Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Т. О. Перемитина

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебное пособие

Томск
«Эль Контент»
2012

УДК 004.92(075.8)
ББК 32.973.26-018.2я73
П 270

Рецензенты:
Тарасенко В. Ф., докт. техн. наук, профессор кафедры теоретической кибернетики факультета прикладной математики и кибернетики Томского государственного университета;
Сенченко П. В., канд. техн. наук, декан факультета систем управления, доцент кафедры автоматизации обработки информации ТУСУР.

Перемитина Т. О.
П 270 Компьютерная графика : учебное пособие / Т. О. Перемитина. — Томск : Эль Контент, 2012. — 144 с.

ISBN 978-5-4332-0077-7

В учебном пособии рассматриваются методы и алгоритмы современной компьютерной графики, излагаются основы прикладной компьютерной графики, приводятся наиболее важные термины и определения в области компьютерной графики. Технические вопросы снабжены комментариями и иллюстрациями. Пособие предназначено для самостоятельной работы студентов при изучении дисциплины «Компьютерная графика».

УДК 004.92(075.8)
ББК 32.973.26-018.2я73

ISBN 978-5-4332-0077-7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В основании бесконечно развивающегося здания компьютерной графики краеугольными камнями лежат фундаментальные дисциплины — аналитическая геометрия и физика, подкрепленные искусством программирования. Возникнув из потребностей рынка, развития информатики и вычислительной техники, компьютерная графика изучает методы построения изображений различных геометрических объектов и сцен.

Главными этапами построения изображения являются:

- Моделирование как искусство применения методов математического описания объектов и сцен, природа которых может быть самой различной: обычные геометрические фигуры и тела в двух- и трехмерном пространстве; естественные явления природы — горы, деревья, облака и другие объекты; огромные массивы чисел, полученных в различных экспериментах, и многое другое.
- Визуализация (отображение) как искусство построения реалистичных изображений объемного мира на плоском экране дисплея ЭВМ заключается в преобразовании моделей объектов и сцен в статическое изображение или фильм — последовательность статических кадров.

В отличие от плоских изображений реальных трехмерных объектов, которые на фотографиях автоматически выглядят натурально благодаря действию в природе оптических законов, синтезированное компьютерное изображение будет похоже на реальное лишь при мастерском владении как методами моделирования геометрических форм, так и средствами их достоверной визуализации. Таким образом, компьютерную графику можно рассматривать как искусство создания реалистичной иллюзии действительного мира. Эта сторона ее применения нашла наиболее яркое воплощение в системах виртуальной реальности — комплексах аппаратных и программных средств имитации окружающей среды с помощью визуальных, акустических, тактильных и других эффектов.

Целью изучения курса «Компьютерная графика» является изучение математических и алгоритмических основ компьютерной графики, а также освоение средств разработки программного обеспечения для визуализации реалистичных изображений сложных трехмерных сцен.

Задачи дисциплины:

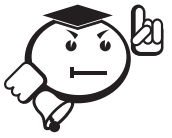
- формирование взглядов на компьютерную графику как на систематическую научно-практическую деятельность, носящую как теоретический, так и прикладной характер;
- ознакомление с базовыми теоретическими понятиями, лежащими в основе компьютерной графики, изучение особенностей растровых и векторных изображений;
- получение представления о методах геометрического моделирования;
- практическое использование алгоритмов и методов компьютерной графики при проектировании пользовательских интерфейсов программных систем;
- создание программ для работы с графикой, обеспечивающих пользователей широким арсеналом инструментов — от простейших графических редакторов до сложных систем трехмерного моделирования.

Соглашения, принятые в книге

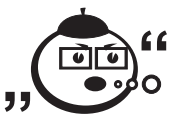
Для улучшения восприятия материала в данной книге используются пиктограммы и специальное выделение важной информации.

Эта пиктограмма означает определение или новое понятие.

Эта пиктограмма означает внимание. Здесь выделена важная информация, требующая акцента на ней. Автор здесь может поделиться с читателем опытом, чтобы помочь избежать некоторых ошибок.



В блоке «На заметку» автор может указать дополнительные сведения или другой взгляд на изучаемый предмет, чтобы помочь читателю лучше понять основные идеи.



Эта пиктограмма означает цитату.

Выводы

Эта пиктограмма означает выводы. Здесь автор подводит итоги, обобщает изложенный материал или проводит анализ.

Контрольные вопросы по главе

Глава 1

ОСНОВНЫЕ ПОНЯТИЯ КОМПЬЮТЕРНОЙ ГРАФИКИ

1.1 Определение и задачи компьютерной графики

За последние несколько десятилетий компьютерная графика прошла путь от вычерчивания простых линий и отрезков до построения виртуальной реальности и создания полнометражных кинофильмов. Само слово «графика» в его привычном понимании уже не соответствует той области интересов, которую охватывает компьютерная графика [9]. На сегодняшний день компьютерная графика — это область информатики, в сферу интересов которой входят все аспекты формирования изображений с помощью компьютеров. Уточняя, можно сказать, что предметом ее изучения является создание, хранение и обработка моделей и их изображений с помощью ЭВМ.

Специальную область информатики, занимающуюся методами и средствами создания изображений с помощью программно-аппаратных вычислительных комплексов, называют ***компьютерной (машинной) графикой****.*

Интерактивная компьютерная (машинная) графика — понятие, которое используется для того, чтобы подчеркнуть наличие аппаратных и программных способов диалога с человеком в графической компьютерной системе [13].

Существует более строгое определение понятия «машинная графика», даваемое Международной организацией по стандартизации ISO (International Standard Organization): *Машинная графика* — это совокупность методов и средств для преобразования данных в графическую форму представления и из графической формы представления с помощью электронно-вычислительной машины [8].

Задачи КГ — воспроизведение (синтез) изображений в тех случаях, когда исходная информация имеет неизобразительную природу.

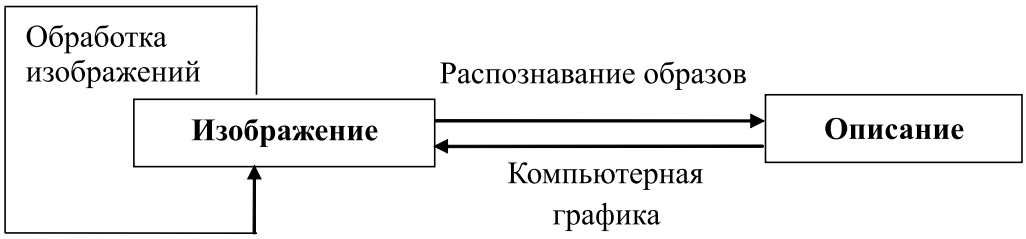


Рис. 1.1 – Основные направления работы с изображением

При обработке информации, связанной с изображением на мониторе, принято выделять три основных направления (рис. 1.1):

- распознавание образов;
- обработку изображений;
- компьютерную графику.

Основная задача распознавания образов состоит в преобразовании уже имеющегося изображения на формально понятный язык символов.

***Распознавание образов**, или система технического зрения, (computer vision) — совокупность методов, позволяющих получить описание изображения, поданного на вход, либо отнести заданное изображение к некоторому классу (так поступают, например, при сортировке почты).*

***Обработка изображений** (image processing) рассматривает задачи, в которых и входные, и выходные данные являются изображениями.*

Например, передача изображения с устранением шумов и сжатием данных, переход от одного вида изображения к другому (от цветного к черно-белому) и т. д.

Компьютерная графика *(computer graphics) воспроизводит изображение в случае, когда исходной является информация неизобразительной природы.*

Например, визуализация экспериментальных данных в виде графиков, гистограмм или диаграмм, создание изображений в компьютерных играх, синтез сцен на тренажерах.

1.2 История развития и области применения компьютерной графики

Развитие компьютерной графики определяется двумя факторами: потребностями потенциальных пользователей и достижениями в области аппаратного и программного обеспечения.

Можно считать, что первые системы КГ появились вместе с первыми цифровыми компьютерами. Сейчас ее рассматривают как средство, которое обеспечивает мощную взаимосвязь между человеком и компьютером, заставляя компьютер говорить с человеком на языке изображений.

Началом эры компьютерной графики можно считать декабрь 1951 года, когда в Массачусетском технологическом институте (МТИ) для системы противовоздушной обороны военно-морского флота США был разработан первый дисплей для компьютера «Вихрь» (Whirl). Изобретателем этого дисплея был Джей Форрестер, работавший инженером в МТИ.

Одним из отцов-основателей компьютерной графики считается Айвен Сазерленд, который впервые в 1962 году все в том же МТИ создал программу компьютерной графики под названием «Блокнот» (Sketchpad). Эта программа могла рисовать достаточно простые фигуры (точки, прямые, дуги окружностей), могла вращать фигуры на экране. После этой программы некоторые крупные фирмы, такие, как General Motors, General Electric, приступили к разработкам в области компьютерной графики.

Если в 1950-х годах КГ давала возможность выводить лишь несколько десятков отрезков на экране, то к середине 60-х наступил период плодотворной работы и в промышленных приложениях КГ. Под руководством Тирбера Мофетта и Нормана Тейлора фирма Itek разработала цифровую электронную чертежную машину. В 1964 году General Motors представила свою систему автоматизированного проектирования DAC-1, разработанную совместно с IBM. К октябрю 1966 года журнал Wall Street уже публиковал статьи по КГ.

В конце 70-х в КГ произошли значительные изменения. Появилась возможность создания растровых дисплеев, имеющих множество преимуществ: вывод больших массивов данных, устойчивое, немерцающее изображение, работа с цветом и недорогие мониторы. Растровая технология в конце семидесятых стала явно доминирующей.

В 80-х годах определяющую роль сыграл выпуск компанией Apple компьютеров Macintosh. Они были для своего времени настоящей революцией. Во-первых,

Macintosh серийно поставлялся с цветным монитором. Во-вторых, его операционная система обладала наглядным, визуальным интерфейсом (своего рода аналог более поздней ОС Windows). В-третьих, их мощности было достаточно для обработки графических изображений. Именно поэтому Macintosh сразу заслужил внимание множества профессиональных художников и дизайнеров, которые поменяли карандаш и кисть на мышь и клавиатуру. Рынок не заставил себя долго ждать — появилось несколько очень впечатляющих для своего времени графических редакторов.

В XXI веке 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 [14].

Совершенно очевидно, что в новом тысячелетии информационные и коммуникационные технологии будут играть важнейшую роль во всех сферах жизни человечества. В таких областях, как кинематография, издательское дело, в научных исследованиях, в образовательных учреждениях, внедрение этих технологий уже произвело поистине революционный переворот.

Области приложения компьютерной графики в настоящее время очень широки. В промышленности используется компьютерное моделирование процессов с графическим отображением происходящего на экране [21].

Разработка новых автомобилей проходит на компьютере от стадии первичных эскизов внешнего вида корпуса автомобиля до рассмотрения поведения деталей автомобиля в различных дорожных условиях.

В медицине применяются компьютерные томографы, позволяющие заглянуть внутрь тела и поставить правильный диагноз.

В архитектуре широко применяются системы визуального автоматизированного проектирования (CAD — Computer Aided Design), которые позволяют разработать проект нового здания, основываясь на методах компьютерной графики.

Химики изучают сложные молекулы белков, пользуясь компьютерными средствами визуального отображения данных.

В кинематографии использование компьютерной графики стало почти необходимым делом. В мире регулярно проводятся выставки, например такие, как SIGGRAPH, картин нарисованных с помощью компьютера.

В математике развитие теории фракталов было бы невозможно без компьютеров с соответствующими средствами графического отображения данных. Средства мультимедиа привели к возможности совместного использования различных источников информации, объединяющих в себе статические и видеоизображения, текст и звук. Новейшие операционные системы работают в графическом режиме и изначально реализуют в своих функциях методы компьютерной графики.

Формально можно выделить *четыре главные области* применения компьютерной графики:

1. *Отображение информации (визуализация)*. На протяжении многих столетий картографы и астрономы вычерчивали карты, чтобы представить информацию о расположении небесных тел и географических областей.

Визуализация *(Rendering) — создание плоских изображений трехмерных (3D) моделей.*

2. *Проектирование*. Проектирование является одной из основных стадий создания изделий и сооружений в технике и строительстве.

Проектирование *— это процесс, в ходе которого создается прототип, прообраз необходимого объекта.*

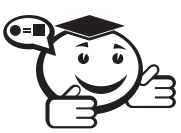
В строительстве и технике чертежи давно представляют собой основу проектирования новых сооружений или изделий. Процесс проектирования с необходимостью является итеративным, т. е. конструктор перебирает множество вариантов с целью выбора оптимального по каким-либо параметрам. Не последнюю роль в этом 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*. Как только графические системы стали обладать достаточной производительностью для создания сложных динамических изображений, возникла идея применять их в качестве средства моделирования реальной обстановки (симулятора) на разного рода тренажерах.

Моделирование *(modeling) — создание и представление трехмерных (3D) моделей.*

Первыми такие системы освоили авиаторы. Это позволило значительно снизить стоимость обучения, гарантируя при этом его высокое качество. В телевидении, кинематографии и рекламном деле в последнее время также широко используются средства КГ. Появилась еще одна область применения средств КГ — формирование виртуальной реальности (VR). С помощью такой системы хирург может отработать методику операции, астронавт может подготовиться к выходу в открытый космос и др.

4. *Пользовательский интерфейс*. В последнее время визуальная парадигма стала доминирующей в сфере взаимодействия пользователя с компьютером.



Интерфейс *— это совокупность средств и методов обеспечения взаимодействия между элементами системы.*

Пользовательский интерфейс *— элементы и компоненты программы, способные оказывать влияние на взаимодействие пользователя с программным обеспечением.*

В том числе:

- средства отображения информации;
- командные режимы, язык «пользователь-интерфейс»;
- устройства и технологии ввода данных;
- диалоги между пользователем и компьютером;
- обратная связь с пользователем и т. д.

Графический пользовательский интерфейс *(graphical user interface, GUI) — обеспечивает возможность управления поведением вычислительной системы через визуальные элементы управления — окна, списки, кнопки, гиперссылки и т. д.*

Первые операционные системы использовали способ взаимодействия через командную строку.

Взаимодействие оператора с вычислительной машиной является важным звеном вычислительного процесса при решении различных прикладных задач как научного, так и производственного плана. Визуальный метод предполагает использование различного рода окон, меню и устройств указания, таких как мышь.

1.3 Графическая система

Система компьютерной графики является, прежде всего, вычислительной системой и включает все компоненты вычислительной системы общего назначения [21]. Основные компоненты системы приведены на рис. 1.2.

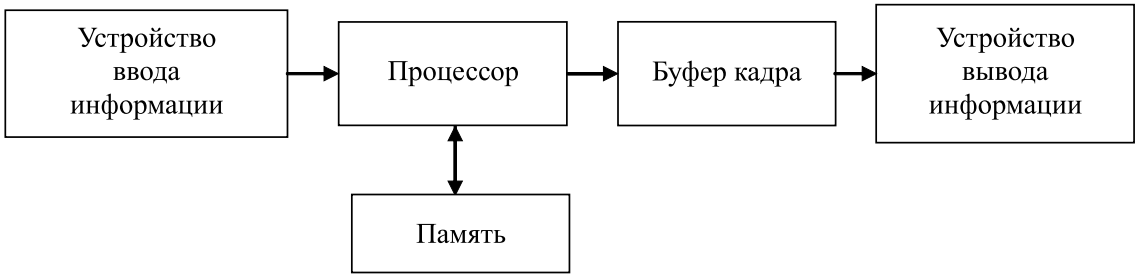


Рис. 1.2 – Основные компоненты графической системы

Буфер кадра *— часть графической памяти для хранения массива кодов, определяющих засветку пикселей на экране.*

Эта модель имеет общий характер и отображает структуру и графической рабочей станции, и персонального компьютера, и графического терминала большой вычислительной системы, работающей в режиме разделения машинного времени, и интеллектуальной системы формирования изображений. К вопросам стандартизации в КГ вплотную приступили в конце 70-х годов. Тогда и был предложен первый проект GSPC CORE SYSTEM, разработанный американским комитетом по стандартизации в машинной графике GSPC ACMSIGGRAPH, оказавший большое влияние на разработку первого Международного стандарта Graphical Kernel System (GKS) — Ядро Графической Системы (ЯГС).

1.3.1 Ядро графической системы

Ядро графической системы (ЯГС) является базисной графической системой, которая может использоваться для решения большинства прикладных задач, генерирующих изображение с помощью ЭВМ. Функциональное описание ядра графической системы представлено в ГОСТ 27817-88 [4].

Ядро графической системы *представляет собой функциональный интерфейс между прикладной программой и конфигурацией графических устройств ввода и вывода.*

Функциональный интерфейс содержит все основные функции для интерактивной и неинтерактивной графики и применим для широкого диапазона графического оборудования. Модель функционирования ЯГС в графической системе представлена на рис. 1.3. Каждый уровень программ может вызывать функции примыкающих нижних уровней. В основном прикладная программа будет использовать проблемно-ориентированный и языково-зависимый уровни, а также другие зависящие от применения уровни и ресурсы операционной системы.

Основные цели, для достижения которых ЯГС вводится как международный стандарт:

- обеспечение возможности переноса прикладных программ с одной установки на другую;
- оказание помощи прикладным программистам в понимании и применении графических методов;
- определение направлений развития графического оборудования путем создания руководства для обеспечения полезных комбинаций графических возможностей в устройстве.

Одним из основных понятий ЯГС является понятие графической рабочей станции, состоящей из набора устройств ввода и одного устройства вывода. Несколько

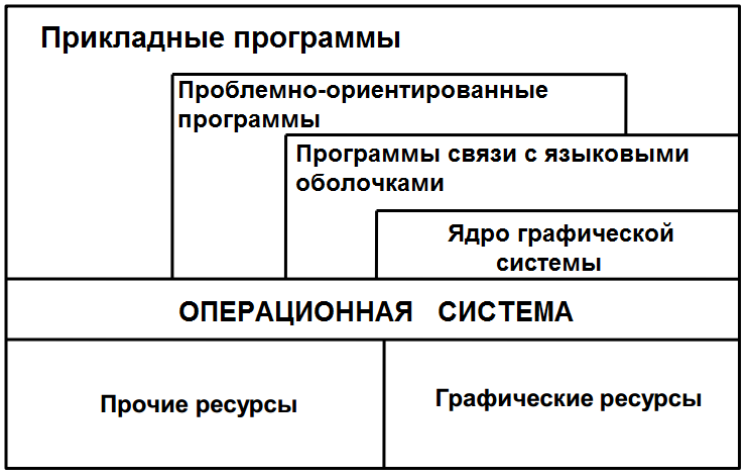


Рис. 1.3 – Модель функционирования ядра графической системы

станций могут использоваться параллельно. ЯГС основывается на концепции абстрактных графических станций, обеспечивающих логический интерфейс, с помощью которого прикладная программа управляет физическими устройствами.

ЯГС не зависит от особенностей языков программирования. Для связи с языками программирования необходимо описать имена функций и типы данных в терминах языка программирования.

Реализация ядра графической системы дает возможность пользователям машинной графики разрабатывать прикладные программы, которые будут переноситься между различными вычислительными системами и различными графическими устройствами.

1.3.2 Пользователи графических систем

Существует три важных класса пользователей графических систем [21]:

- *Разработчик* создает графическую систему, использует базовое программное обеспечение. Задача — обеспечение доступа прикладного программиста к возможностям графических устройств.
- *Прикладной программист* использует систему компьютерной графики, вызывая из своих программ графические функции.
- *Оператор графической системы* — лицо, которое взаимодействует с графической программой путем физического воздействия на устройство ввода.

Интерфейс между прикладной программой и графической системой — это множество функций, которые в совокупности образуют графическую библиотеку. Спецификация этих функций и есть то, что мы называем интерфейсом прикладного программирования (API — application programmer's interface). Для программиста, занимающегося разработкой прикладной программы, существует только API, и он избавлен от необходимости вникать в подробности работы аппаратуры и программной реализации функций графической библиотеки.

1.4 Методы представления графической информации

Несмотря на то, что для работы с компьютерной графикой существует множество классов программного обеспечения, различают всего три вида компьютерной графики. Это растровая графика, векторная графика и фрактальная графика. Они отличаются принципами формирования изображения при отображении на экране монитора или при печати на бумаге.

В зависимости от способа формирования изображений компьютерную графику принято подразделять на растровую и векторную [12].

1.4.1 Растровая графика

Растровая графика *— способ построения изображений, в котором изображение представляется массивом простейших элементов — пикселей, где каждый пиксель имеет четко заданное положение.*

Растровая графика имеет весьма точный аналог в реальном мире — мозаику. В растровой графике цельное изображение составляется из отдельных элементов, называемых пикселями. Все они одинакового размера и формы, упорядоченно размещены и различаются только цветом. За счет малого размера пиксели не воспринимаются глазом как отдельные объекты, и мы видим только цельное изображение. На рис. 1.4 приведен пример растрового рисунка (на фрагменте справа для наглядности показана растровая сетка, согласно которой размещаются пиксели).

