преломления лучей в прозрачных и полупрозрачных телах. Во- вторых, часть объектов будет отражаться зеркальными поверхностями, и эти отражения появятся на проекциях зеркальных объектов. В изображенной на рис. 10.3 сцене точки на поверхности призмы $C,\ D$ видны на картинной плоскости дважды: один раз - сквозь полупрозрачный параллелепипед в виде точек $\tilde{C},\ \tilde{D}$, а второй раз - как дважды отраженные невидимой поверхностью параллелепипеда и зеркалом $C'',\ D''$. Параллелепипед в данном случае частично обладает зеркальными свойствами.



увеличить изображение

Рис. 10.3. Сцена, содержащая зеркальные и полупрозрачные поверхности

Глобальная модель освещения для каждого пикселя изображения определяет его интенсивность. Будем для простоты считать, что все источники света - точечные. Сначала определяется непосредственная освещенность источниками без учета отражений от других поверхностей (вторичная освещенность): отслеживаются лучи, направленные ко всем источникам. Тогда наблюдаемая интенсивность (или отраженная точкой энергия) выражается следующим соотношением:

$$I = k_0 I_0 + k_d \sum_j I_j (\overrightarrow{n} \cdot \overrightarrow{l}_j) + k_r \sum_j I_j (\overrightarrow{s} \cdot \overrightarrow{r}_j)^\beta + k_r I_r + k_t I_t,$$

где

k_0 - коэффициент фонового (рассеянного) освещения;

k_d - коэффициент диффузного отражения;

k_r - коэффициент зеркального отражения;

k_t - коэффициент пропускания;

$\overrightarrow{n}$ - единичный вектор нормали к поверхности в точке;

$\overrightarrow{l}_j$ - единичный вектор, направленный к j -му источнику света;

$\overrightarrow{s}$ - единичный локальный вектор, направленный в точку наблюдения;

$\overrightarrow{r}_j$ - отраженный вектор $\overrightarrow{l}_j$;

I_0 - интенсивность фонового освещения;

I_j - интенсивность j -го источника света;

I_r - интенсивность, приходящая по зеркально отраженному лучу;

I_t - интенсивность, приходящая по преломленному лучу.

В алгоритме удаления невидимых линий трассировка луча продолжалась до первого пересечения с поверхностью. В глобальной модели освещения этим дело не ограничивается: осуществляется дальнейшая трассировка отраженного и преломленного лучей. Таким образом, происходит разветвление алгоритма в виде двоичного дерева. Процесс продолжается до тех пор, пока очередные лучи не останутся без пересечений. Отражение и преломление рассчитываются по законам геометрической оптики, которые уже рассматривались в предыдущей главе.