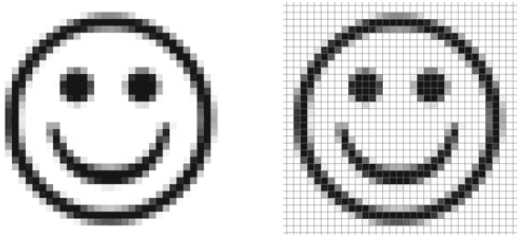


Рис. 1.4 – Пример растрового рисунка

Достоинства растровой графики — растровые редакторы являются наилучшим средством обработки фотографий и рисунков, т. к. обеспечивают высокую точность передачи градаций цветов и полутонов.

Недостатки растровой графики:

- растровая графика чрезвычайно чувствительна к изменению размера рисунка, и масштабировать ее затруднительно;
- хранение и обработка файлов растровой графики требует больших объемов памяти.

Пиксели, составляющие изображение, могут быть разного цвета. Цвет каждого пикселя записывается комбинацией битов. Чем больше битов используется для представления цвета, тем больше цветов и оттенков мы получаем.

Глубина буфера кадра *— величина, характеризующая количество бит информации, определяющих засветку каждого отдельного пикселя, в частности количество цветов, которое может быть представлено на экране данной системы.*

Глубина буфера 1 бит — двухградационное изображение; 8 бит это 2^8 = 256 цветов.

Современные полноцветные системы характеризуются глубиной буфера 24 бита. Это системы с правильной цветопередачей, или RGB системы, поскольку в кодировке засветки каждого пикселя можно выделить отдельные группы битов, характеризующие интенсивность засветки по каждому из основных цветов — красному (Red), зеленому (Green) и синему (Blue).

Размер буфера кадра определяет одну из главных характеристик графической системы — разрешающую способность, или разрешение.

Разрешение изображения *— это количество пикселей на единицу длины.*

При этом следует различать:

- Разрешение оригинала. Разрешение оригинала в мировой практике измеряют в точках на дюйм (dots per inch — dpi).
- Разрешение экранного изображения. Для экранных копий разрешение измеряется в пикселях на дюйм (ppi).
- Разрешение печатного изображения.

1.4.2 Векторная графика

Векторная графика — изображение на основе регулярных структур. Изображения этого типа определяются на основе простейших геометрических понятий — примитивов (точка, отрезок прямой, прямоугольник, треугольник и т. д.).

Векторное изображение *— это тип изображения, которое состоит из геометрических объектов, описанных математически.*

Все эти фигуры задаются определенными наборами параметров. На рис. 1.5 приведен пример векторного рисунка — его внешний вид и «строение» рисунка, состоящего из точек и линий.

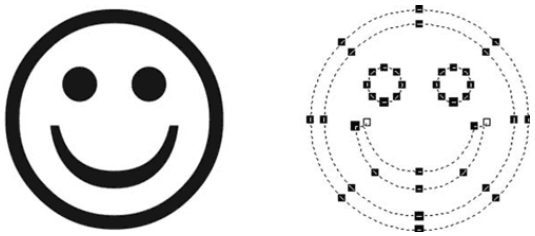


Рис. 1.5 – Пример векторного рисунка

Например, для окружности достаточно задать координаты ее центра, радиус и цвет. Для масштабирования достаточно присвоить соответствующим параметрам новые значения. Таким образом, здесь каждый элементарный объект задается вектором (линией) и некоторыми параметрами.

Узлы также имеют свойства, параметры которых влияют на форму конца линии и характер сопряжения с другими объектами.

Векторные изображения, равно как и растровые, имеют свои плюсы и минусы. Обсудим плюсы векторных изображений:

- можно изменять размеры изображений без потери его визуальных качеств;
- максимальная точность построенного изображения (координаты точек, между которыми могут быть проведены кривые, могут иметь точность до сотых доль микрона);
- файл с векторным изображением имеет значительно меньший размер по сравнению с растровым изображением;
- рисунок имеет высокое качество при печати;
- возможность редактирования всех частей векторного изображения;
- простой экспорт векторного рисунка в растровый.

Есть так же и минусы векторных изображений:

- отсутствие реалистичности у векторных рисунков. Реалистичность достигается путем применения различных сложных цветовых схем;
- невозможность использования эффектов, которые можно применять в растровой графике;
- практически полная невозможность экспорта растрового рисунка в векторный.

1.4.3 Фрактальная графика

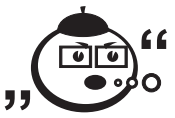
Фрактальная графика основана на автоматической генерации изображений путем математических расчетов. Создание фрактальных изображений основано не в рисовании, а в программировании. Фрактальная графика редко используется в печатных или электронных документах.

Фрактал *— геометрическая фигура, обладающая свойством самоподобия, то есть составленная из нескольких частей, каждая из которых подобна всей фигуре целиком.*

Понятия фрактал и фрактальная геометрия появились в конце 70-х — середине 80-х. Слово «фрактал» образовано от латинского «fractus» и в переводе означает состоящий из фрагментов. Понятие «фрактал» было предложено Бенуа Мандельбротом в 1975 году для обозначения нерегулярных, но самоподобных структур, которыми он занимался. Рождение фрактальной геометрии принято связывать с выходом в 1977 году книги Мандельброта «The Fractal Geometry of Nature». В его работах использованы научные результаты других ученых, работавших в период 1875–1925 годов в той же области (Пуанкаре, Фату, Жюлиа, Кантор, Хаусдорф).

Роль фракталов в КГ сегодня достаточно велика. Они приходят на помощь, например, когда требуется, с помощью нескольких коэффициентов, задать линии и поверхности очень сложной формы. С точки зрения компьютерной графики, фрактальная геометрия незаменима при генерации облаков, гор, поверхности моря. Фактически найден способ легкого представления сложных неевклидовых объектов, образы которых весьма похожи на природные [10, 13].

Определение фрактала, данное Мандельбротом, звучит так:



«Фракталом называется структура, состоящая из частей, которые в каком-то смысле подобны целому.»

Одним из основных свойств фракталов является самоподобие. В самом простом случае небольшая часть фрактала содержит информацию о всем фрактале. Фрактал можно определить как объект довольно сложной формы, получающийся в результате выполнения простого итерационного цикла. Итерационность, рекурсивность обуславливают такие свойства фракталов, как самоподобие — отдельные части похожи по форме на весь фрактал в целом [13].

Фракталом Мандельброта названа фигура, которая порождается очень простым циклом. Для создания этого фрактала необходимо для каждой точки изображения выполнить цикл итераций согласно формуле:

$$z_{k+1} = z_k^2 + z_0,\ k = 0, 1, \ldots, n.$$

Величины z_k — это комплексные числа, $z_k = x_k + i \cdot y_k$, причем стартовые значения x_0 и y_0 — это координаты точки изображения. Для каждой точки изображения итерации выполняются ограниченное количество раз (n) или до тех пор, пока модуль числа z_k не превышает 2. Модуль комплексного числа равен корню квадратному из $x^2 + y^2$. Для вычисления квадрата величины z_k можно воспользоваться формулой:

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

поскольку $i^2 = -1$.

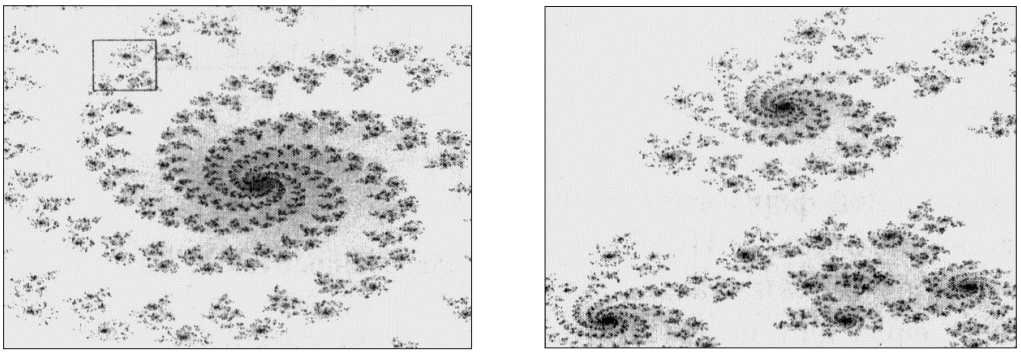
Цикл итераций для фрактала Мандельброта можно выполнять в диапазоне $x = (\text{от } -2.2 \text{ до } 1)$, $y = (\text{от } -1.2 \text{ до } 1.2)$. Для того чтобы получить изображение в растре, необходимо пересчитывать координаты этого диапазона в пиксельные.

Фрактал Джулия внешне совсем не похож на фрактал Мандельброта, однако он определяется итерационным циклом, почти полностью тождественным с циклом генерации Мандельброта. Формула итераций для фрактала Джулия такая:

$$z_{k+1} = z_k^2 + c,$$

где c — комплексная константа. Условием завершения итераций является $|z_k| > 2$ — так же, как для фрактала Мандельброта.

На рис. 1.6 приведено два изображения фрактала Джулия для $c = 0.36 + i \cdot 0.36$, изображение построено в границах $x \in [-1, 1]$, $y \in [-1.2, 1.2]$. На рис. 1.6, *б* показан увеличенный фрагмент фрактала. Как видим, фрактал самоподобный — при любом увеличении отдельные части напоминают формы целого.



а) *б)*

Рис. 1.6 – Пример фрактального изображения

Рассмотрим следующий пример фрактала — *фрактал Ньютон*. Для него итерационная формула имеет такой вид:

$$z_{k+1} = \frac{3z_k^4 + 1}{4z_k^3},$$

где z — также комплексные числа, причем $z_0 = x + i \cdot y$ соответствует координатам точки изображения.

Условием прекращения цикла итераций для фрактала Ньютон есть приближение значений $|z^4 - 1|$ к нулю. Например, изображение на рис. 1.7 было получено для $|z^4 - 1|^2 > 0.001$, границы расчета $x \in [-1, 1]$, $y \in [-1, 1]$.

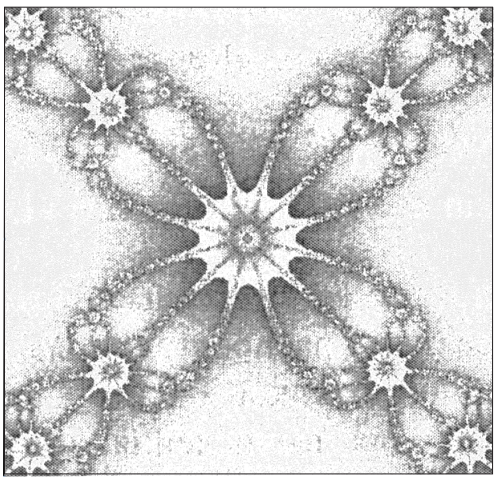


Рис. 1.7 – Фрагмент фрактала Ньютон

1.5 Форматы файлов графики

Рассмотрим некоторые наиболее широко распространенные форматы векторных и растровых изображений [9].

К векторным форматам относятся:

- **.cdr** — основной внутренний формат программы Corel DRAW (внутренний — это означает, что данный формат используется только программой Corel DRAW и никакой больше);
- **.pct** — векторный формат, используемый на компьютерах Apple Macintosh в операционных системах Mac;
- **.ai** — внутренний формат файла для программы Adobe Illustrator;
- **.wmf** (windows metafile) — является «внутренним» форматом ОС Windows на платформе IBM PC. Изначально предназначен для обмена векторными данными между приложениями через буфер обмена. Однако не все программы умеют обрабатывать его код. Результатом при переносе изображений является искажение цветов, неправильная установка толщины контура и т. д. Кроме того, в этот формате нельзя включить растровое изображение.

Все вышеперечисленные форматы поддерживаются программой Corel DRAW.

К растровым форматам относятся:

- **.jpeg / .jpg** (Joint Photographic Expert Group) является методом (алгоритмом) сжатия изображений с потерей части информации. Применение компрессии JPEG позволяет в 500 раз уменьшить объем файла по сравнению с обычным bitmap. Вместе с тем искажение цветовой модели и деградация деталей не позволяют рекомендовать JPEG в качестве средства хранения изображения высокого качества. Обычно JPEG используют для электронных публикаций;
- **.gif** (CompuServe Graphics Interchange Format) разработан в 1987 г. Формат поддерживает функции прозрачности цветов и некоторые виды анимации. Все данные в файле сжимаются методом Lempel — Ziv — Welch (LZW) без потери качества;
- **.png** (Portable network graphics) — растровый формат хранения графической информации. Все данные в файле сжимаются методом Deflate (сжатие без потерь). Формат PNG является аналогом формата GIF (за исключением анимации) и возник после перехода формата GIF в разряд коммерческих продуктов;
- **.bmp** (Windows Device Independent Bitmap) служит для обмена растровыми изображениями между приложениями ОС Windows. Формат поддерживает большинство цветовых моделей, вплоть до 24-битного пространства RGB. Файлы в формате BMP занимают значительный объем. Качество изображений, выводимых на печать, ввиду ограниченности цветовых моделей оставляет желать лучшего. Формат BMP в настоящее время устарел и не имеет каких-либо видимых достоинств, оправдывающих его применение;
- **.tiff** (Tagged Image File Format) считают лучшим вариантом для записи полутоновых изображений. Формат TIFF распознается практически всеми

программами верстки, растровыми и векторными редакторами и позволяет хранить изображения высочайшего качества. Последние версии формата TIFF поддерживают несколько способов сжатия изображения: LZW (без потери информации), ZIP (без потери информации), JPEG (с потерей части информации). Универсальным считают метод сжатия LZW.

1.6 Цветовые модели

Способ разделения цветового оттенка на составляющие компоненты называется ***цветовой моделью****.*

1.6.1 Цветовая модель RGB

Цветовая модель RGB *(Red Green Blue) — аддитивная цветовая модель, согласно которой цвет кодируется тремя компонентами — красным, зеленым и синим.*

Аддитивный от add — сложить. Основные цвета складываются, образуя результирующие цвета.

Модель RGB используется для излучаемого цвета, т. е. при подготовке экранных документов [9, 13].

Для модели RGB каждая из компонент может представляться числами, ограниченными некоторым диапазоном, — например дробными числами от 0 до 1 либо целыми числами от 0 до некоторого максимального значения (рис. 1.8). В настоящее время достаточно распространенным является формат True color, в котором каждая компонента представлена в виде байта, что дает 256 градаций для каждой компоненты: $R = 0 \ldots 255$, $G = 0 \ldots 255$, $B = 0 \ldots 255$. Количество цветов составляет $256 \cdot 256 \cdot 256 = 16.7$ млн (2^{24}).

Такой способ кодирования цветов можно назвать компонентным. В компьютере коды изображений True color представляются в виде троек байтов либо упаковываются в длинное целое (четырехбайтное) — 32 бита (так, например, сделано в API Windows):

$$C = 0000000\ bbbbbbb\ ggggggg\ rrrrrrr;$$

При ограничении количества цветов используют палитру, представляющую набор цветов, важных для данного изображения. Палитру можно воспринимать как таблицу цветов. Палитра устанавливает взаимосвязь между кодом цвета и его компонентами в выбранной цветовой модели.

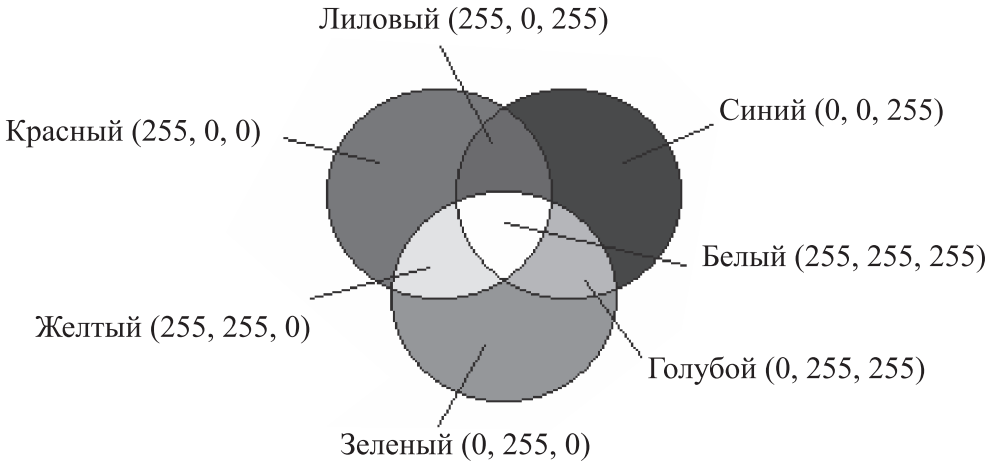


Рис. 1.8 – Цветовая модель RGB

Любой цвет можно представить в виде комбинации трех основных цветов: *красного* (Red), *зеленого* (Green) и *синего* (Blue). Эти цвета называются *цветовыми составляющими*. Чем больше значение байта цветовой составляющей, тем ярче этот цвет. При наложении одной составляющей на другую яркость суммарного цвета также увеличивается. Поэтому цветовая модель RGB, использующаяся для излучаемого цвета, называется *аддитивной*.

В качестве примера в таблице 1.1 приведена стандартная палитра дисплейных 16-цветных видеорежимов EGA, VGA.

Таблица 1.1

Код цвета	R	G	B	Название цвета
0	0	0	0	Черный
1	128	0	0	Темно-красный
2	0	128	0	Зеленый
3	0	0	128	Темно-синий
4	128	0	128	Темно-пурпурный
5	128	128	128	Серый 50%
6	255	0	0	Красный
7	0	255	0	Ярко-зеленый
8	255	255	0	Желтый
9	0	0	255	Синий
10	255	0	255	Пурпурный
11	0	255	255	Голубой
12	255	255	255	Белый

Недостатком такой палитры можно считать отсутствие одного из важных цветов — оранжевого. Существуют также иные стандартные палитры, например 256-цветная для VGA. Компьютерные видеосистемы обычно предоставляют возможность программисту установить собственную палитру.

Каждый цвет изображения, использующего палитру, кодируется индексом, который будет определять номер строки в таблице палитры. Поэтому такой способ кодирования цвета называют индексным.

1.6.2 Цветовая модель CMYK

Цветовая модель CMYK *(Cyan Magenta Yellow BlacK) — субтрактивная цветовая модель, согласно которой цвет кодируется четырьмя компонентами — голубым, лиловым, желтым и черным.*

Данная модель используется при работе с отраженным цветом, т. е. для подготовки печатных документов.

Субтрактивный от subtract — вычесть. Основные цвета вычитаются друг из друга, образуя результирующие цвета.

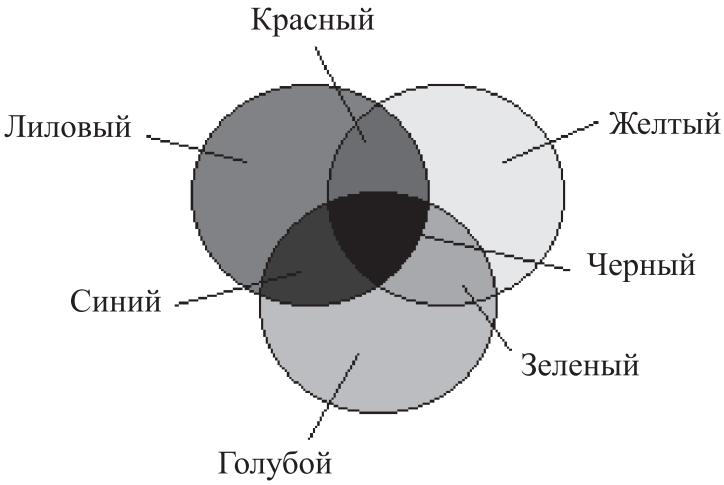


Рис. 1.9 – Цветовая модель CMYK

Эти цвета получаются в результате вычитания основных цветов модели RGB из белого цвета. Черный цвет задается отдельно (рис. 1.9). Увеличение количества краски приводит к уменьшению яркости цвета. Поэтому цветовая модель CMYK, использующаяся для отраженного цвета, называется *субтрактивной*.

1.6.3 Цветовая модель HSB

Цветовая модель HSB *(Hue Saturation Brightness) — цветовое пространство, основанное на трех характеристиках цвета: цветовом тоне, насыщенности и яркости.*

Системы цветов RGB и CMYK связаны с ограничениями, накладываемыми аппаратным обеспечением (монитор компьютера в случае RGB и типографские краски в случае CMYK). Модель HSB наиболее удобна для человека, т. к. она хорошо согласуется с моделью восприятия цвета человеком.

Компонентами модели HSB являются:

- *тон* — это конкретный оттенок цвета;
- *насыщенность* — характеризует его интенсивность;
- *яркость цвета* — зависит от примеси черной краски, добавленной к данному цвету.

Значение цвета выбирается как вектор, выходящий из центра окружности. Точка в центре соответствует белому цвету, а точки по границе окружности — чистым цветам. Направление вектора определяет цветовой оттенок и задается в угловых градусах. Длина вектора определяет насыщенность цвета (рис. 1.10).

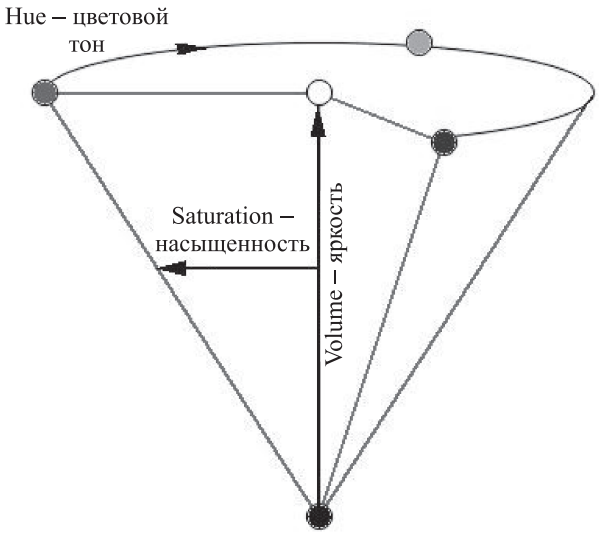


Рис. 1.10 – Цветовая модель HSB

Модель HSB удобно применять при создании собственно изображения, а по окончании работы изображение можно преобразовать в модель RGB или CMYK.

Контрольные вопросы по главе 1

1) К какому направлению работы с изображением относится передача изображения с устранением шумов и сжатием данных?
2) В каких единицах измеряют разрешение изображения оригинала?
3) Как называют наименьший элемент растровой графики?
4) Какой вид изображения масштабируется без потери качества?
5) Какая цветовая модель называется субтрактивной?

Глава 2

МАТЕМАТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ ГРАФИКИ

2.1 Геометрическое моделирование

Реалистичность восприятия человеком сложного компьютерного изображения определяется умением разработчика математической модели изображаемого объекта или процесса достоверно повторить на экране его развитие в пространстве и во времени. Модель включает в себя систему уравнений и алгоритмов их реализации. Математической основой построения модели являются уравнения, описывающие форму и движение объектов. Все многообразие геометрических объектов является комбинацией различных примитивов — простейших фигур, которые, в свою очередь, состоят из графических элементов — точек, линий и поверхностей. Как было рассмотрено ранее, на примере двухмерных приложений, все разнообразие геометрических объектов можно свести к ограниченному множеству простейших сущностей. Рассмотрим три базовых типа — скаляр, точку и вектор [6, 10].

Геометрический примитив *— элементарный фрагмент изображения, при помощи которого описывается объемный объект.*

Скаляр *(scalaris — ступенчатый) — величина, каждое значение которой может быть выражено одним (действительным) числом. Примерами скаляра являются длина, площадь, время, масса, плотность и т. д.*

Вектор *— это направленный отрезок прямой линии, характеризуемый только его длиной и направлением.*

В геометрическом смысле вектор — направленный отрезок, то есть отрезок, у которого указаны начало и конец.

Существует три варианта определения, в зависимости от того, с какой точки зрения их рассматривать:

- с математической (формально);
- геометрической;
- с точки зрения программной реализации.

2.1.1 Геометрическое определение базовых типов

Роль фундаментального геометрического объекта в КГ играет точка. В КГ точки часто связывают с направленными отрезками.

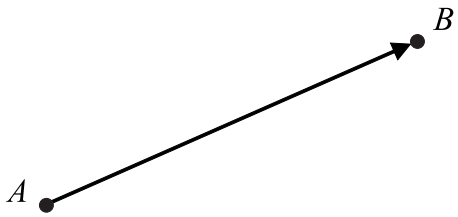
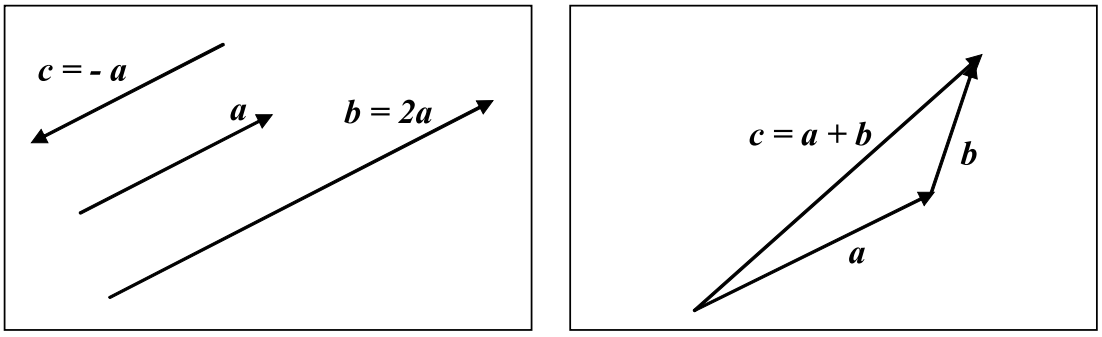


Рис. 2.1 – Соединение двух точек вектором

В общем смысле физики используют термин «вектор» для обозначения любой величины, характеризуемой направлением и значением (рис. 2.1).



а) *б)*

Рис. 2.2 – Соотношение между векторами

Физические величины, такие, как скорость и сила, являются векторами. Однако вектор в этом смысле не имеет фиксированной точки приложения (позиции). Длина и направление вектора характеризуются вещественными числами. Вектор ***a*** на рис. 2.2 имеет то же направление, что и вектор ***b***, но ***b*** имеет длину в два раза

большую, поэтому можно записать $\boldsymbol{b} = 2\boldsymbol{a}$. Вектор $\boldsymbol{c}$ имеет ту же длину, что и $\boldsymbol{a}$, но противоположное направление, а поэтому соотношение между ними выразится формулой: $\boldsymbol{c} = -\boldsymbol{a}$.

Объединять векторы можно с помощью правила сочленения начала с концом, схематически представленного на рис. 2.2, *б*. Новый вектор $\boldsymbol{c}$ называют суммой векторов $\boldsymbol{a}$ и $\boldsymbol{b}$, и он равен: $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$.

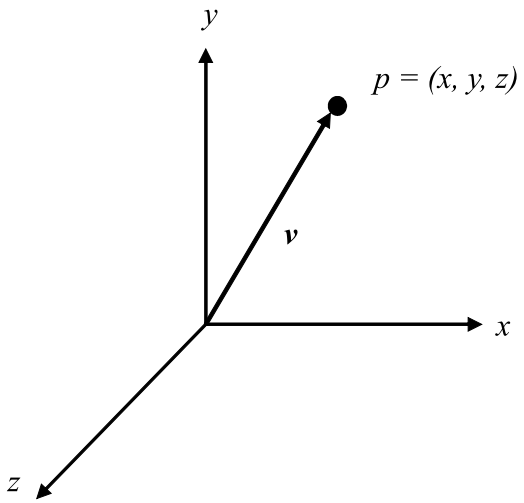
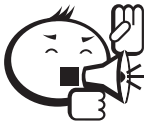


Рис. 2.3 – Сомнительное представление вектора

Точки и векторы — это разные геометрические типы. Геометрическое представление точки в виде направленного отрезка, соединяющего некоторую опорную точку с заданной (рис. 2.3), следует рассматривать как сомнительное.

2.1.2 Математическое определение базовых типов

Математическое обеспечение систем компьютерной графики — это совокупность методов представления и манипулирования множеством геометрических элементов, главными из которых являются точки и отрезки прямых. Эти методы базируются на свойствах нескольких типов абстрактных пространств [15].



Векторное (линейное) пространство содержит объекты только двух типов: скаляры (действительные числа) и векторы.

Множество вещественных чисел и операций над ними представляет собой скалярное поле. Рассмотрим множество S скалярных элементов, обозначаемых α, β, γ. Над скалярами определены две основные операции — сложение и умножение.

Если $\forall\alpha, \beta \in S$, то $\alpha + \beta \in S$, $\alpha \cdot \beta \in S$. Эти операции обладают свойствами ассоциативности, коммутативности и дистрибутивности, т. е. если $\forall\alpha, \beta, \gamma \in S$, то

$$\alpha + \beta = \beta + \alpha,$$
$$\alpha \cdot \beta = \beta \cdot \alpha,$$
$$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma,$$
$$\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma,$$
$$\alpha \cdot (\beta + \gamma) = (\alpha \cdot \beta) + (\alpha \cdot \gamma).$$

Существуют два специальных скаляра (универсальные тождественные элементы) — числа, таких, что прибавление первого и умножение на второе не меняют ни одного числа, т. е. для $\forall \alpha \in S$:

$$\alpha + 0 = 0 + \alpha = \alpha,$$
$$\alpha \cdot 1 = 1 \cdot \alpha = \alpha.$$

Каждый элемент скалярного множества имеет соответствующие ему аддитивный и мультипликативный инверсные элементы:

$$\alpha + (-\alpha) = 0,$$
$$\alpha \cdot \alpha^{-1} = 1,$$
$$-\alpha,\ \alpha^{-1} \in S.$$

Векторное пространство помимо скаляров содержит вектора. Над векторами определены две операции: сложение вектора с вектором и умножение вектора на скаляр, т. о. можно создавать новые векторы с помощью умножения скаляра на вектор или с помощью операции сложения векторов.

Аффинное пространство *— это расширение векторного пространства, в которое включен дополнительный тип объектов — точка.*

Здесь определена операция сложения точки и вектора, результатом которой является точка. Обратной ей является операция вычитания двух точек, результатом которой будет вектор.

2.2 Координатный метод

Координатный метод был введен в XVII веке французскими математиками Р. Декартом и П. Ферма [11]. На этом зиждется аналитическая геометрия, которую можно считать фундаментом КГ. В современной КГ широко используется координатный метод. Этому есть несколько причин:

- каждая точка на экране (или на бумаге при печати на принтере) задается координатами — например, пиксельными;
- координаты используются для описания объектов, которые будут отображаться как пространственные;

- при выполнении многих промежуточных действий отображения используют разные системы координат и преобразования из одной системы в другую.

Положение точки в R^n задается радиус-вектором $p = \begin{bmatrix} p_1 & p_2 & \dots & p_n \end{bmatrix}$, имеющим n координат и разложение:

$$p = \begin{bmatrix} p_1 & p_2 & \dots & p_n \end{bmatrix} = \sum_{i=1}^{n} p_i \cdot e_i.$$

По n линейно-независимым базисным векторам $e_1 \dots e_n$. Совокупность базисных векторов и единиц измерения расстояний вдоль этих векторов составляет систему координат.

2.2.1 Системы координат

Наиболее употребительными в компьютерной графике являются декартовы системы координат как способ удобного создания абстракций реальных предметов окружающего мира [10]. Цилиндрические, сферические, проективные и различные другие координаты в пространстве обычно используются для моделирования специальных эффектов при деформации объектов.

При отображении пространственных объектов используют понятие мировых координат — трёхмерных декартовых координат пространства, в котором размещаются объекты.

Мировая система координат *(МСК) — xyz — содержит точку отсчета (начало координат) и линейно независимый базис, благодаря которым становится возможным цифровое описание геометрических свойств любого графического объекта в абсолютных единицах.*

Экранная система координат *(ЭСК) — $x_э y_э z_э$ — система координат, в которой задается положение проекций геометрических объектов на экране дисплея.*

ЭСК связана с тем графическим устройством, где в заданной проекции отображается создаваемая трёхмерная сцена. Так как практически все устройства графического вывода являются плоскостными, то две из трёх экранных координат (X и Y) располагаются в плоскости экрана соответственно по горизонтали и вертикали, а координата Z направлена перпендикулярно им (если «вглубь» экрана, то получившаяся система координат *левосторонняя*).

Проекция точки в ЭСК имеет координату $z_э = 0$. Тем не менее не следует отбрасывать эту координату, поскольку МСК и ЭСК часто выбираются совпадающими, а вектор проекции $[x_э, y_э, 0]$ может участвовать в преобразованиях, где нужны не две, а три координаты.

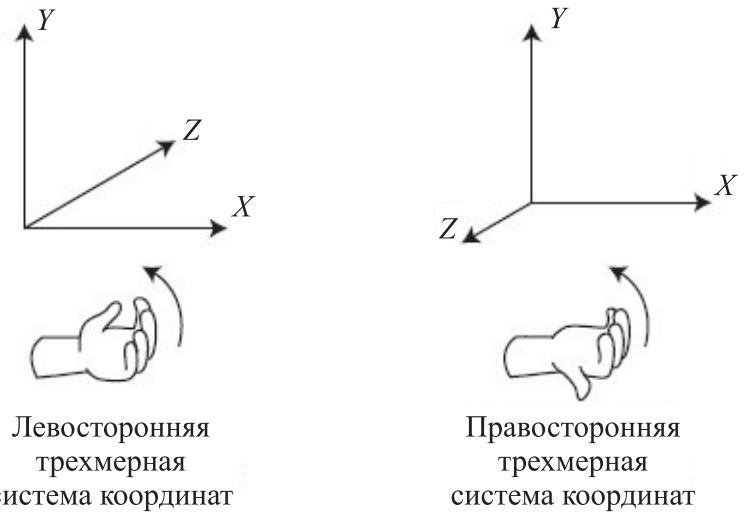


Рис. 2.4 – Левосторонняя и правосторонняя трехмерные системы координат

Система координат сцены *(СКС) — $x_c y_c z_c$ — описывает положение всех объектов сцены — некоторой части мирового пространства с собственным началом отсчета и базисом, которые используются для описания положения объектов независимо от МСК.*

Каждый из объектов обычно имеет собственную объектную систему координат.

Объектная система координат *(ОСК) — $x_o y_o z_o$ — связана с конкретным объектом и совершает с ним все движения в СКС или МСК.*

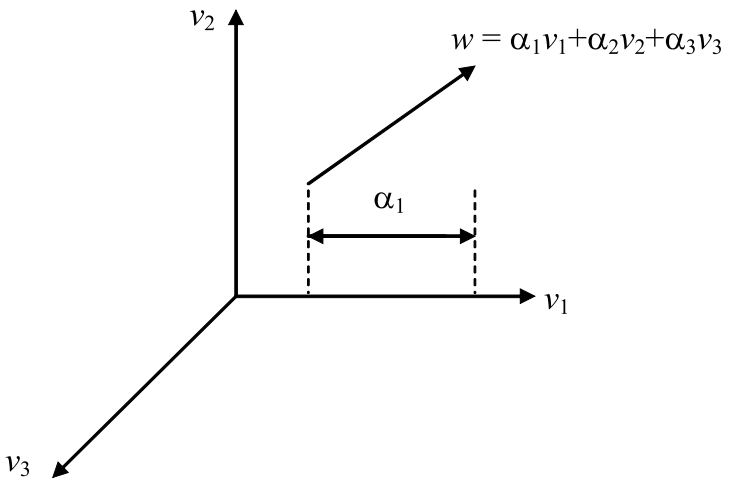


Рис. 2.5 – Представление вектора

В трехмерном векторном пространстве можно однозначно представить любой вектор w в виде линейно-независимых векторов v_1, v_2, v_3:

$$w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3,$$

где α_1, α_2, α_3 — это компоненты или координаты вектора w в базисе v_1, v_2, v_3 (рис. 2.4). В векторном пространстве векторы как геометрические объекты не имеют точки приложения, а характеризуются только направлением и модулем.

Аффинное пространство включает точки, в нем фиксируется точка отсчета — начало координат. Такое представление требует использования точки отсчета и векторов базиса, которые в совокупности называются фреймом (окном). В конкретном фрейме любой вектор можно записать:

$$w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3,$$

как и в векторном пространстве. Для каждой точки можно записать соотношение:

$$P = P_0 + \eta_1 v_1 + \eta_2 v_2 + \eta_3 v_3.$$

Итак, любой вектор описывается в фрейме тремя скалярами, а описание точки включает три скаляра и данные о точке отсчета.

2.2.2 Преобразования координат

Преобразования координат в явном и неявном виде широко используются в компьютерной графике с целью описания графического объекта в наиболее удобных координатных системах, для изменения масштаба элементов чертежа, построения проекций пространственных образов, направленной деформации объектов [16, 17].

В двумерном пространстве (R^2) наиболее распространены *декартова СК* (x, y) и *полярная СК* (r, φ) (r — радиус-вектор точки, φ — угол поворота).

$$\begin{cases} x = r\cos\varphi \\ y = r\sin\varphi \end{cases} \qquad \begin{cases} r = \sqrt{x^2 + y^2} \\ \operatorname{tg}\varphi = \dfrac{y}{x} \end{cases}.$$

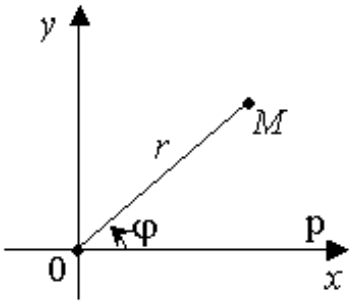


Рис. 2.6 – Полярная система координат

В трехмерном пространстве (R^3):

- ортогональная декартова СК (x, y, z);
- цилиндрическая СК (ρ, y, φ).

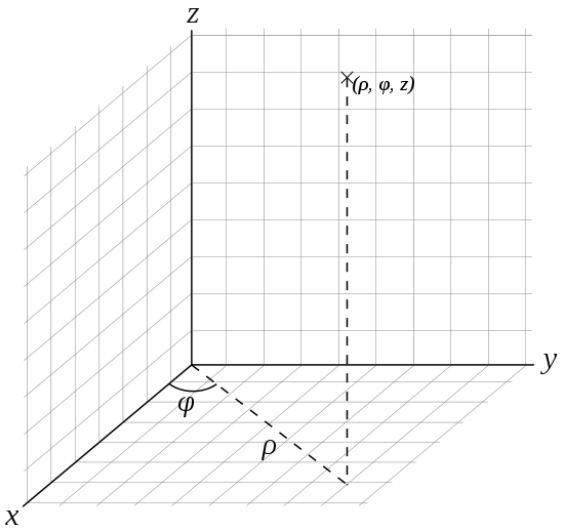


Рис. 2.7 – Цилиндрическая система координат

Закон преобразования координат от цилиндрических к декартовым (рис. 2.7):

$$\begin{cases} x = \rho \cos \varphi, \\ y = \rho \sin \varphi, \\ z = z. \end{cases}$$

Закон преобразования координат от декартовых к цилиндрическим:

$$\begin{cases} \rho = \sqrt{x^2 + y^2}, \\ \varphi = \operatorname{arctg}\left(\dfrac{y}{x}\right), \\ z = z. \end{cases}$$

- сферическая СК (r, φ, ω).

Если заданы сферические координаты точки, то переход к декартовым осуществляется по формулам (рис. 2.8):

$$\begin{cases} x = r \sin \theta \cos \varphi, \\ y = r \sin \theta \sin \varphi, \\ z = r \cos \theta. \end{cases}$$

Обратно, от декартовых к сферическим:

$$\begin{cases} r = \sqrt{x^2 + y^2 + z^2}, \\ \theta = \arccos\left(\dfrac{z}{\sqrt{x^2 + y^2 + z^2}}\right) = \operatorname{arctg}\left(\dfrac{\sqrt{x^2 + y^2}}{z}\right), \\ y = \operatorname{arctg}\left(\dfrac{y}{x}\right). \end{cases}$$

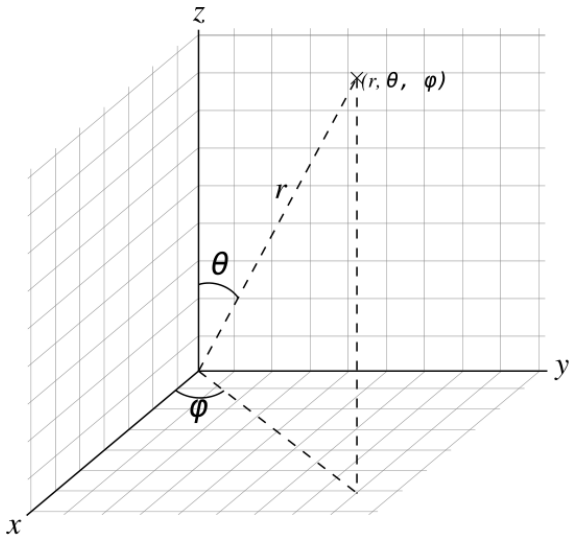


Рис. 2.8 – Сферическая система координат

Пусть задана n-мерная система координат в базисе $(k_1, k_2, \ldots, k_n)$, описывающая положение точки в пространстве с помощью числовых значений k_i. Если задать другую, N-мерную систему координат в базисе $(m_1, m_2, \ldots, m_n)$ и поставить задачу определения координат в новой системе, зная координаты в старой, то решение (если оно существует) можно записать в таком виде:

$$\begin{cases} m_1 = f_1(k_1, k_2, \ldots, k_n), \\ m_2 = f_2(k_1, k_2, \ldots, k_n), \\ \ldots\ldots\ldots \\ m_N = f_N(k_1, k_2, \ldots, k_n), \end{cases}$$

где f_i — функции пересчёта i-й координаты.