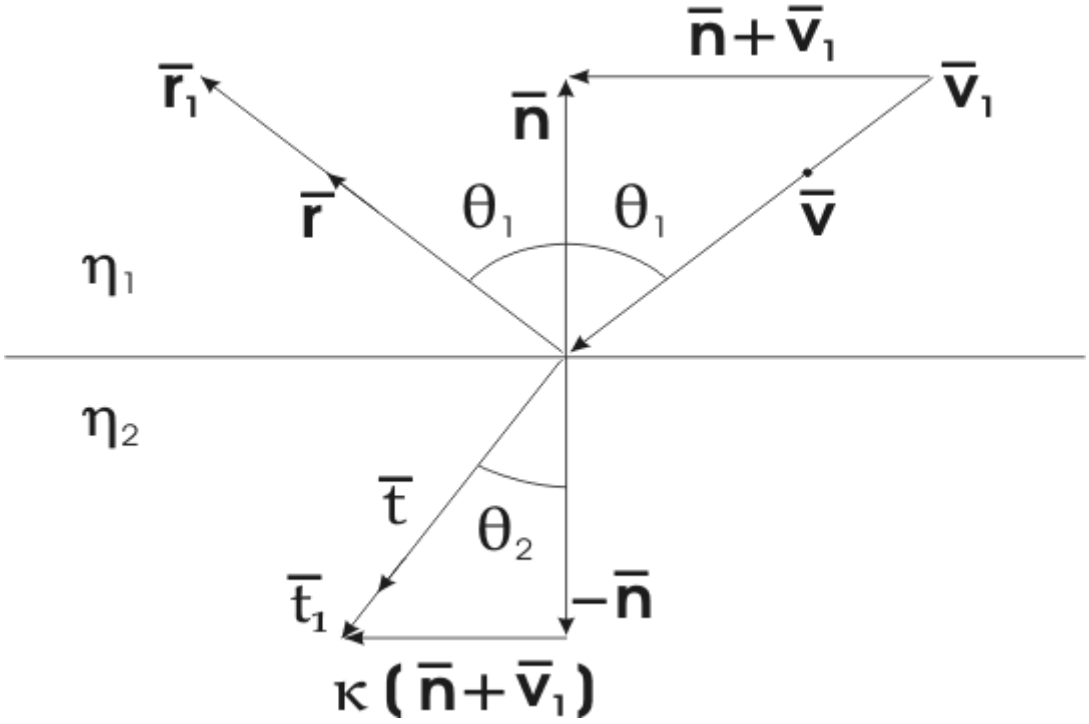


Рис. 10.4. Зеркальное отражение и преломление

Пусть $\overrightarrow{v}, \overrightarrow{r}, \overrightarrow{t}$ - направления падающего, отраженного и преломленного лучей (рис. 10.4), $\overrightarrow{v}_1 = \overrightarrow{v} / \cos(\theta_1)$, $\overrightarrow{n}$ - единичная внешняя нормаль, η_1, η_2 - коэффициенты преломления сред, разделенных поверхностью. Тогда можно показать, что

$$\overrightarrow{r}_1 = \overrightarrow{v}_1 + 2 \cdot \overrightarrow{n}, \quad \overrightarrow{t}_1 = \kappa(\overrightarrow{n} + v_1) - \overrightarrow{n},$$
$$\kappa = (k_\eta^2 \cdot |\overrightarrow{v}_1|^2 - |\overrightarrow{v}|_1 + \overrightarrow{n}^2)^{-1/2}, \quad k_\eta = \frac{\eta_2}{\eta_1}.$$

Соответствующие единичные векторы получить нетрудно.

Двоичное дерево лучей можно строить по принципу "левое поддерево соответствует отраженному лучу, а правое - преломленному". После того как оно построено, можно вычислить интенсивность в точке. Для этого осуществляется обратный проход от вершин к корню, и при прохождении узлов интенсивность убывает.

Теоретически дерево может оказаться бесконечным, поэтому при его построении желательно задать максимальную глубину, чтобы избежать переполнения памяти компьютера.

Поскольку значительная часть лучей, исходящая от источников света и других поверхностей, не попадает в поле зрения наблюдателя, то отслеживать их все не имеет смысла. Поэтому для формирования изображения используется обратная трассировка, т. е. лучи отслеживаются в обратном порядке: от положения наблюдателя через все точки картинной плоскости к объектам и далее - по отраженным и преломленным лучам.

Текстуры

Текстура поверхности - это детализация ее строения, учитывающая микрорельеф и особенности окраски. Во-первых, гладкая поверхность может быть покрыта каким-либо узором, и тогда при ее изображении решается задача отображения этого узора на проекции фрагментов поверхности (многоугольники). Во-вторых, поверхность может

быть шероховатой, поэтому нужны специальные приемы имитации такого микрорельефа при закрашивании.

Сначала рассмотрим методы отображения узоров. Чаще всего узор задается в виде образца, заданного на прямоугольнике в декартовой системе координат η, ξ в пространстве текстуры. Фрагмент поверхности может быть задан в **параметрическом виде** в трехмерной декартовой системе координат:

$$x = f(u,v), \quad y = g(u,v), \quad z = h(u,v), \quad a \le u \le b, \quad c \le v \le d.$$

Теперь достаточно построить отображение области в пространстве текстуры в область параметров поверхности