По виду функций преобразования различают линейные и нелинейные преобразования. Если при всех $i = 1, 2, \ldots, N$ функции f_i — линейные относительно аргументов $(k_1, k_2, \ldots, k_n)$, то есть:

$$f_i = a_{i1}k_1 + a_{i2}k_2 + \ldots + a_{in}k_n + a_{i,n+1},$$

где a_{ij} — константы, то такие преобразования называют линейными, а при $n = N$ — *аффинными*.

Аффинная система координат — *система в n-мерном аффинном пространстве, определяемая совокупностью n линейно независимых векторов, исходящих из начала координат.*

Координаты точки в этой системе — это компоненты разложения радиус-вектора точки по координатным векторам. Число независимых координат, которые определяют ее положение, задает размерность пространства [19].

В компьютерной графике наиболее распространены двумерные и трехмерные аффинные преобразования. В таких преобразованиях особую роль играют несколько важных частных случаев: поворот (вокруг начальной точки), растяжение (сжатие), отражение и сдвиг (перенос). Любое аффинное преобразование может быть представлено в виде последовательности этих простейших геометрических операций.

Классические геометрические преобразования (сдвиги, повороты) описываются с помощью математических моделей, основанных на использовании матриц размера 3×3. При классическом подходе каждое из преобразований представляется отдельной матрицей. Для компьютерной графики это часто приводит к неоправданному увеличению объема вычислений и, следовательно, к снижению скорости вывода графической информации. Поэтому используется математический аппарат, обеспечивающий более компактное описание геометрических преобразований.

2.3 Аффинные преобразования

При построении изображений часто приходится иметь дело с ситуациями, когда общее изображение включает в себя целый ряд компонент (фрагментов), отличающихся друг от друга только местоположением, ориентацией, масштабом.

В этом случае целесообразно описать один фрагмент в качестве базового, а затем получать остальные требуемые фрагменты путем использования операций преобразования.

С помощью операций преобразования можно выполнять следующие действия:

1) перемещать рисунки из одного места экрана в другое;
2) создавать рисунок из более мелких элементов;
3) добавлять к существующему рисунку новые элементы;
4) увеличивать размер рисунка для улучшения его наглядности или отображения более мелких деталей;
5) уменьшать размер рисунка для внесения, например, поясняющих надписей или отображения на экране новых рисунков;
6) создавать движущиеся изображения.

Все изменения рисунков можно выполнить с помощью трех базовых операций:

- смещения (переноса, перемещения) изображения;
- масштабирования (увеличения или уменьшения размеров) изображения;
- поворота изображения (употребляют также термины «вращение», «изменение ориентации»).

Эти операции называются аффинными преобразованиями. Различают двумерные и трехмерные аффинные преобразования.

2.3.1 Двумерные аффинные преобразования

Предположим, что на плоскости введена декартова система координат (OXY). Тогда каждой точке M ставится в соответствие упорядоченная пара чисел (x, y) ее координат. Вводя на плоскость еще одну декартову систему координат $(O'X'Y')$,

ставим в соответствие той же точке M другую пару чисел (x', y'). Переход от одной декартовой системы координат на плоскости к другой описывается следующими соотношениями:

$$\begin{cases} x' = \alpha x + \beta y + \lambda, \\ y' = \gamma x + \delta y + \mu, \end{cases} \tag{2.1}$$

где α, β, γ, λ — произвольные числа, но:

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

Формулу (2.1) можно рассматривать как один из следующих случаев:

1) Сохраняется точка, и изменяется координатная система (рис. 2.9, *а*) — произвольная точка M остается той же, изменяются лишь ее координаты: $(x, y) \to (x', y')$.
2) Изменяется точка, и сохраняется координатная система (рис. 2.9, *б*) — формула (2.1) задает отображение, переводящее произвольную точку $M(x, y)$ в точку $M(x', y')$, координаты которой определены в той же координатной системе.

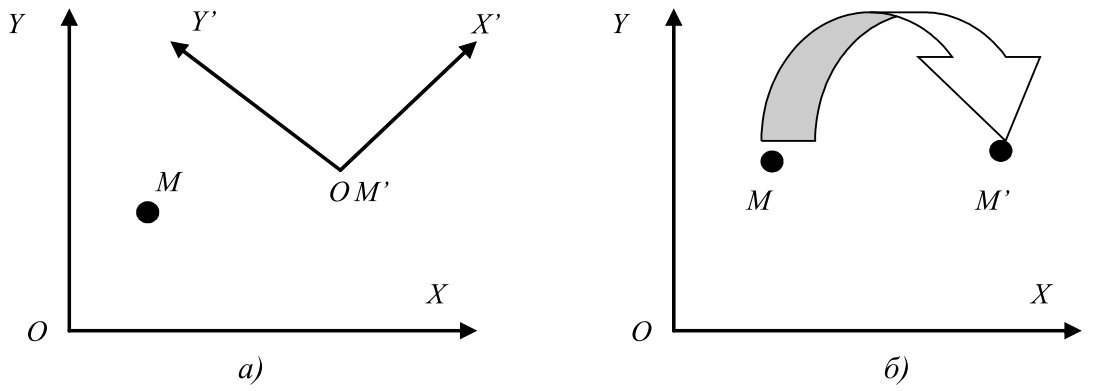


Рис. 2.9 – Аффинные преобразования на плоскости

В дальнейшем будем рассматривать формулу (2.1) как второй случай преобразования, соответствующий рис. 2.9, *б*.

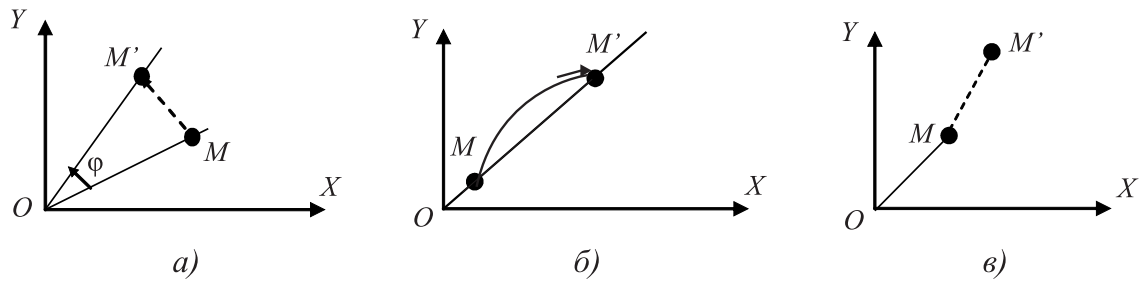


Рис. 2.10 – Двумерные преобразования: *а* — поворота, *б* — растяжения (сжатия), *в* — сдвига

Рассмотрим несколько частных случаев преобразования координат.

I. Поворот (вокруг начальной точки на угол φ) описывается формулами (рис. 2.10, *а*):

$$x' = x\cos(\varphi) - y\sin(\varphi),$$
$$y' = x\sin(\varphi) + y\cos(\varphi).$$

II. Растяжение (сжатие) вдоль координатных осей:

$$x' = \alpha x,$$
$$y' = \delta y,$$
$$\alpha > 0, \quad \delta > 0.$$

Растяжение (сжатие) вдоль оси абсцисс обеспечивается при условии, что $\alpha > 1$ $(\alpha < 1)$. На рис. 2.10, *б* $\alpha = \delta > 1$.

III. Отражение относительно оси абсцисс задается при помощи формул:

$$x' = x, \quad y' = -y.$$

IV. Перенос обеспечивают соотношения:

$$x' = x + \lambda,$$
$$y' = y + \mu.$$

На рис. 2.10, *в* вектор переноса $\overrightarrow{MM'}$ имеет координаты λ и μ.

Выбор этих частных случаев определяется двумя обстоятельствами:

- каждое из приведенных выше преобразований имеет простой и наглядный геометрический смысл;
- как доказывается в курсе аналитической геометрии, любое преобразование вида (2.1) всегда можно представить как последовательное исполнение (суперпозицию) простейших преобразований вида I, II, III, IV.

Для эффективного использования этих известных формул в задачах КГ более удобной является их матричная запись. Матрицы, соответствующие случаям I, II, III, строятся легко. Они имеют следующий вид:

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

В данном случае аффинное преобразование переноса (IV) в матричном виде размерностью 2×2 записать невозможно. Для простоты алгоритмизации любого преобразования необходимы все четыре простейших преобразования I, II, III, IV. Для этого введем понятия однородных координат.

Чтобы перейти к единой (матричной) форме записи всех четырех преобразований I, II, III, IV, нужно перейти к однородным координатам произвольной точки [19].

Однородными координатами *точки* $M(x,y)$ *называется тройка одновременно не равных нулю чисел* x_1, x_2, x_3, *если:*

$$x = \frac{x_1}{x_3}, \quad y = \frac{x_2}{x_3}.$$

В КГ однородные координаты вводятся так: точке $M(x,y)$ ставится в соответствие точка $M'(x,y,1)$ в пространстве рис. 2.11. Заметим, что произвольная точка на прямой, соединяющей начало координат, точку $O(0,0,0)$, с точкой $M'(x,y,1)$, может быть задана тройкой (hx,hy,h). Исключая точку O из рассмотрения, будем считать, что $h \neq 0$. Вектор, определяемый тройкой hx, hy, h, является направляющим вектором прямой, соединяющей точки O и M'. Эта прямая пересекает плоскость $z = 1$ в точке $(x,y,1)$, которая однозначно определяет точку (x,y) координатной плоскости xOy. Таким образом, между точкой (x,y) и множеством троек (hx,hy,h), $h \neq 0$ устанавливается взаимно однозначное соответствие. Это позволит считать hx, hy, h однородными координатами точки M.

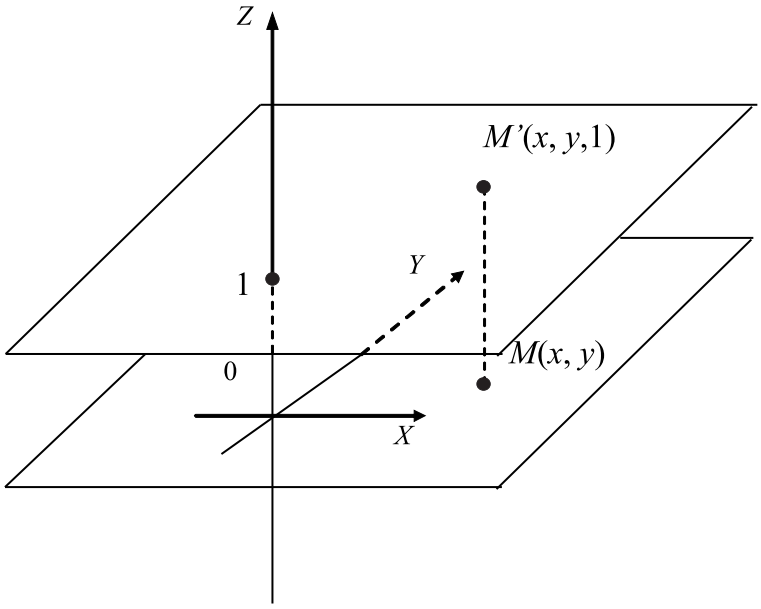


Рис. 2.11 – Однородные координаты

В проективной геометрии однородные координаты обозначают $x_1 : x_2 : x_3$. Основная цель введения однородных координат — удобство их использования. Например, если у точки координаты $(10000, 5000, 3000)$, то при преобразовании ее координат может произойти переполнение арифметического устройства. В этом случае можно взять $h = 0{,}001$, при этом однородные координаты будут $(10, 5, 3)$. Данный пример показывает, насколько полезны однородные координаты при проведении расчетов. Однако основной целью введения однородных координат в КГ является их удобство.

Считая $h = 1$, выражение (2.1) можно переписать в виде:

$$(x', y', 1) = (x, y, 1) \cdot \begin{bmatrix} \alpha & \gamma & 0 \\ \beta & \delta & 0 \\ \lambda & \mu & 1 \end{bmatrix}, \quad \text{или} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & \gamma & 0 \\ \beta & \delta & 0 \\ \lambda & \mu & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Чтобы реализовать то или иное отображение по заданным геометрическим характеристикам, надо найти элементы соответствующей матрицы. Обычно ее построение разбивают на этапы. На каждом этапе строится матрица для одного из выделенных выше случаев I, II, III, IV.

Все многообразие различных преобразований теперь можно реализовать поэтапным применением следующих преобразований:

I. Матрица вращения (rotation):

$$[R] = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

II. Матрица растяжения-сжатия (dilatation):

$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

здесь $\alpha > 0$ — коэффициент растяжения (сжатия) вдоль оси абсцисс, $\delta > 0$ — коэффициент растяжения (сжатия) вдоль оси ординат.

III. Матрица отражения (mirror reflection):

$$[M] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

IV. Матрица переноса (translation):

$$[T] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \lambda & \mu & 1 \end{bmatrix}.$$

Пример

Построить матрицу растяжения с коэффициентом растяжения α вдоль оси абсцисс и δ вдоль оси ординат и с центром в точке $A(a, b)$.

Решение:

Шаг 1. Перенос на вектор $-A(-a, -b)$ для смещения центра растяжения с началом координат:

$$[T_{-A}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{bmatrix}.$$

Шаг 2. Растяжение вдоль координатных осей с коэффициентами α и δ. Матрица преобразования имеет вид:

$$[D] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

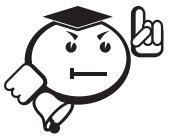
Шаг 3. Перенос на вектор $A(a,b)$ для возвращения центра растяжения в прежнее положение:

$$[T_A] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}.$$

Перемножив матрицы в том же порядке $[T_{-A}] \cdot [D] \cdot [T_A]$, получим вид нашего преобразования [19]:

$$(x', y', 1) = (x, y, 1) \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \delta & 0 \\ a \cdot (1-\alpha) & b \cdot (1-\delta) & 1 \end{bmatrix}.$$

. . . . . .



. . . . . .

Для выполнения преобразований необходимо знать основные положения матричной алгебры.

. . . . . .

Матрица — это прямоугольный массив чисел размером $m \times n$ (m — число строк, n — число столбцов). Если $m = n$, то матрица называется квадратной. Размер матрицы называется ее порядком. Элементы $a_{11}, \ldots, a_{nm}$ называются главными диагональными элементами.

Нулевая матрица — матрица, все элементы которой равны 0.

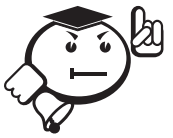
Единичная матрица — элементы главной диагонали равны 1, а остальные 0.

Если две матрицы A и B имеют одинаковый порядок $m \times n$, то допустимо их сложение и вычитание.

$$C = A + B, \quad \text{или} \quad c_{ij} = a_{ij} + b_{ij},\ i = 1 \ldots n,\ j = 1 \ldots m.$$

Самая распространенная операция в КГ — умножение матриц. Пусть матрица A имеет размерность $k \times m$, матрица B — $m \times n$. Результирующая матрица будет иметь порядок $k \times n$.

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



Умножение матриц не коммутативно, т. е. $A \cdot B \neq B \cdot A$, однако выполняется:

- дистрибутивность: $A \cdot (B + C) = A \cdot B + A \cdot C$, $(A + B) \cdot C = A \cdot C + B \cdot C$;
- ассоциативность: $A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C$.

2.3.2 Аффинные преобразования в пространстве

Как и в двумерном случае перейдем к однородным координатам, заменим тройку (x, y, z), задающую точку в пространстве, на $(x, y, z, 1)$. В общем виде замена производится на (hx, hy, hz, h), где $h \neq 0$.

Каждая точка пространства, кроме начала координат, может быть задана четверкой одновременно не равных нулю чисел. Четверка чисел (hx, hy, hz, h), где $h \neq 0$, определена однозначно с точностью до общего множителя.

Как известно, любое аффинное преобразование в трехмерном пространстве может быть представлено в виде суперпозиции вращений, растяжений, отражений и переносов.

I. Матрица вращения в пространстве.

Матрица вращения вокруг оси абсцисс на угол φ:

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица вращения вокруг оси ординат на угол ψ:

$$[R_Y] = \begin{bmatrix} \cos\psi & 0 & -\sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица вращения вокруг оси аппликат на угол θ:

$$[R_Z] = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

II. Матрица растяжения (сжатия):

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

здесь $\alpha > 0$ — коэффициент растяжения (сжатия) вдоль оси абсцисс, $\beta > 0$ — коэффициент растяжения (сжатия) вдоль оси ординат, $\gamma > 0$ — коэффициент растяжения (сжатия) вдоль оси аппликат.

III. Матрица отражения.

Матрица отражения относительно плоскости *XOY*:

$$[M_Z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица отражения относительно плоскости *YOZ*:

$$[M_X] = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матрица отражения относительно плоскости *ZOX*:

$$[M_Y] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

IV. Матрица переноса:

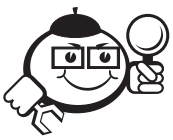
$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & \mu & \nu & 1 \end{bmatrix},$$

здесь (λ, μ, ν) — вектор переноса.

При помощи матриц I, II, III, IV можно подвергать простейшим преобразованиям любые плоские и пространственные фигуры.

Как и в двумерном случае, все выписанные матрицы невырождены.

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



Пример

Построить матрицу вращения на угол φ вокруг прямой L, проходящей через точку $A(a,b,c)$ и имеющей направляющий вектор (k,m,n).

Решение:

Можно считать, что направляющий вектор прямой является единичным [8]:

$$k^2 + m^2 + n^2 = 1.$$

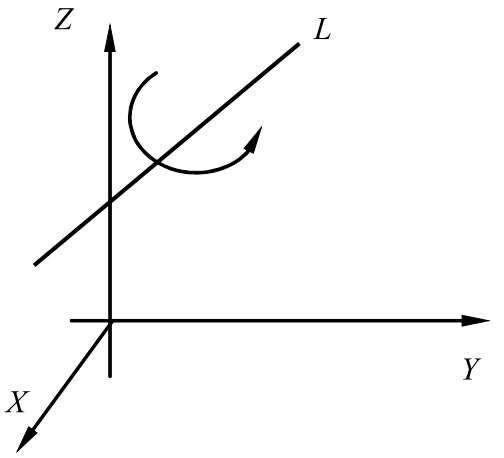


Рис. 2.12 – Пример трехмерного преобразования

На рис. 2.12 схематично показано, матрицу какого преобразования требуется найти.

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

Шаг 1. Перенос на вектор $-A(-a,-b,-c)$ при помощи матрицы

$$[T]=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ -a & -b & -c & 1\end{bmatrix}.$$

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

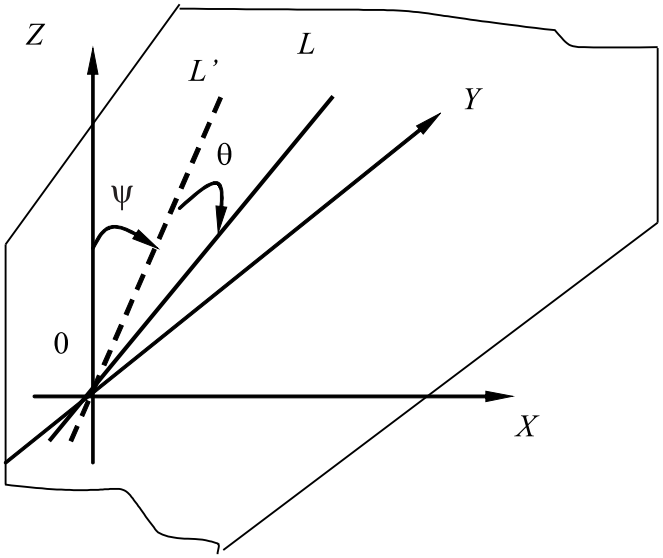


Рис. 2.13 – Совмещение оси Z с прямой L двумя поворотами вокруг оси X и оси Y

Шаг 2. Совмещение оси аппликат с прямой L двумя поворотами вокруг оси абсцисс и оси ординат. 1-й поворот — вокруг оси абсцисс на угол ψ (подлежащий определению). Чтобы найти этот угол, рассмотрим ортогональную проекцию L' исходной прямой L на плоскость $X = 0$ (рис. 2.13).