$$u = \varphi(\eta,\xi), \quad v = \psi(\eta,\xi),$$

или

$$\eta = \chi(u,v), \quad \xi = \theta(u,v),$$

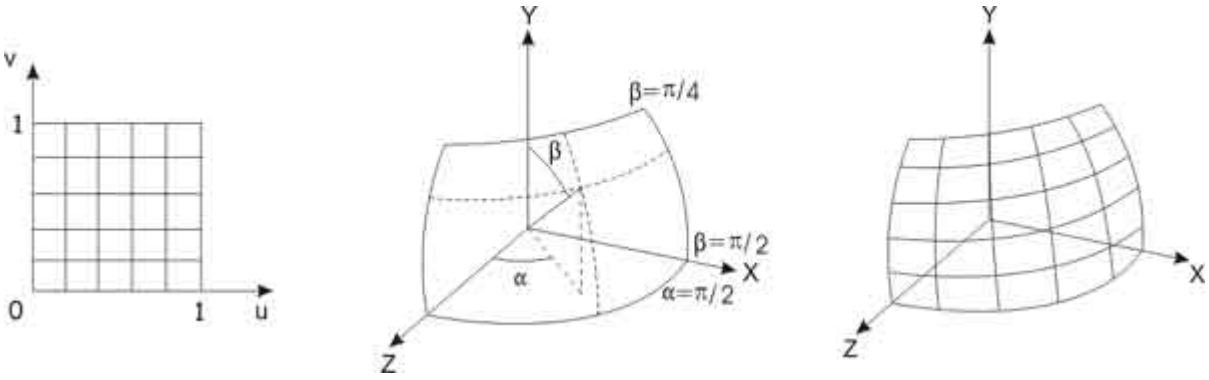
и тем самым каждой точке поверхности будет соответствовать точка образца текстуры. Пусть, например, поверхность представляет собой один октант сферы единичного радиуса, заданный формулами

$$x = \sin\alpha \cdot \sin\beta, \quad y = \cos\beta, \quad z = \cos\alpha \cdot \sin\beta,$$
$$0 \le \alpha \le \pi/2, \quad \pi/4 \le \beta \le \pi/2$$

а образец текстуры задан на квадрате $0 \le \eta \le 1, \quad 0 \le \xi \le 1$. Тогда можно воспользоваться линейным отображением вида

$$\alpha = a\eta + b, \quad \beta = c\eta + d.$$

Если положить $a = \pi/2, \quad b = 0, \quad c = -\pi/4, \quad d = \pi/2$, то углы образца отобразятся в углы криволинейного четырехугольника, как это показано на рис. 10.6.



увеличить изображение

Рис. 10.5. Текстура на сферической поверхности

Обратное отображение имеет вид

$$\eta = \frac{\alpha}{\pi/2}, \quad \xi = \frac{\pi/2 - \beta}{\pi/4},$$

следовательно, вертикальные и горизонтальные линии образца отобразятся на окружности большого круга сферы.

Пусть теперь нужно нанести текстуру при перспективном проецировании произвольно ориентированной прямоугольной грани. Грань задана в пространстве набором своих

вершин A, B, C, D . Построим векторы $\overrightarrow{e}_1 = B - A$ и $\overrightarrow{e}_2 = D - A$, направленные вдоль сторон прямоугольника. Любую точку прямоугольника можно единственным образом представить в виде

$$P = A + u\overrightarrow{e}_1 + v\overrightarrow{e}_2.$$

Будем считать, что используется простейший случай перспективного преобразования, задаваемый формулами

$$x' = \frac{x}{z}, \quad y' = yz.$$

Найдем образ точки P при таком преобразовании:

$$x' = \frac{A_x + u\overrightarrow{e}_{1x} + v\overrightarrow{e}_{2x}}{A_z + u\overrightarrow{e}_{1z} + v\overrightarrow{e}_{2z}}, \quad y' = \frac{A_y + u\overrightarrow{e}_{1y} + v\overrightarrow{e}_{2y}}{A_z + u\overrightarrow{e}_{1z} + v\overrightarrow{e}_{2x}},$$

или

$$\begin{cases} u(x'\overrightarrow{e}_{1z} - \overrightarrow{e}_{1x}) + v(x'\overrightarrow{e}_{2z} - \overrightarrow{e}_{2x}) = A_x - A_z x' \\ \\ u(y'\overrightarrow{e}_{1z} - \overrightarrow{e}_{1y}) + v(y'\overrightarrow{e}_{2z} - \overrightarrow{e}_{2y}) = A_y - A_z y' \end{cases}.$$

Если теперь рассматривать эти соотношения как систему уравнений для нахождения параметров u, v , то, решив ее, получим требуемое обратное преобразование. Для решения можно воспользоваться, например, правилом Крамера:

$$u = \Delta_u/\Delta, \quad v = \Delta_v/\Delta$$

где

$$\Delta = (x'\overrightarrow{e}_{1z} - \overrightarrow{e}_{1x}) \cdot (y'\overrightarrow{e}_{2z} - \overrightarrow{e}_{2y}) - (x'\overrightarrow{e}_{2z} - \overrightarrow{e}_{2x}) \cdot (y'\overrightarrow{e}_{1z} - \overrightarrow{e}_{1y});$$

$$\Delta_1 = (A_x - A_z x') \cdot (y'\overrightarrow{e}_{2z} - \overrightarrow{e}_{2y}) - (x'\overrightarrow{e}_{2z} - \overrightarrow{e}_{2x}) \cdot (A_y - A_z y');$$

$$\Delta_2 = (x'\overrightarrow{e}_{1z} - \overrightarrow{e}_{1x}) \cdot (A_y - A_z y') - (A_x - A_z x') \cdot (y'\overrightarrow{e}_{1z} - \overrightarrow{e}_{1y}).$$

Найденные параметры будут определять точку текстуры, соответствующую точке проекции.

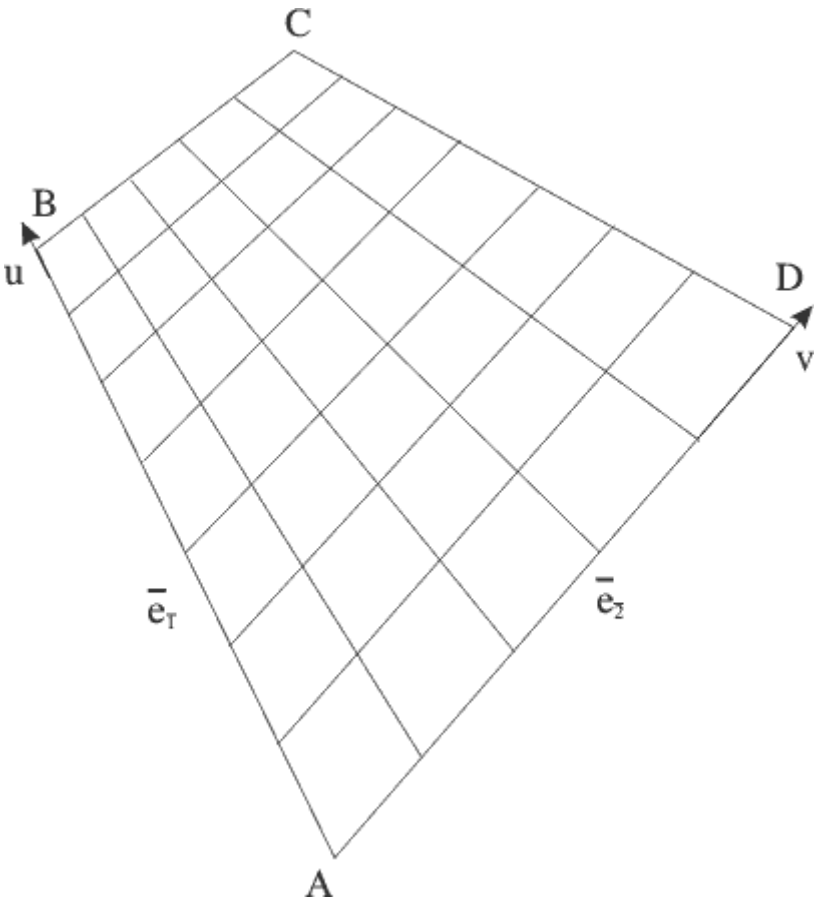


Рис. 10.6. Текстура при перспективной проекции

Можно рассмотреть более общий случай перспективной проекции, задаваемый соотношениями

$$x' = \frac{x}{1+\frac{z}{d}}, \quad y' = \frac{y}{1+\frac{z}{d}}$$

Тогда уравнения для определения u, v немного усложнятся:

$$\begin{cases} u(x'\overrightarrow{e}_{1z}/d - \overrightarrow{e}_{1x}) + v(x'\overrightarrow{e}_{2z}/d - \overrightarrow{e}_{2x}) = A_x - (1 + A_z/d)x' \\ u(y'\overrightarrow{e}_{1z}/d - \overrightarrow{e}_{1y}) + v(y'\overrightarrow{e}_{2z}/d - \overrightarrow{e}_{2y}) = A_y - (1 + A_z/d)y' \end{cases}.$$

Соответственно, изменится и решение:

$$\Delta = (x'\overrightarrow{e}_{1z}/d - \overrightarrow{e}_{1x}) \cdot (y'\overrightarrow{e}_{2z}/d - \overrightarrow{e}_{2y}) -$$
$$- (x'\overrightarrow{e}_{2z}/d - \overrightarrow{e}_{2x}) \cdot (y'\overrightarrow{e}_{1z}/d - \overrightarrow{e}_{1y});$$
$$\Delta_1 = (A_x - (1 + A_z/d)x') \cdot (y'\overrightarrow{e}_{2z}/d - \overrightarrow{e}_{2y}) -$$
$$- (x'\overrightarrow{e}_{2z}/d - \overrightarrow{e}_{2x}) \cdot (A_y - (1 + A_z/d)y')$$
$$\Delta_2 = (x'\overrightarrow{e}_{1z}/d - \overrightarrow{e}_{1x}) \cdot (A_y - (1 + A_z/d)y') -$$
$$- (A_x - (1 + A_z/d)x') \cdot (y'\overrightarrow{e}_{1z}/d - \overrightarrow{e}_{1y}).$$