Направляющий вектор прямой L' определяется просто — он равен $(0, m, n)$. Отсюда сразу же вытекает, что

$$\cos\psi = \frac{n}{d}; \quad \sin\psi = \frac{m}{d},$$

где $d = \sqrt{m^2 + n^2}$.

Соответствующая матрица вращения имеет следующий вид:

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{n}{d} & \frac{m}{d} & 0 \\ 0 & -\frac{m}{d} & \frac{n}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

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

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

Второй поворот вокруг оси ординат на угол θ, определяемый соотношениями:

$$\cos\theta = k, \quad \sin\theta = -d.$$

Соответствующая матрица вращения записывается в следующем виде:

$$[R_Y] = \begin{bmatrix} k & 0 & d & 0 \\ 0 & 1 & 0 & 0 \\ -d & 0 & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Шаг 3. Вращение вокруг прямой L на заданный угол φ.

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

$$[R_Z] = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Шаг 4. Поворот вокруг оси ординат на угол $-\theta$ и затем поворот вокруг оси абсцисс на угол $-\psi$.

Шаг 5. Перенос на вектор $A(a, b, c)$.

Перемножив найденные матрицы в порядке их построения, получим следующую матрицу [19]:

$$[T][R_x][R_y][R_z][R_y]^{-1}[R_x]^{-1}[T]^{-1}.$$

. . . . . .

Контрольные вопросы по главе 2

1) Какой базовый тип компьютерной графики определен как величина, каждое значение которой может быть выражено одним числом?
2) Что будет являться результатом сложения точки и вектора в аффинном пространстве?
3) Какая система координат содержит точку отсчета (начало координат) и линейно независимый базис, благодаря которым становится возможным цифровое описание геометрических свойств любого графического объекта в абсолютных единицах?
4) Какое двумерное аффинное преобразование невозможно записать в виде матрицы 2×2 и поэтому все четыре преобразования представляют в виде матриц 3×3?
5) Какое аффинное преобразование реализует матрица $[A] = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$?

Глава 3

БАЗОВЫЕ ВЫЧИСЛИТЕЛЬНЫЕ И РАСТРОВЫЕ АЛГОРИТМЫ

3.1 Область визуализации и функция кадрирования

Для построения изображения на экране дисплея определяется область, в которой это изображение формируется. Окно — это прямоугольник, в пределах которого вычерчивается объект или его часть, размеры и положение окна определяются в системе мировых координат.

Введем еще одно понятие — рабочая область визуализации, или поле вывода, т. е. та часть экрана, на которую осуществляется вывод (рис. 3.1). Она задается аналогично окну, т. е. указываются максимальные и минимальные значения по координатным осям в единицах измерения на экране. Теперь окно надо отобразить на экранную область. Вначале вычисляются коэффициенты масштабирования по осям [1, 2]:

$$f_x = \frac{X_{\max} - X_{\min}}{x_{\max} - x_{\min}}, \quad f_y = \frac{Y_{\max} - Y_{\min}}{y_{\max} - y_{\min}},$$

Затем вычисляем расстояние $X - X_{min}$ точки изображения от левого края экранной области:

$$\begin{cases} X = X_{\min} + f_x(x - x_{\min}) \\ Y = Y_{\min} + f_y(y - y_{\min}) \end{cases}$$

Пусть рабочая область визуализации ограничена прямыми $X = n_1$, $Y = m_1$, $X = n_2$, $Y = m_2$. X, Y — координаты на экране. Где m_1, m_2, n_1, n_2, — координаты растра, и пусть $W = R \times R$ — произвольное прямоугольное окно в мировой системе координат. Тогда функция F: $W \to [m_1 : m_2] \times [n_1 : n_2]$ будет называться функцией кадрирования.

Обычно кадрирование заключается в отображении прямоугольной области (мирового пространства) в прямоугольную область на экране.

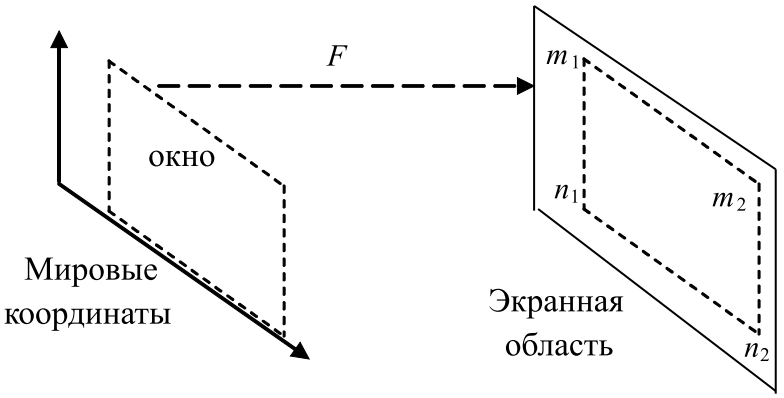


Рис. 3.1 – Рабочая область визуализации

Ограничения прямоугольными областями объясняются тем, что при таких отображениях необходимы только геометрические преобразования переноса, вращения, масштабирования и дополнительная операция, называемая отсечением.

3.2 Отсечение

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

Отсечение *(clipping) — процесс отбрасывания частей изображения, выходящих за границы окна.*

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

Программное исполнение отсечения достаточно медленный процесс, поэтому, естественно, в мощные дисплеи встраивается соответствующая аппаратура. Первое сообщение об аппаратуре отсечения, использующей алгоритм отсечения делением отрезка пополам и реализованной в устройстве Clipping Divider, появилось в 1968 г. Этот алгоритм был рассмотрен при изучении технических средств. Здесь мы рассмотрим программные реализации алгоритма отсечения.

Отсекаемые отрезки могут быть трех классов:

- целиком видимые;
- целиком невидимые;
- пересекающие окно.

Очевидно, что целесообразно без выполнения большого объема вычислений принять решение о видимости целиком или отбрасывании. По способу выбора простого решения об отбрасывании невидимого отрезка целиком или принятия его существует два основных типа алгоритмов отсечения — алгоритмы, использующие кодирование концов отрезка или всего отрезка, и алгоритмы, использующие параметрическое представление отсекаемых отрезков и окна отсечения. Представителем первого типа алгоритмов является алгоритм Коэна—Сазерленда (Cohen—Sutherland, CS-алгоритм). Представителем алгоритмов второго типа — алгоритм Лианга—Барского (Liang—Barsky, LB-алгоритм).

Алгоритмы с кодированием применимы для прямоугольного окна, стороны которого параллельны осям координат, в то время как алгоритмы с параметрическим представлением применимы для произвольного окна [3].

3.2.1 Двумерный алгоритм Коэна—Сазерленда

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

1) большинство примитивов содержится целиком в большом окне;
2) большинство примитивов лежит целиком вне относительно маленького окна.

Идея алгоритма состоит в следующем:

Окно отсечения и прилегающие к нему части плоскости вместе образуют 9 областей (рис. 3.2). Каждой из областей присвоен 4-разрядный код.

Две конечные точки отрезка получают 4-разрядные коды, соответствующие областям, в которые они попали. Смысл разрядов кода:

1 рр = 1 — точка над верхним краем окна;

2 рр = 1 — точка под нижним краем окна;

3 рр = 1 — точка справа от правого края окна;

4 рр = 1 — точка слева от левого края окна.

Определение того, лежит ли отрезок целиком внутри окна или целиком вне окна, выполняется следующим образом:

- если коды обоих концов отрезка равны 0 то отрезок целиком внутри окна, отсечение не нужно, отрезок принимается как тривиально видимый (отрезок *AB* на рис. 3.2);
- если логическое умножение (конъюнкция) кодов обоих концов отрезка не равно нулю, то отрезок целиком вне окна, отсечение не нужно, отрезок отбрасывается как тривиально невидимый (отрезок *KL* на рис. 3.2);

- если логическое умножение (конъюнкция) кодов обоих концов отрезка равно нулю, то отрезок подозрительный, он может быть частично видимым (отрезки *CD*, *EF*, *GH*) или целиком невидимым (отрезок *IJ*); для него нужно определить координаты пересечений со сторонами окна и для каждой полученной части определить тривиальную видимость или невидимость. При этом для отрезков *CD* и *IJ* потребуется вычисление одного пересечения, для остальных (*EF* и *GH*) — двух.

При расчете пересечения используется горизонтальность либо вертикальность сторон окна, что позволяет определить координату *X* или *Y* точки пересечения без вычислений.

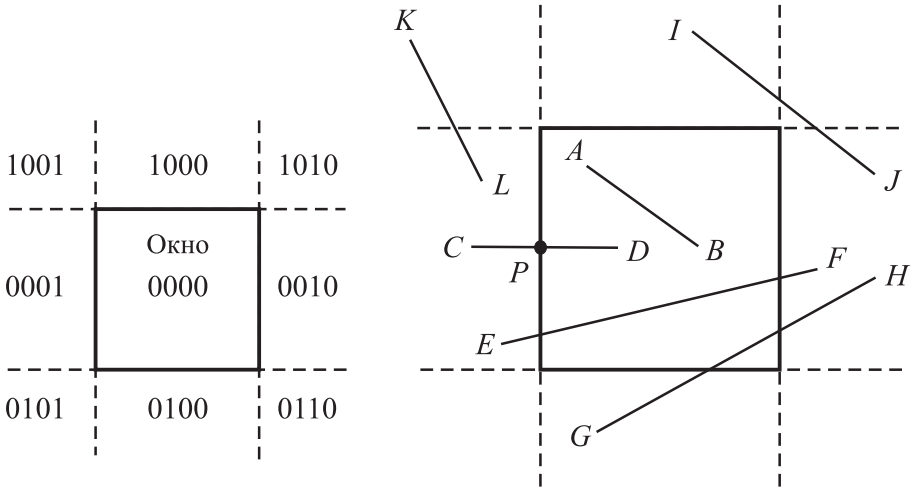


Рис. 3.2 – Окно отсечения и прилегающие к нему части плоскости

При непосредственном использовании описанного выше способа отбора целиком видимого или целиком невидимого отрезка после расчета пересечения потребовалось бы вычисление кода расположения точки пересечения. Для примера рассмотрим отрезок *CD*. Точка пересечения обозначена как *P*. В силу того, что граница окна считается принадлежащей окну, можно просто принять только часть отрезка *PD*, попавшую в окно. Часть же отрезка *CP*, на самом деле оказавшаяся вне окна, потребует дальнейшего рассмотрения, так как логическое И кодов точек *C* и *P* даст 0, т. е. отрезок *CP* нельзя просто отбросить. Для решения этой проблемы Коэн и Сазерленд предложили заменять конечную точку с ненулевым кодом конца на точку, лежащую на стороне окна либо на ее продолжении.

В целом схема алгоритма Коэна—Сазерленда следующая:

1) Рассчитать коды конечных точек отсекаемого отрезка.
2) В цикле повторять пункты 2–6.
3) Если логическое умножение (конъюнкция) кодов конечных точек не равно 0, то отрезок целиком вне окна. Он отбрасывается, и отсечение закончено.
4) Если оба кода равны 0, то отрезок целиком видим. Он принимается, и отсечение закончено.
5) Если начальная точка внутри окна, то она меняется местами с конечной точкой.

6) Анализируется код начальной точки для определения стороны окна, с которой есть пересечение, и выполняется расчет пересечения. При этом вычисленная точка пересечения заменяет начальную точку.
7) Определение нового кода начальной точки.

3.2.2 Алгоритм Лианга—Барского

В 1982 г. Лианг и Барский предложили алгоритмы отсечения прямоугольным окном с использованием параметрического представления для двух-, трех- и четырехмерного отсечения. По утверждению авторов, данный алгоритм в целом превосходит алгоритм Коэна—Сазерленда. Однако в работе показывается, что это утверждение справедливо только для случая, когда оба конца видимого отрезка вне окна и окно небольшое (до 50×50 при разрешении 1000×1000). Как уже говорилось, при 2D отсечении прямые отсекаются по 2D области, называемой окном отсечения. В частности, внутренняя часть окна отсечения может быть выражена с помощью следующих неравенств:

$$\begin{aligned} x_{\text{лев}} \leqslant x \leqslant x_{\text{прав}}; \\ y_{\text{верх}} \leqslant y \leqslant y_{\text{низ}}. \end{aligned} \tag{3.1}$$

Продолжим каждую из четырех границ окна до бесконечных прямых. Каждая из таких прямых делит плоскость на две области. Назовем «видимой частью» ту, в которой находится окно отсечения. Видимой части соответствует внутренняя сторона линии границы. Невидимой части плоскости соответствует внешняя сторона линии границы. Таким образом, окно отсечения может быть определено как область, которая находится на внутренней стороне всех линий границ.

Отсекаемый отрезок прямой может быть преобразован в параметрическое представление следующим образом. Пусть конечные точки отрезка есть V_0 и V_1 с координатами (x_0, y_0) и (x_1, y_1) соответственно (рис. 3.3).

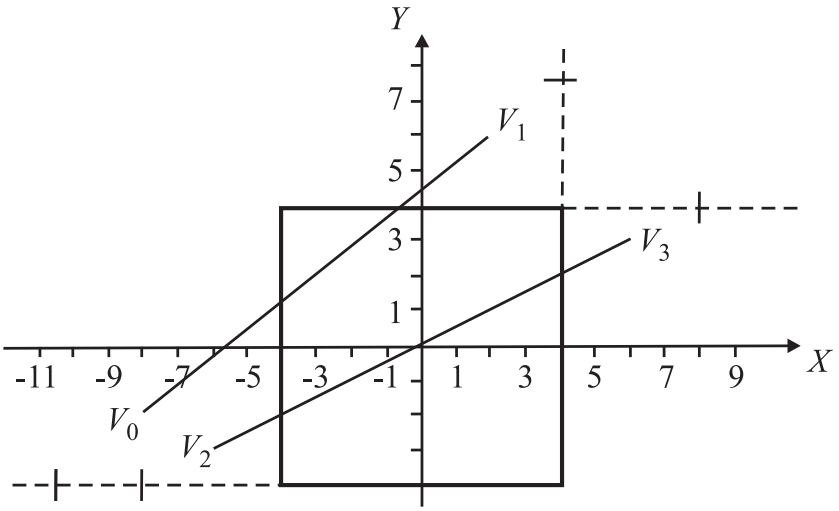


Рис. 3.3 – Пример расчета отсечения в алгоритме Лианга—Барского

Тогда параметрическое представление линии может быть задано следующим образом:

$$\begin{aligned} x &= x_0 + dx \cdot t; \quad y = y_0 + dy \cdot t. \\ dx &= x_1 - x_0; \quad dy = y_1 - y_0. \end{aligned} \tag{3.2}$$

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

$$V(t) = V_0 + (V_1 - V_0) \cdot t. \tag{3.3}$$

Для точек V_0 и V_1 параметр t равен 0 и 1 соответственно. Меняя t от 0 до 1, перемещаемся по отрезку V_0V_1 от точки V_0 к точке V_1. Изменяя t в интервале $[-1, 1]$, получаем бесконечную прямую, ориентация которой — от точки V_0 к точке V_1.

Подставляя параметрическое представление, заданное уравнениями (3.2) и (3.3), в неравенства (3.1), получим следующие соотношения для частей удлиненной линии, которая находится в окне отсечения:

$$\begin{aligned} -dx \cdot t &\leqslant x_0 - x_{\text{лев}}; \quad dx \cdot t \leqslant x_{\text{прав}} - x_0; \\ -dy \cdot t &\leqslant y_0 - y_{\text{низ}}; \quad dy \cdot t \leqslant y_{\text{верх}} - y_0. \end{aligned} \tag{3.4}$$

Заметим, что соотношения (3.4) — неравенства, описывающие внутреннюю часть окна отсечения, в то время как равенства определяют его границы. Рассматривая неравенства (3.4), видим, что они имеют одинаковую форму вида:

$$P_i \cdot t \leqslant Q_i, \tag{3.5}$$

где $i = 1, 2, 3, 4$.

Здесь использованы следующие обозначения:

$$\begin{aligned} P_1 &= -dx; \quad Q_1 = x_0 - x_{\text{лев}}; \quad P_3 = -dy; \quad Q_3 = y_0 - y_{\text{низ}}; \\ P_2 &= dx; \quad Q_2 = x_{\text{прав}} - x_0; \quad P_4 = dy; \quad Q_4 = y_{\text{верх}} - y_0. \end{aligned} \tag{3.6}$$

По определению внутренней и внешней стороны линии границы заметим, что каждое из неравенств (3.5) соответствует одной из граничных линий (левой, правой, нижней и верхней соответственно) и описывает ее видимую сторону. Удлиним V_0V_1 в бесконечную прямую. Тогда каждое неравенство задает диапазон значений параметра t, для которых эта удлиненная линия находится на видимой стороне соответствующей линии границы. Более того, конкретное значение параметра t для точки пересечения есть $t = Q_i/P_i$. Причем знак Q_i показывает, на какой стороне соответствующей линии границы находится точка V_0. А именно, если $Q_i > 0$, тогда V_0 находится на видимой стороне линии границы, включая и ее. Если же $Q_i < 0$, тогда V_0 находится на невидимой стороне.

Рассмотрим P_i в соотношениях (3.6). Возможны три случая:

- $P_i < 0$, тогда соответствующее неравенство становится:

$$t \geqslant \frac{Q_i}{P_i}. \tag{3.7}$$

  Очевидно, что диапазон значений параметра t, для которых удлиненная линия находится на видимой стороне соответствующей граничной линии, имеет минимум в точке пересечения направленной удлиненной линии, заданной вектором V_0V_1, и идущей с невидимой на видимую сторону граничной линии.

- $P_i > 0$, тогда соответствующее неравенство становится:

$$t \leqslant \frac{Q_i}{P_i}. \tag{3.8}$$

  Так как значения параметра t только на границе равны Q_i/P_i, а в остальной видимой части меньше Q_i/P_i, то значение параметра t имеет максимум на границе.

- $P_i = 0$, тогда соответствующее неравенство превращается в

$$0 \leqslant Q_i.$$

Геометрически, если $P_i = 0$, то нет точек пересечения удлиненной линии, определяемой точками V_0V_1, с линиями границы. Более того, если $Q_i < 0$, то удлиненная линия находится на внешней стороне линии границы, а при $Q_i \geqslant 0$ находится на внутренней стороне (включая ее). В последнем случае отрезок V_0V_1 может быть видим или нет, в зависимости от того, где находятся точки V_0V_1 на удлиненной линии. В предыдущем же случае нет видимого сегмента, так как удлиненная линия вне окна, т. е. это случай тривиального отбрасывания.

Итак, рассмотрение неравенств дает диапазон значений параметра t, для которого удлиненная линия находится внутри окна отсечения.

3.3 Операции с изображением на уровне растра

Так как в подавляющем большинстве графические устройства являются растровыми, то есть представляют изображение в виде прямоугольной матрицы (сетки, целочисленной решетки) пикселей, то возникает необходимость в растровых алгоритмах [20].

Процесс генерации растрового образа геометрического объекта принято называть ***растрированием*** *(rasterization).*

Для отображения на экране геометрический объект представляется в виде набора типовых элементов, растрирование которых может быть выполнено с высокой скоростью встроенными аппаратными средствами графического процессора. Такими элементами, называемыми графическими примитивами вывода, в зависимости от архитектуры процессора могут быть отрезок прямой, дуга кривой второго порядка, параметрические или алгебраические сплайны и некоторые другие виды кривых и поверхностей.

Для представления на экране растрового дисплея любой кривой единичной толщины необходимо найти ее цепочный код или, что одно и то же, найти координаты близких к линии смежных точек на целочисленной сетке [19].

В дальнейшем будем считать, что на плоскости имеется квадратная сетка с шагом 1, причем узлы целочисленной решетки являются центрами соответствующих квадратных ячеек сетки. Другими словами, узлы растра окружены «единичными» квадратными окрестностями «радиуса» 1/2.

Инициализации точки растра с координатами (i,j) соответствует закраска каким-либо цветом ее квадратной окрестности (рис. 3.4). Точки на плоскости называются непосредственными соседями (4-соседями), если у них отличаются только x-координаты или только y-координаты, причем только на 1 (рис. 3.5, *а*).