В рассмотренных примерах мы имели дело с гладкими поверхностями. Можно имитировать шероховатость путем выбора подходящего образца нерегулярной текстуры, но все равно изображение будет выглядеть так, словно неоднородности нанесены на **гладкой** поверхности. Для моделирования микрорельефа Дж.Блин предложил метод, основанный на возмущении нормали к поверхности.

Пусть, как и ранее, поверхность задана в параметрическом виде с помощью векторной функции $\overrightarrow{F}(u,v)$. В каждой ее точке можно построить вектор нормали, воспользовавшись частными производными этой функции. Известно, что производные $\overrightarrow{F}_u$ и $\overrightarrow{F}_v$ представляют собой векторы, лежащие в касательной плоскости данной поверхности. Тогда вектор нормали может быть получен как векторное произведение этих двух векторов $\overrightarrow{n} = \overrightarrow{F}_u \times \overrightarrow{F}_v$. После этого точку поверхности можно отклонить от первоначального положения в направлении нормали на некоторую малую величину, задаваемую с помощью функции возмущения $P(u,v)$:

$$\overrightarrow{F}'(u,v) = \overrightarrow{F}(u,v) + P(u,v) \cdot \overrightarrow{n}(u,v).$$

Можно показать, что нормаль к новой возмущенной поверхности будет определяться выражением

$$\overrightarrow{n}' = \overrightarrow{n} + \frac{P_u \cdot (\overrightarrow{n} \times \overrightarrow{F}_u)}{|\overrightarrow{n}|} + \frac{P_v \cdot (\overrightarrow{n} \times \overrightarrow{F}_v)}{|\overrightarrow{n}|}.$$

Применяя в модели освещения новую нормаль, можно получить эффект шероховатости поверхности. В качестве функции возмущения можно использовать произвольную дифференцируемую по каждой из переменных функцию.

Вопросы и упражнения

1. Какие этапы выделяются в свето-теневом анализе?
2. К какому типу относится алгоритм Аппеля: итеративному или рекурсивному?
3. Возможно ли использование алгоритма Аппеля для сцен с неполным затенением?
4. Что такое теневой буфер? Чем он отличается от традиционного Z-буфера?
5. В чем состоит модификация алгоритма Вейлера-Азертона для выполнения свето-теневого анализа?
6. В какой модели освещенности можно использовать метод излучательности?
7. Чем отличается трассировка лучей в глобальной модели освещенности от метода удаления невидимых граней?
8. Какие составляющие интенсивности рассматриваются в методе трассировки?
9. Каким образом можно использовать двоичные деревья в алгоритме трассировки?
10. Какой способ задания поверхности наиболее удобен для текстурирования?
11. В чем состоит идея моделирования микрорельефа при нанесении текстур?

Литература

1. Ильин В.А., Позняк Э.Г, Аналитическая геометрия, М.: Наука, 1981
2. Фокс А., Пратт М, Вычислительная геометрия. Применение в проектировании и на производстве, М.: Мир, 1982
3. Фоли Дж., ван Дэм А, Основы интерактивной машинной графики. Кн. 1, 2, М.: Мир, 1985
4. Ньюмен У., Спрул Р, Основы интерактивной машинной графики, М.: Мир, 1985
5. Роджерс Д, Алгоритмические основы машинной графики, М.: Мир, 1989

6. Шикин Е.В., Боресков А.В, Компьютерная графика. Динамика, реалистические изображения, М.: ДИАЛОГ-МИФИ, 1995
7. Вельтмандер П.В, Машинная графика. (Учебное пособие), Новосибирск, Новосибирский Государственный технический университет, 1997
8. Шикин Е.В., Боресков А.В, Компьютерная графика. Полигональные модели, М.: ДИАЛОГ-МИФИ. 2001
9. Эйнджел Э, Интерактивная компьютерная графика. Вводный курс на базе OpenGL, М., СПб., Киев: Изд. дом "Вильямс", 2001
10. Ward M, A (Spotty) History and Who's Who of Computer Graphics, WPI CS Department
11. , What is Computer Graphics?,