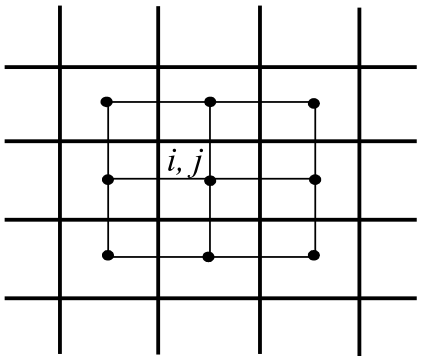
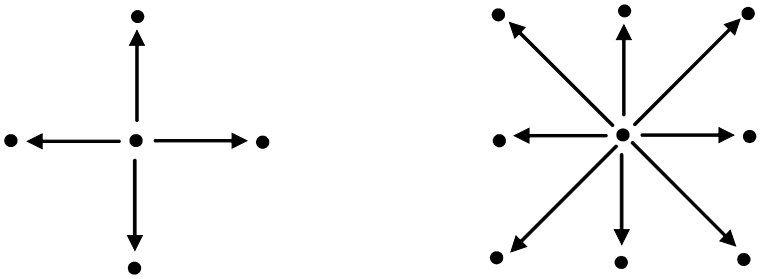


Рис. 3.4 – Инициализация точки растра

Точки на плоскости называются косвенными соседями (8-соседями) если у них отличаются x-координаты или y-координаты, но не более чем на 1 (рис. 3.5, *б*).



а) *б)*

Рис. 3.5 – Соседние точки на плоскости: *а* — непосредственные, *б* — косвенные

Всякая точка на плоскости имеет четырех непосредственных соседей и восемь косвенных соседей. Если точки являются непосредственными соседями, то их квадратные окрестности имеют общую сторону. Квадратные окрестности косвенных соседей имеют общую сторону или общую вершину.

Сильносвязным путем *(4-путем) на плоскости называется множество точек $A_1, A_2, \ldots, A_n$, для которых точки A_i и A_{i+1} являются непосредственными соседями для $i = 1, 2, \ldots, n-1$.*

Слабосвязным путем *(8-путем) на плоскости называется множество точек $A_1, A_2, \ldots, A_n$, для которых точки A_i и A_{i+1} являются косвенными соседями для $i = 1, 2, \ldots, n-1$.*

Путь называется замкнутым, если $A_1 = A_n$.

Множество на целочисленной решетке называется сильносвязным (4-связным), если любые две точки его можно соединить сильносвязным путем.

Множество на целочисленной решетке называется слабосвязным (8-связным), если любые две точки его можно соединить слабосвязным путем.

Простой кривой на плоскости называется множество, у которого все точки, за исключением двух, имеют двух соседей. Две исключительные точки имеют ровно по одному соседу.

Простой замкнутой кривой на плоскости называется множество, у которого все точки имеют двух соседей. Простая замкнутая слабосвязная кривая разбивает плоскость на два сильносвязных множества. Простая замкнутая сильносвязная кривая разбивает плоскость на два слабосвязных множества.

3.3.1 Алгоритм вывода прямой линии

Прямая линия — это один из графических примитивов, из которого складываются более сложные изображения. Отрезки прямых линий чаще всего используется в современных графических системах по сравнению с другими примитивами, не считая точек (пикселей). Несмотря на, то что прямая — это одна из простейших геометрических фигур, вывод отрезка представляет собой достаточно сложную задачу. Предположим, что у нас есть координаты концов отрезка: x_1, y_1, x_2, y_2. Необходимо закрасить все пиксели вдоль линии в определенный цвет.

Наиболее простой случай — вывод горизонтальных и вертикальных линий. Для отображения этих линий достаточно составить цикл по соответствующей координате с инкрементом в один пиксель. В этих циклах выполняются простейшие операции над целыми и сравнение (если не дошли до конца отрезка), поэтому рисование отрезка выполняется достаточно быстро. Конечно, этот алгоритм можно еще улучшить, например можно работать напрямую с памятью, но это не дает значительного выигрыша в быстродействии. Следует отметить, что горизонталь обычно рисуется быстрее, т. к. информация записывается в последовательно стоящие ячейки памяти, а при отображении вертикали обращение идет к ячейкам памяти, отстоящим друг от друга на ширину экрана.

Следующий шаг — построение наклонных линий. Для рисования линии общего вида необходимо вычислять координаты каждого пикселя. Известно несколько алгоритмов расчетов координат точек линии. Один из них — это прямое вычисление координат.

3.3.2 Прямое вычисление координат

Для реализации алгоритма с прямым вычислением координат точек применяется уравнение прямой:

$$y = k \cdot x + b,$$

где k — тангенс угла наклона прямой к оси ox, b — координата по y точки пересечения оси oy.

Через известные нам точки начала и конца отрезка коэфициенты k и b вычисляются следующим образом:

$$k = \frac{(y_2 - y_1)}{(x_2 - x_1)}; \quad b = y_1 - k \cdot x_1.$$

Непосредсвенная реализация данного алгоритма неэффективна, т. к. требует больших затрат процессорного времени из-за операций с вещественными числами.

3.4 Инкрементные алгоритмы

Брезенхэм предложил подход, позволяющий разрабатывать так называемые инкрементные алгоритмы растеризации. Основной целью для разработки таких алгоритмов было построение циклов вычисления координат на основе только целочисленных операций сложения/вычитания без использования умножения и деления.

Инкрементные алгоритмы выполняются как последовательное вычисление координат соседних пикселей путем добавления приращений координат. Приращения рассчитываются на основе анализа функции погрешности. В цикле выполняются только целочисленные операции сравнения и сложения/вычитания. Достигается повышение быстродействия для вычислений каждого пикселя по сравнению с прямым способом.

Процедура на языке Паскаль, реализующая алгоритм Брезенхема

```
Procedure Bresenham(x1,y1,x2,y2,Color: integer);
Var dx,dy,incr1,incr2,d,x,y,xend: integer;
begin
    dx:= ABS(x2-x1);
    dy:= Abs(y2-y1);
    d:=2*dy-dx; {начальное значение для d}
    incr1:=2*dy; {приращение для d < 0}
    incr2:=2*(dy-dx); {приращение для d >=0}
    if x1 < x2 then {начинаем с точки с меньшим знач. x}
    begin
        x:=x2;
        y:=y2;
        xend:=x1;
    end
    else
    begin
        x:=x1;
        y:=y1;
        xend:=x2;
    end;
    PutPixel(x,y,Color); {первая точка отрезка}
    While x < xend do
        begin
            x:=x+1;
            if d < 0 then
```

```
            d:=d+incr1 {выбираем нижнюю точку}
        else
        begin
            y:=y+1;
            d:=d+incr2; {выбираем верхнюю точку, y-возрастает}
        end;
    PutPixel(x,y,Color);
    end; {while}
end; {procedure}
```

Данный алгоритм использует только целые числа, поэтому имеет достаточно высокую скорость работы.

3.5 Алгоритмы вывода фигур

Фигурой здесь будем считать плоский геометрический объект, который состоит из линий контура и точек, которые содержатся внутри контура.

В общем случае линий контура может быть несколько — когда объект имеет внутри пустоты. В этом случае для описания таких фигур необходимы два и более контуров (рис. 3.6).

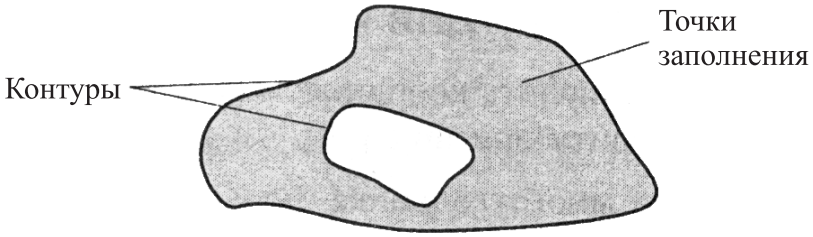


Рис. 3.6 – Пример фигуры

В некоторых графических системах одним объектом может считаться и более сложная многоконтурная фигура — совокупность островов с пустотами.

Графический вывод фигур разделяется на две задачи: вывод контура и вывод точек заполнения. Поскольку контур представляет собой линию, то вывод контура проводится на основе алгоритмов вывода линий. В зависимости от сложности контура это могут быть отрезки прямых, кривых или произвольная последовательность соседних пикселей [13].

3.6 Заполнение сплошных областей

Вопрос о заполнении внутренности сплошной области занимает важное место среди задач растровой графики. Большинство задач о заполнении двухмерной фигуры может быть отнесено к одному из двух типов: заполнение внутренности многоугольника, заданного своими вершинами или ребрами, и заполнение внутренности области, ограниченной замкнутым контуром, представленным своей растровой разверткой.

3.6.1 Тест принадлежности точки многоугольнику

Будем понимать под многоугольником фигуру, ограниченную на плоскости простой (несамопересекающейся) замкнутой ломаной. Сама ломаная задается набором своих вершин $A_i(x_i, y_i)$, $i = 1, \ldots, n$, причем соседние точки в этом списке являются смежными вершинами ломаной. Задача состоит в том, чтобы получить растровую развертку многоугольника, т. е. инициировать его внутренние точки.

Начнем с обсуждения задачи о локализации точки относительно многоугольника. Решение этой задачи даст возможность эффективно определять, является ли точка внутренней или внешней по отношению к нему.

Теорема Жордана утверждает, в частности, что простая (т. е. не имеющая самопересечений) замкнутая плоская ломаная разбивает плоскость на две связные компоненты — ограниченную, которая является внутренностью многоугольника, и неограниченную, которая является внешней по отношению к многоугольнику.

Алгоритм должен уметь различать внутренние и внешние точки плоскости. Обозначим ребра многоугольника через E_i:

$$E_i : [A_i, A_{i+1}],\ i = 1, \ldots, n.$$

Пусть $P(x, y)$ — некоторая точка плоскости, не лежащая на ломаной, и нужно определить, принадлежит она этому многоугольнику или нет.

Проведем через точку P горизонтальную полупрямую с правым концом в точке P. Так как ломаная ограничена, то всегда легко найти на этой полупрямой достаточно удаленную точку Q, которая заведомо не принадлежит многоугольнику. Если отрезок QP не имеет пересечений с границей многоугольника, то точки Q и P лежат в одной компоненте связности и, следовательно, точка P — внешняя.

Рассмотрим случай, когда отрезок QP пересекает ломаную (рис. 3.7). Будем двигаться от точки Q в направлении к точке P. Миновав первое пересечение отрезка и границы, мы попадем внутрь многоугольника. Миновав следующее пересечение отрезка и границы, мы окажемся снаружи многоугольника, и так далее.

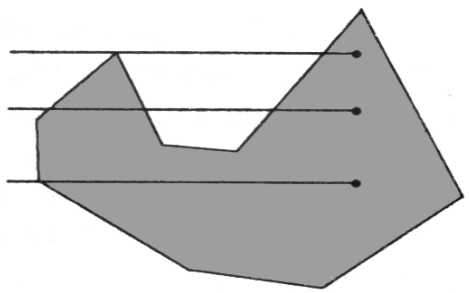


Рис. 3.7 – Тест принадлежности точки многоугольнику

Легко видеть, что если мы встретим на своем пути четное число пересечений, то точка P будет внешней точкой многоугольника. Если же число пересечений окажется нечетным, то точка P будет внутренней точкой многоугольника. Важно только удостовериться, что пересечения отрезка с границей были существенными, т. е. отрезок действительно пересек ломаную, а не просто касался ее в одной из вершин.

Для выявления существенных пересечений можно воспользоваться следующим правилом:

- Пересечения отрезка с горизонтальными ребрами игнорируются.
- При подсчете числа пересечений негоризонтальных ребер ломаной с отрезком PQ пересечение игнорируется, если точкой пересечения является верхняя вершина ребра, и засчитывается в любом другом случае. С учетом этого соглашения касание отрезка PQ с ломаной в точках максимума игнорируется, а в точках минимума считается дважды. Тем самым несущественные пересечения не изменяют четности общего числа пересечений.
- Если же существенное пересечение имеет место в вершине ломаной (верхнее ребро пересекает отрезок PQ в нижней вершине, а нижнее ребро — в верхней вершине, и по нашему соглашению принимается во внимание лишь первое пересечение), то число подсчитанных пересечений увеличивается на единицу.

3.6.2 Заполнение многоугольников

Для заполнения внутренности многоугольника можно было бы воспользоваться описанным выше тестом принадлежности, перебирая последовательно точки растра и инициируя те из них, которые лежат внутри многоугольника. Такой алгоритм прост и понятен, однако в смысле временных затрат очень неэффективен. Его можно несколько улучшить, поместив многоугольник внутрь минимального объемлющего прямоугольника со сторонами, параллельными осям координат, и анализировать лишь те точки, которые попали внутрь прямоугольника.

Ломаная, ограничивающая многоугольник, разбивает всякую горизонтальную прямую на чередующиеся интервалы, лежащие внутри или снаружи многоугольника. Для определения концов этих интервалов можно поступить следующим образом.

Зафиксируем горизонтальную прямую L, на которой предполагается определить местоположение интервалов, находящихся внутри многоугольника, ищем точки пересечения с этой прямой, если они есть, каждого ребра многоугольника. Для проверки наличия пересечения предварительно убедимся в том, что концы ребра расположены по разные стороны от прямой.

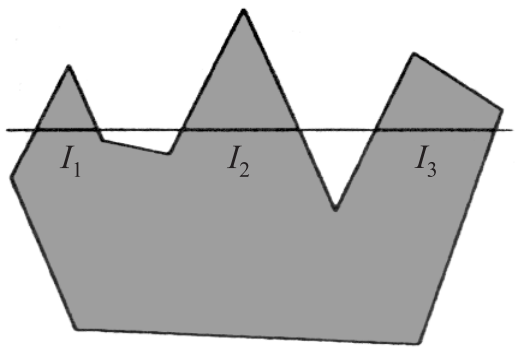


Рис. 3.8 – Заполнение многоугольника

При поиске точек пересечения будем пользоваться правилами, примененными в тесте принадлежности. Тогда кратные пересечения с контуром многоугольника в его вершинах будут правильно учтены. Далее упорядочим полученные точки слева направо и сгруппируем их попарно. Эти пары и будут являться концами интервалов, лежащих внутри многоугольника и подлежащих закраске (рис. 3.8).

Изложенная схема заполнения многоугольника носит название заполнения в порядке сканирования строк, а сам алгоритм относится к типу алгоритмов построчного сканирования.

3.6.3 Стиль заполнения. Кисть. Текстура

При выводе фигур могут использоваться различные стили заполнения. Для обозначения стилей заполнения, отличных от сплошного стиля, используют такие понятия, как кисть и текстура. Их можно считать синонимами, однако понятие текстуры обычно используется применительно к трехмерным объектам, а кисть — для изображения двумерных объектов [13].

Текстура *— это стиль заполнения, закрашивание, которое имитирует сложную рельефную объемную поверхность, выполненную из какого-то материала.*

Для описания алгоритмов заполнения фигур с определенным стилем используем тот же способ, что и для описания алгоритмов рисования линий. Например, в алгоритме вывода полигонов пиксели заполнения рисуются в теле цикла горизонталей, а все другие операции предназначены для подсчета координат (x, y) этих пикселей. Сплошное заполнение означает, что цвет всех пикселей одинаков.

Преобразование координат пикселя заполнения (x, y) в координаты внутри образца кисти можно выполнить таким образом:

$$x_T = x \cdot \bmod m,$$
$$y_T = y \cdot \bmod n,$$

где m, n — размеры растра кисти соответственно по горизонтали и вертикали. При этом координаты (x_T, y_T) будут в диапазоне $x_T = 0 \ldots m - 1$, $y_T = 0 \ldots n - 1$ при любых значениях x и y. Таким образом, обеспечивается циклическое копирование фрагментов кисти внутри области заполнения фигуры (рис. 3.9).

Удобно, когда размеры кисти равны степени двойки. В этом случае вместо операций взятия остатка (mod) можно использовать более быстродействующие для цифровых компьютеров поразрядные двоичные операции. Ниже приведен пример вычисления остатка от деления на 16.

Для пикселей текстур часто употребляется название тексель.

Тексель *— минимальная единица текстуры.*

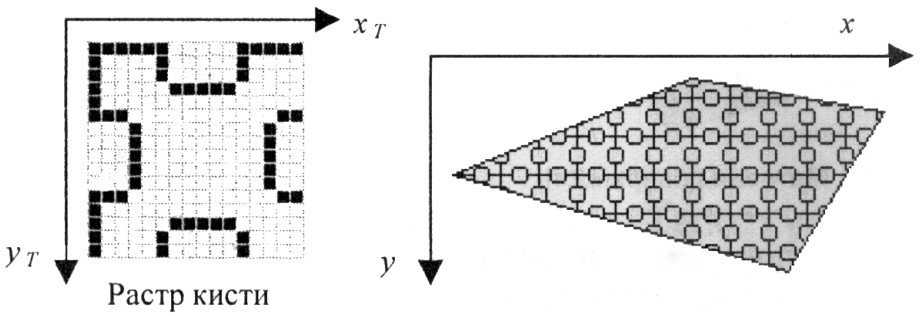


Рис. 3.9 – Копирование растра кисти

Растровые текстуры и кисти широко используются в современной компьютерной графике.

Для отображения трехмерных объектов часто используются полигональные поверхности, каждая грань отображается с наложенной текстурой. Поскольку объекты обычно показываются с разных ракурсов — повороты, изменения размеров и тому подобное, то необходимо соответственно трансформировать и каждую грань с текстурой. Для этого используются проективные текстуры.

Принято подразделять текстуры на два вида:

1) Упорядоченная. Изменение тона в виде правильных или почти правильных геометрических рисунков (кирпичная кладка, кафельная облицовка, шахматный рисунок).
2) Стохастическая (случайная). Соответствует естественным объектам и, как правило, является следствием шероховатости поверхности.

Один из основных принципов формирования текстуры — перенос регулярного или стохастического рисунка на поверхность объекта. В большинстве практических случаев такие преобразования линейны, а коэффициент преобразований может быть вычислен из соотношений между известными точками в системах координат. Характерные точки узора из пространства текстуры переносятся в объектное пространство, затем в пространство изображений, и определенным образом соединяются линии рисунка. В процедурах нанесения текстуры часто используются фрагментирование как узоров, так и поверхностей.

При переносе на поверхность предмета текстуры, представляющей непрерывное регулярное или случайное поле яркости, наиболее подходящим является метод обратной трассировки лучей. Центр пикселя — изображение проецируется на поверхность предмета, и по координатам точки на поверхности определяется соответствующая ей точка в пространстве текстуры. Для устранения эффектов, вызванных пространственной дискретизацией, используются процедуры сглаживания. Одной из возможных процедур сглаживания является трассировка четырех точек, соответствующих углам пикселя, и использование среднего значения яркости текстуры для этих четырех точек.

Для поверхностей со значительной кривизной (шар, эллипсоид) перенос на них даже стохастической текстуры не обеспечивает реалистического вида. Причина этого в том, что текстура, сформированная без учета формы объекта, не пере-

дает изменение освещенности, обусловленной рельефом поверхности. Для реальных шероховатых поверхностей вектор нормали содержит небольшую случайную составляющую, которая определяет характер изменения освещенности в изображении.

3.7 Методы улучшения растровых изображений

Рассмотрим некоторые из существующих методов улучшения качества изображений, которые основываются на субъективном восприятии разрешающей способности и количества цветов. При одних и тех же значениях технических параметров устройства графического вывода можно создать иллюзию увеличения разрешающей способности или количества цветов. Причем субъективное улучшение одной характеристики происходит за счет ухудшения другой [20].

3.7.1 Устранение ступенчатого эффекта

В растровых системах при невысокой разрешающей способности (меньше 300 dpi) существует проблема ступенчатого эффекта (*aliasing*) — при большом шаге сетки растра пиксели линий образуют как бы ступени лестницы.

Алиасинг *— дефекты изображения, эффект «ступенчатости» изображения, против которого используются различные алгоритмы сглаживания.*

Рассмотрим это на примере отрезка прямой линии. Вообще говоря, растровое изображение объекта определяется алгоритмом закрашивания пикселей, соответствующих телу изображаемого объекта. Разные алгоритмы могут дать существенно отличающиеся варианты растрового изображения одного и того же объекта. Можно сформулировать условие корректного закрашивания следующим образом — если в контур изображаемого объекта попадает больше половины площади ячейки сетки растра, то соответствующий пиксель закрашивается цветом объекта (C), иначе — пиксель сохраняет цвет фона ($C_ф$).

На рис. 3.10 показано растровое изображение толстой прямой линии, на которую для сравнения наложен идеальный контур исходной линии.

Устранение ступенчатого эффекта называется на английском языке *«antialiasing»*.

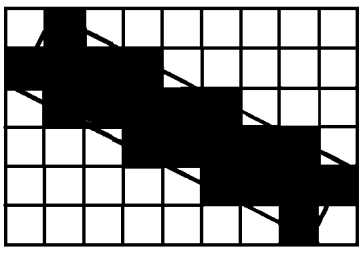


Рис. 3.10 – Растровое изображение прямой линии

Антиалисинг *— технология, убирающая эффект «ступенчатости» с использованием различных алгоритмов сглаживания.*

Для того чтобы растровое изображение линии выглядело более ровным, можно цвет угловых пикселей «ступенек лестницы» заменить определенным оттенком, промежуточным между цветом объекта и цветом фона. Будем вычислять цвет пропорционально части площади ячейки растра, покрываемой идеальным контуром объекта. Если площадь всей ячейки обозначить как S, а часть площади, покрываемой контуром, — S_x, то искомый цвет равняется:

$$C_x = \frac{C \cdot S_x + C_\text{ф} \cdot (S - S_x)}{S}.$$

На рис. 3.11 показано сглаженное растровое изображение, построенное указанным выше методом.

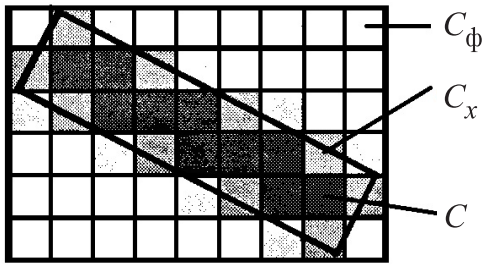


Рис. 3.11 – Сглаженное растровое изображение прямой линии

Методы визуализации сглаженных растровых изображений можно разделить на две группы. Первую группу составляют алгоритмы растеризации для отдельных примитивов — линий, фигур с заполнением. В ходе вывода последовательности примитивов для любого пикселя текущего примитива рассчитывается соответствующий цвет с учетом сглаживания.

Другую группу методов сглаживания составляют методы обработки уже существующего изображения. Для сглаживания растровых изображений часто используют алгоритмы цифровой фильтрации. Один из таких алгоритмов — локальная фильтрация. Она осуществляется путем взвешенного суммирования яркости пикселей, расположенных вокруг текущего обрабатываемого пикселя (рис. 3.12).

Если в ходе обработки новые значения цвета пикселей записываются в первоначальный растр и вовлекаются в вычисления для очередных пикселей, то такую фильтрацию называют *рекурсивной*. При *нерекурсивной фильтрации* в вычисления вовлекаются только прежние значения цвета пикселей. Нерекурсивность можно обеспечить, если новые значения записывать в отдельный массив. Рекурсивный фильтр может дать больший эффект сравнительно с нерекурсивным фильтром — изменение цвета одного пикселя может привести к изменениям пикселей всего изображения.

На рис. 3.12 представлены результаты работы двух вариантов нерекурсивного сглаживающего фильтра, использующего окно (маску) 3×3.

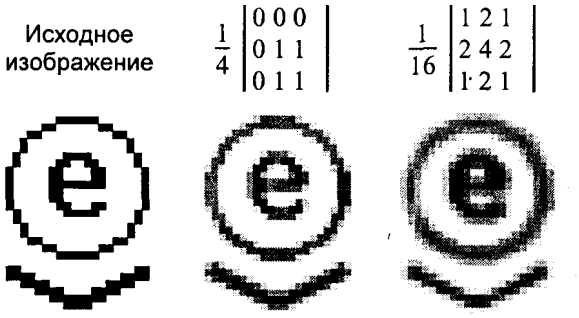


Рис. 3.12 – Два сглаживающих фильтра с маской 3 × 3

Значение нормирующего коэффициента здесь выбрано равным сумме элементов маски. Этим обеспечивается сохранение масштаба яркости преобразованного растра. Отметим, что маска — это не матрица, а массив коэффициентов, располагающихся соответственно пикселям окна. Средний фильтр можно задать и маской 2 × 2 — отбросить нулевые коэффициенты.

При сглаживании цветных изображений можно использовать модель RGB и производить фильтрацию по каждому компоненту.

С помощью локальной цифровой фильтрации можно выполнять довольно разнообразную обработку изображений — повышение резкости, выделение контуров и многое другое.

3.7.2 Дизеринг

Современные растровые дисплеи достаточно качественно воспроизводят миллионы цветов, благодаря чему без проблем можно отображать цветные фотографии. Устройства печати обычно имеют высокую разрешающую способность (dpi), часто на порядок выше, чем дисплеи. Однако они не могут непосредственно воссоздать даже сотню градаций серого цвета для пикселей черно-белых фотографий, не говоря уже о миллионах цветов. В большинстве случаев можно увидеть, что оттенки цветов (для цветных изображений) или градации серого (для черно-белых) имитируются комбинированием, смесью точек. Чем качественнее полиграфическое оборудование, тем меньше отдельные точки и расстояние между ними.

Для устройств печати на бумаге проблема количества красок достаточно важна. В полиграфии для цветных изображений обычно используют три цветных краски и одну черную, что в смеси дает восемь цветов (включая черный цвет и белый цвет бумаги). Встречаются образцы печати с большим количеством красок — например карты, напечатанные с использованием восьми красок, однако такая технология печати намного сложнее.

Если графическое устройство не способно воссоздавать достаточное количество цветов, тогда используют *растрирование*, — независимо от того, растровое это устройство или не растровое. В полиграфии растрирование известно давно. Оно использовалось несколько столетий тому назад для печати гравюр. В гравюрах изображение создается многими штрихами, причем полутоновые градации представляются или штрихами разной толщины на одинаковом расстоянии, или штрихами одинаковой толщины с переменной густотой расположения. Такие способы используют особенности человеческого зрения и в первую очередь — простран-

ственную интеграцию. Если достаточно близко расположить маленькие точки разных цветов, то они будут восприниматься как одна точка с некоторым усредненным цветом. Если на плоскости густо расположить много маленьких разноцветных точек, то будет создана визуальная иллюзия закрашивания плоскости определенным усредненным цветом. Однако если увеличивать размеры точек и (или) расстояние между ними, то иллюзия сплошного закрашивания исчезает — включается другая система человеческого зрения, которая обеспечивает способность различать объекты, подчеркивать контуры.

В компьютерных графических системах часто используют эти методы. Они позволяют увеличить количество оттенков цветов за счет снижения пространственного разрешения растрового изображения. Иначе говоря — это обмен разрешающей способности на количество цветов. В литературе по КГ такие методы растрирования получили название «дизеринг» (*dithering*, разрежение, дрожание).

Простейшим вариантом дизеринга можно считать создание оттенка цвета парами соседних пикселей. Если рассмотреть ячейки из двух пикселей (рис. 3.13), то ячейка номер 1 дает оттенок цвета :

$$C = \frac{C_1 + C_2}{2},$$

где C_1 и C_2 — цвета, которые графическое устройство непосредственно способно воссоздать для любого пикселя. Числовые значения C, C_j и C_2 можно рассчитать в полутоновых градациях или в модели RGB — в отдельности для любого компонента.

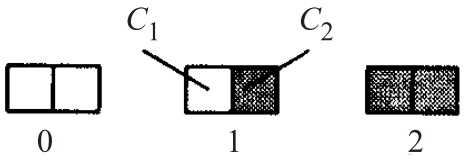


Рис. 3.13 – Ячейки из двух пикселей

Пример растра с использованием ячеек из двух пикселей приведен на рис. 3.14. Как видим, для создания промежуточного оттенка (C) ячейки образуют вертикальные линии, которые очень заметны. Для того чтобы человек воспринял это как сплошной оттенок, необходимо, чтобы угловой размер ячеек был меньше одной угловой минуты. Можно изменять положения таких ячеек в растре, располагая их, например, по диагонали.

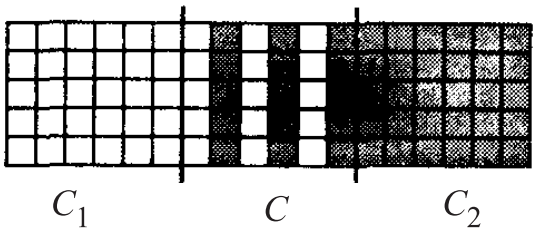


Рис. 3.14 – Простейший дизеринг

Наиболее часто используют квадратные ячейки больших размеров. Приведем пример ячеек размером 2×2 (рис. 3.15). Такие ячейки дают 5 градаций, из них три комбинации (1, 2, 3) образуют новые оттенки.

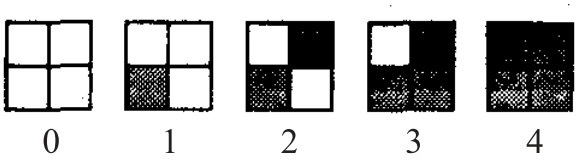


Рис. 3.15 – Ячейки 2 × 2

Предоставим также примеры ячеек других размеров, ячейки 3 × 3 предоставляют 10 градаций (рис. 3.16, *а*) и ячейки 5 × 5 дают 26 градаций (рис. 3.16, *б*).

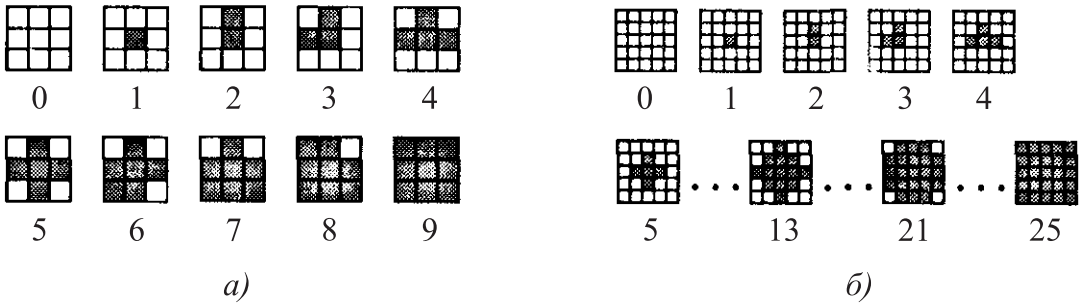


Рис. 3.16 – Градации цвета: *а* — 10 градаций (ячейки 3 × 3), *б* — 25 градаций (ячейки 5 × 5)

Расчет цвета, который соответствует одной из комбинаций пикселей в ячейке, можно выполнить следующим образом: Если пиксели ячейки могут быть только двух цветов (C_1 и C_2), то необходимо подсчитать часть площади ячейки для пикселей каждого цвета. Цвет ячейки C можно оценить соотношением:

$$C = \frac{S_1 C_1 + (S - S_1) C_2}{S} = \frac{S_1 C_1 + S_2 C_2}{S},$$

где S — общая площадь ячейки; S_1 и S_2 — части площади, занятые пикселями цветов C_1 и C_2 соответственно, причем $S_1 + S_2 = S$. Проще всего, если пиксели квадратные, а их размер равняется шагу расположения пикселей. Примем площадь одного пикселя за единицу. В этом случае площадь, занятая пикселями в ячейке, равняется их количеству (рис. 3.17).

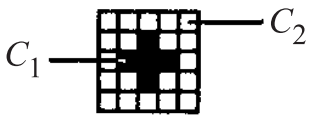
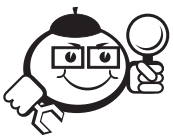


Рис. 3.17 – Определение площади количеством пикселей

$$S = 25, \quad S_1 = 20, \quad S_2 = 5, \quad C = \frac{20C_1 + 5C_2}{25}.$$



Пример

Для ячейки 5×5, изображенной на рисунке 3.16, *б*, рассчитаем цвет C для некоторых цветов C_1 и C_2. Если C_1 — белый цвет $(R_1\, G_1\, B_1) = (255, 255, 255)$, а C_2 — черный $(R_2\, G_2\, B_2) = (0, 0, 0)$, тогда:

$$C = \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} (S_1 R_1 + S_2 R_2)/S \\ (S_1 G_1 + S_2 G_2)/S \\ (S_1 B_1 + S_2 B_2)/S \end{bmatrix} = \begin{bmatrix} 244 \\ 244 \\ 244 \end{bmatrix}.$$

Следовательно, получили светло-серый цвет.

Если C_1 — желтый $(R_1\, G_1\, B_1) = (255, 255, 0)$, а C_2 — красный $(R_2\, G_2\, B_2) = (255, 0, 0)$, то $C = (255, 204, 0)$. Это оттенок оранжевого.

Итак, если в ячейке размерами $n \times n$ использованы два цвета, то с помощью этой ячейки можно получить $n^2 + 1$ разных цветовых градаций. Две комбинации пикселей ячейки, если все пиксели ячейки имеют цвет C_1 или C_2, дают цвет соответственно C_1 или C_2. Все другие комбинации дают оттенки, промежуточные между C_1 и C_2.

Можно считать, что ячейки размером $n \times n$ образуют растр с разрешающей способностью, в n раз меньшей, чем у начального растра, а глубина цвета возрастает пропорционально n^2. Для характеристики изображений, которые создаются методом дизеринга, используют термин — *«линиатура растра»*. Линиатура вычисляется как количество линий (ячеек) на единицу длины — сантиметр, миллиметр, дюйм. В последнем случае единица измерения для линиатуры — lpi (по аналогии с dpi).

Линиатура растра *— характеризует период сетки и обозначает количество линий растра на единицу длины изображения.*

Рассмотрим примеры преобразования растрового изображения размером $p \times q$ с определенной глубиной цвета в другой растр, предназначенный для отображения с помощью графического устройства, в котором используется ограниченное количество основных цветов. В таком случае надо выбрать размеры ячейки $m \times n$, которые обеспечивают достаточное количество цветовых градаций. Потом любой пиксель растра превращается в пиксель растра отображения. Это можно осуществить двумя способами.

Первый способ. Любой пиксель заменяется ячейкой из $m \times n$ пикселей. Это самое точное преобразование по цветам, но размер растра увеличивается и составляет $mp \times nq$ пикселей.

Второй способ. Здесь размеры растра не изменяются. Цвет любого пикселя преобразованного растра вычисляется следующим образом:

- Определяем координаты пикселя (x, y) для преобразуемого растра.
- Находим цвет пикселя (x, y).

- По цвету пикселя находим номер (k) ячейки, наиболее адекватно представляющей этот цвет.
- По координатам (x,y) вычисляем координаты пиксела внутри ячейки:

$$x_k = x \cdot \bmod m,$$
$$y_k = y \cdot \bmod n.$$

- Находим цвет (C) пикселя ячейки с координатами (x_k, y_k).
- Записываем в преобразованный растр пиксель (x,y) с цветом C.

Такой способ можно использовать не для любых вариантов расположения пикселей в ячейках. Конфигурации пикселей должны быть специально разработаны для таких преобразований. Одно из требований можно сформулировать так: если ячейки разработаны на основе двух цветов, например белого и черного, а градации изменяются пропорционально номеру ячейки, то необходимо, чтобы ячейка с номером (i) для более темной градации серого содержала бы все черные пиксели ячейки номер $(i - 1)$.

Рассмотрим пример изображения, созданного на основе ячеек 5 × 5.

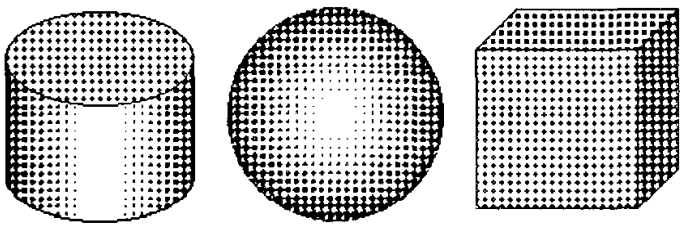


Рис. 3.18 – Пример квадратного растра с ячейками 5 × 5

Для создания такого изображения специально была выбрана небольшая разрешающая способность, чтобы подчеркнуть структуру изображения. Ячейки образуют достаточно заметный квадратный растр (рис. 3.18). Для улучшения восприятия изображения можно использовать другое расположение ячеек, например диагональное. Диагональное расположение можно получить, если сдвигать четные строки ячеек. А для того чтобы получить диагональную структуру растра, подобную той, что используется для печати газет, можно использовать квадратное расположение ячеек другого типа

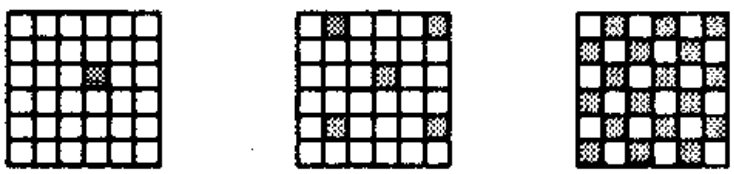


Рис. 3.19 – ЧМ-ячейки 6 × 6

Для всех приведенных выше примеров дизеринга ячейки образуют точки переменного размера с постоянным шагом. Однако часто используется другой подход — переменная плотность расположения точек постоянного размера. Такой способ получил название частотной модуляции (ЧМ) (рис. 3.19).

Положительная черта способа ЧМ — меньшая заметность структуры растра. Однако его использование усложнено в случае, когда размер пикселей больше, чем шаг. Начиная с определенной плотности, пиксели смыкаются. Кроме того, на дискретном растре невозможно обеспечить плавное изменение плотности (частоты), в особенности для ячеек небольшого размера. Рассмотрим пример ячеек 5 × 5, реализующих ЧМ-дизеринг (рис. 3.20).

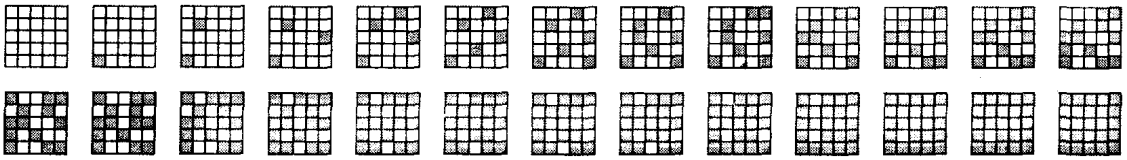


Рис. 3.20 – Набор ЧМ-ячеек 5 × 5

Для изображений, созданных методом ЧМ-дизеринга, растрирование менее заметно (рис. 3.21).

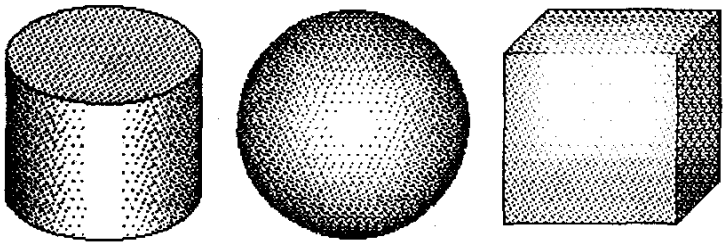


Рис. 3.21 – Диагональное расположение ЧМ-ячеек 5 × 5

Общим недостатком методов, использующих регулярное расположение одинаковых ячеек, является то, что всегда образуется текстура. Одной из важных задач исследований в этой области считалась разработка таких вариантов ячеек, которые обуславливают наименее заметную растровую структуру (кроме тех случаев, когда, наоборот, такую структуру нужно подчеркнуть для создания изображения в стиле гравюры).

Другой разновидностью дизеринга являются методы, в которых вообще не используются ячейки. Одним из популярнейших методов дизеринга в настоящее время является «error diffusion» — метод *Флойда—Стейнберга*. Он обеспечивает высокое качество изображений и часто используется в драйверах принтеров и графических редакторах (рис. 3.22).

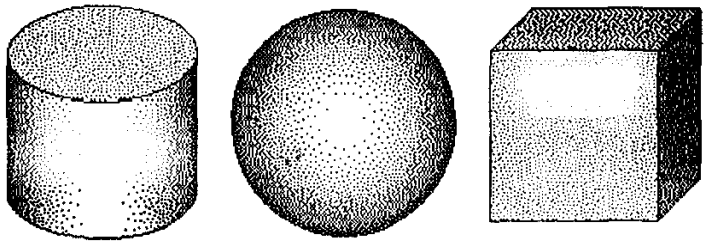


Рис. 3.22 – Метод дизеринга Флойда—Стейнберга

Рассмотрим алгоритм Флойда—Стейнберга. При обработке первой строки растра для первого пикселя цвет (C) заменяется на ближайший из возможных (X). Например, пиксель серого цвета заменяется соответствующим черным или белым. Для этого пикселя вычисляется ошибка $E = C - X$. Эта ошибка в пропорции (7/16, 5/16, 3/16, 1/16) распределяется по соседним пикселям. При обработке следующего пикселя рассматривается уже сумма его собственного цвета плюс значение ошибки, которое дошло к этому пикселю. Как распространяется ошибка — это зависит от направления сканирования строки.

Для уменьшения вероятности образования регулярных узоров рекомендуется соседние строки сканировать в противоположных направлениях — «змейкой». Этот метод был предложен для градаций серого, однако он с успехом используется, например, для преобразования 24-битных изображений в 256-цветные.

Контрольные вопросы по главе 3

1) Какой двумерный алгоритм отсечения использует операцию логического умножения?
2) На сколько областей разделено окно отсечения и прилегающие к нему плоскости согласно алгоритму двумерного отсечения Коэна-Сазерленда?
3) Сколько косвенных соседей имеет каждая точка на плоскости?
4) К какому виду текстуры относится текстура «шахматная доска»?
5) Какая их единиц измерения может быть отнесена к единице измерения линиатуры растра?

Глава 4

МЕТОДЫ И АЛГОРИТМЫ ТРЕХМЕРНОЙ ГРАФИКИ

4.1 Визуализация трехмерных изображений

В самом названии рассматриваемой области — «трехмерная графика» — заложено указание на то, что нам предстоит иметь дело с тремя пространственными измерениями, в обобщенном виде — шириной, высотой и глубиной. Однако термин «трехмерная графика» все же является искажением истины. Трехмерная компьютерная графика имеет дело всего лишь с двумерными проекциями объектов воображаемого трехмерного мира. В компьютерной графике объекты существуют лишь в памяти компьютера. Они не имеют физической формы — это не более чем совокупность математических уравнений и движения электронов в микросхемах. На рис. 4.1. приведена модель процесса визуализации трехмерных изображений.

Рис. 4.1 – Модель процесса визуализации трехмерных изображений

Поскольку объекты, о которых идет речь, не могут существовать вне компьютера, единственным способом увидеть их является добавление новых математических уравнений, описывающих источники света и различные спецэффекты для придания реалистичности создаваемого изображения [5].

4.2 Виды проектирования

Проектирование *— отображение точек, заданных в системе координат с размерностью N, в точки в системе меньшей размерности n, где N < n.*

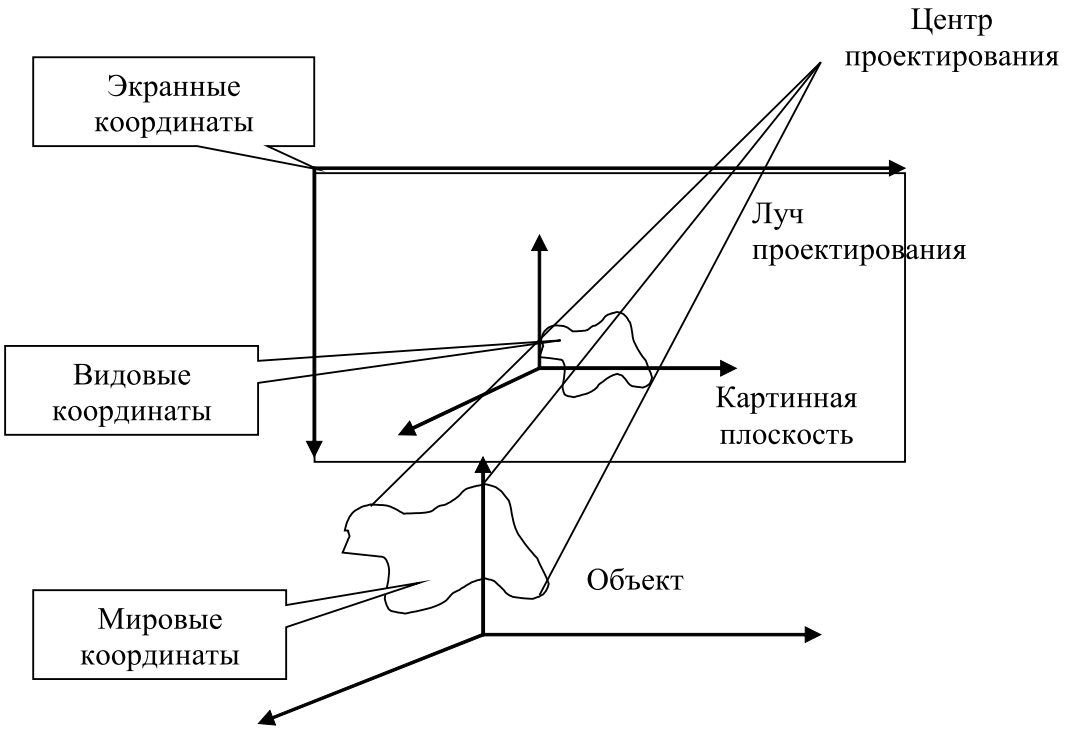


Рис. 4.2 – Проектирование

Проекция трехмерного объекта (представленного в виде совокупности точек) строится при помощи прямых проектирующих лучей, которые называются проекторами и выходят из центра проектирования, проходят через каждую точку объекта и, пересекая картинную плоскость, образуют проекцию (рис. 4.2).

Проектирующие лучи *— отрезки прямых, идущих из центра проекции через каждую точку объекта до пересечения с картинной плоскостью (плоскостью проекции).*

На рис. 4.3 приведена классификация плоских проекций.

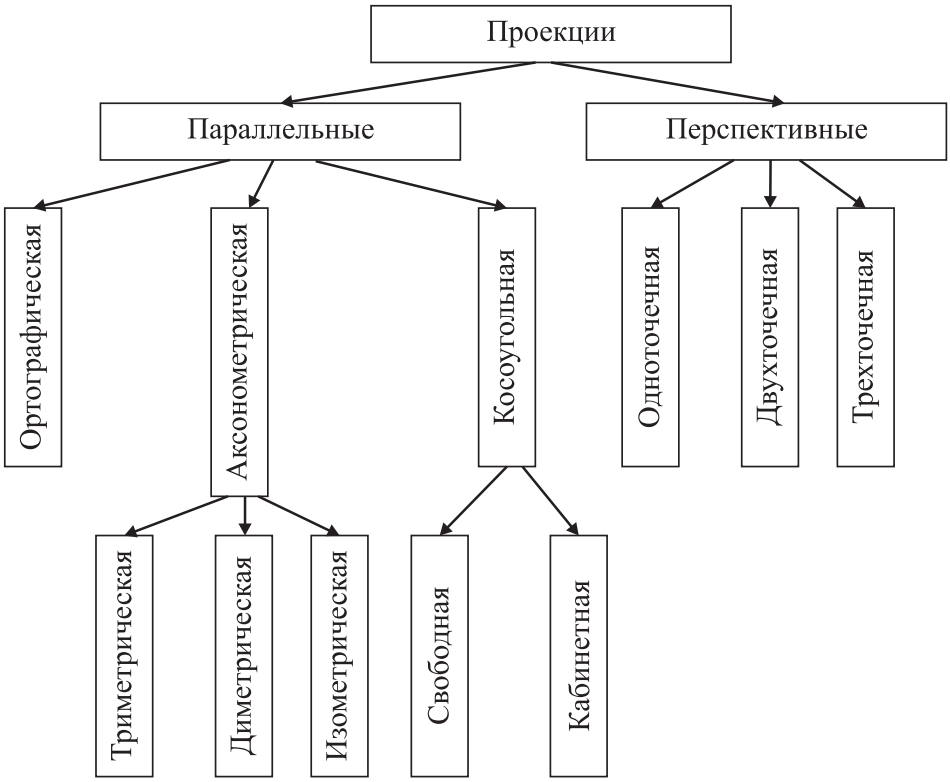


Рис. 4.3 – Классификация плоских проекций

В компьютерной графике наиболее распространены параллельная (рис. 4.4, *а*) и перспективная (рис. 4.4, *б*) проекции.

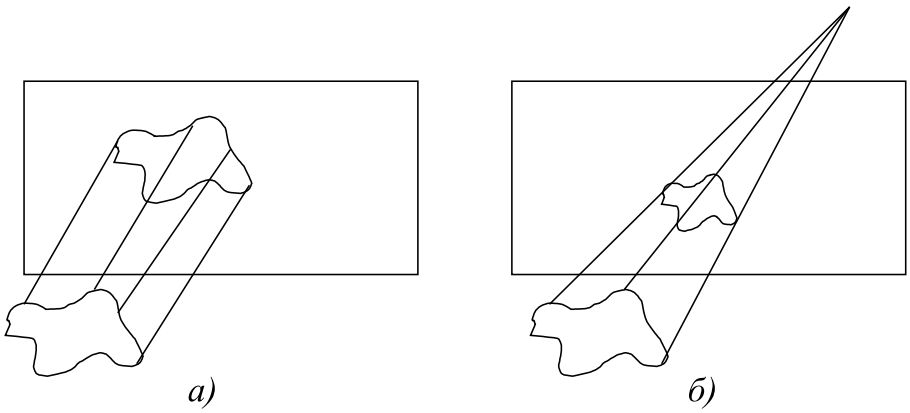


Рис. 4.4 – Типы проекций: *а* — параллельная проекция, *б* — перспективная проекция

4.2.1 Параллельное проектирование

В параллельной проекции лучи проектирования параллельны друг другу. На рис. 4.5 приведен пример параллельной ортографической проекции детали.

Ортографической проекцией *называется проекция, в которой картинная плоскость совпадает с одной из координатных плоскостей или параллельна ей.*

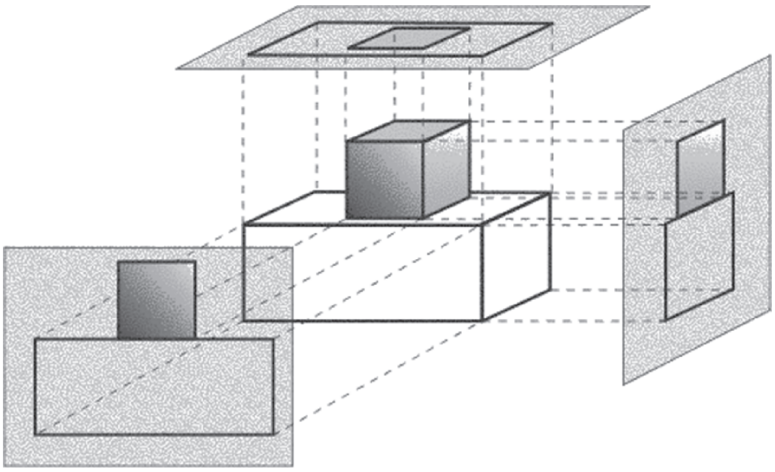


Рис. 4.5 – Ортографическая проекция

Матрица проектирования вдоль оси X на плоскость YOZ имеет вид:

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

В случае если плоскость проектирования параллельна координатной плоскости, необходимо умножить матрицу $[P_x]$ на матрицу переноса:

$$[P_x^{\lambda}] = [P_x] \cdot [T_x] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \lambda & 0 & 0 & 1 \end{bmatrix}.$$

Аналогично записываются матрицы ортографического проектирования вдоль других координатных осей:

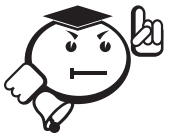
$$[P_y^{\mu}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \mu & 0 & 1 \end{bmatrix}, \quad [P_z^{\nu}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \nu & 1 \end{bmatrix}.$$

Аксонометрической проекцией *называется проекция, у которой проектирующие прямые перпендикулярны картинной плоскости, сама картинная плоскость может располагаться в пространстве произвольным образом.*

Различают три вида аксонометрических проекций, в зависимости от взаимного расположения плоскости проецирования и координатных осей:

1) Триметрия — нормаль к картинной плоскости образует с ортами координатных осей попарно различные углы.
2) Диметрия — два угла между нормалью картинной плоскости и координатными осями равны.
3) Изометрия — все три угла между нормалью картинной плоскости и координатными осями равны.

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



Из курса аналитической геометрии [6] известно, что любые две одинаково ориентированные тройки координатных осей можно совместить двумя поворотами, при каждом из которых остается неизменной одна координатная ось [8].

При повороте на угол ψ относительно оси Y, на угол φ вокруг оси X и последующего проектирования на плоскость $Z = 0$ возникает матрица:

$$[M] = [R_y] \cdot [R_x] \cdot [P_z] = \begin{bmatrix} \cos\psi & \sin\varphi \cdot \cos\psi & 0 & 0 \\ 0 & \cos\varphi & 0 & 0 \\ \sin\psi & -\sin\varphi \cdot \cos\psi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

При этом преобразуются и единичные орты координатных осей X, Y, Z:

$$(1\ 0\ 0\ 1) \cdot [M] = (\cos\psi\ \ \sin\varphi \cdot \sin\psi\ 0\ 1);$$
$$(0\ 1\ 0\ 1) \cdot [M] = (0\ \ \cos\varphi\ 0\ 1);$$
$$(0\ 0\ 1\ 1) \cdot [M] = (\sin\psi\ \ -\sin\varphi \cdot \cos\psi\ 0\ 1).$$

При триметрии полученные в результате длины проекций различны. Диметрия характеризуется тем, что длины двух проекций совпадают:

$$\cos^2\psi + \sin^2\varphi \cdot \sin^2\psi = \cos^2\varphi.$$

В случае изометрии:

$$\sin^2\psi + \sin^2\varphi \cdot \cos^2\psi = \cos^2\varphi.$$

Из последних двух соотношений легко получаем, что:

$$\sin^2\psi = \frac{1}{2}, \quad \sin^2\varphi = \frac{1}{3}.$$

Косоугольной проекцией *называется такая проекция, у которой проектирующие прямые образуют с плоскостью проекции угол, отличный от 90°.*

При косоугольном проектировании орта оси Z на плоскость XOY имеем:

$$(0\ 0\ 1\ 1) \mapsto (\alpha\ \beta\ 0\ 1).$$

Матрица соответствующего преобразования имеет вид:

$$[K_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \alpha & \beta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Различают два типа косоугольных проекций:

1) Свободная проекция — угол между проектирующими прямыми и плоскостью проекции равен 45° (рис. 4.6). Из этого следует:

$$\alpha = \beta = \cos\frac{\pi}{4}.$$

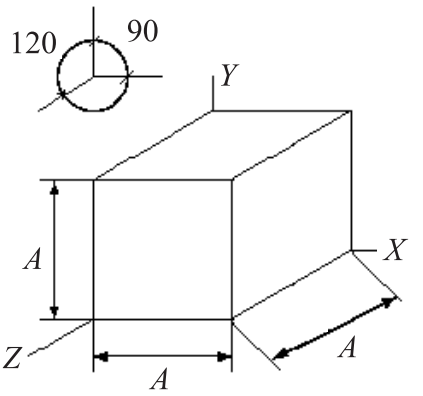


Рис. 4.6 – Свободная проекция

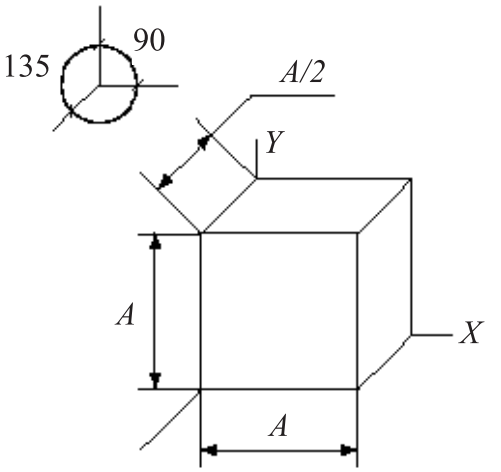


Рис. 4.7 – Кабинетная проекция

2) Кабинетная проекция — частный случай свободной проекции, в котором масштаб по третьей оси вдвое меньше, чем по двум первым (рис. 4.7):

$$\alpha = \beta = \frac{1}{2}\cos\frac{\pi}{4}.$$

4.2.2 Перспективное проектирование

Перспективная (центральная) проекция *— вид проекции, где лучи проектирования исходят из одного центра (центра проектирования), размещенного на конечном расстоянии от объектов и плоскости проектирования.*

Данный вид проектирования позволяет получить наиболее реалистичные изображения трехмерных объектов из-за перспективных искажений сцены.

Перспективная проекция получается путем перспективного преобразования и проецирования на некоторую плоскость наблюдения. Перспективные проекции параллельных прямых, не параллельных плоскости проекции, будут сходиться в точке схода. Разделяют три вида перспективного проектирования:

1) Одноточечное перспективное преобразование характеризуется одной точкой схода (рис. 4.8).

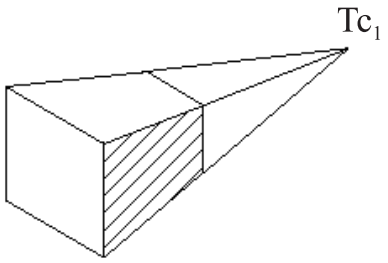


Рис. 4.8 – Одноточечное перспективное преобразование

Перспективная проекция на плоскость $z = 0$ обеспечивается преобразованием:

$$[E_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Отсюда:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot [E_z] = \begin{bmatrix} \dfrac{x}{r \cdot z + 1} & \dfrac{y}{r \cdot z + 1} & 0 & 1 \end{bmatrix}.$$

Возьмем точку в бесконечности на оси z и выполним перспективное преобразование:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \cdot [E_z] = \begin{bmatrix} 0 & 0 & 1 & r \end{bmatrix}.$$

Поскольку параллельные линии исходного пространства «сходятся» в бесконечности, то в преобразованном пространстве линии, которые были параллельны оси z, будут сходиться в точке схода:

$$\begin{bmatrix} 0 & 0 & \frac{1}{r} & 1 \end{bmatrix}.$$

2) Двухточечное перспективное преобразование характеризуется двумя точками схода (рис. 4.9).

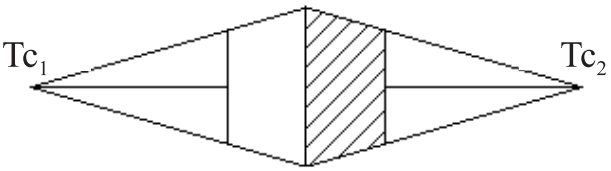


Рис. 4.9 – Двухточечное перспективное преобразование

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{p \cdot x + q \cdot y + 1} & \frac{y}{p \cdot x + q \cdot y + 1} & \frac{z}{p \cdot x + q \cdot y + 1} & 1 \end{bmatrix}$$

Точки схода в данном случае расположены:

- на оси x: $\begin{bmatrix} \frac{1}{p} & 0 & 0 & 1 \end{bmatrix}$;
- на оси y: $\begin{bmatrix} 0 & \frac{1}{q} & 0 & 1 \end{bmatrix}$.

3) Трехточечное перспективное преобразование характеризуется тремя точками схода (рис. 4.10).

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{x}{p \cdot x + q \cdot y + r \cdot z + 1} & \frac{y}{p \cdot x + q \cdot y + r \cdot z + 1} & \frac{z}{p \cdot x + q \cdot y + r \cdot z + 1} & 1 \end{bmatrix}$$

Точки схода в данном случае расположены:

- на оси x: $\begin{bmatrix} \frac{1}{p} & 0 & 0 & 1 \end{bmatrix}$;
- на оси y: $\begin{bmatrix} 0 & \frac{1}{q} & 0 & 1 \end{bmatrix}$;
- на оси z: $\begin{bmatrix} 0 & 0 & \frac{1}{z} & 1 \end{bmatrix}$.

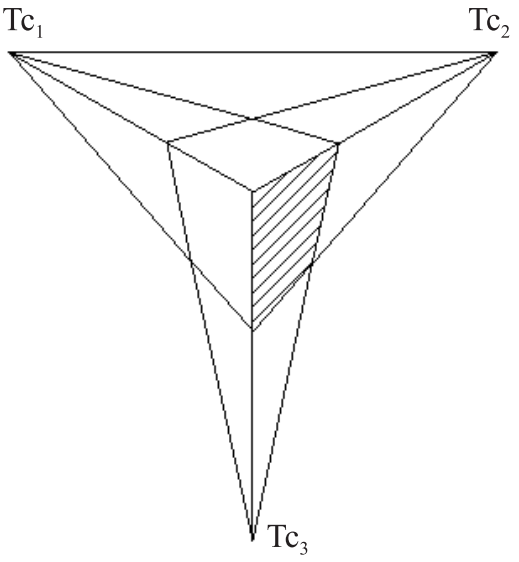


Рис. 4.10 – Трехточечное перспективное преобразование

4.3 Удаление невидимых линий и поверхностей

После того как вершины прошли через все этапы геометрических преобразований и процедуру отсечения, «на конвейере» остались только те геометрические объекты, которые потенциально могут попасть в формируемое изображение. Но прежде чем приступать к их растровому преобразованию, нужно решить еще одну задачу — удалить объекты, перекрываемые другими объектами с точки зрения наблюдателя.

Каркасные изображения легко получать, но они мало похожи на реальные объекты. Большинство объектов непрозрачны, и составляющие их поверхности закрывают собой другие тела. Неважно, откуда мы смотрим на них, в любом случае некоторые части тел от нас скрыты. 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 физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные, вообще говоря, лишь точностью вычислений. Полученные изображения можно свободно увеличивать во много раз.

Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность.

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

В программах КГ применяются несколько методов удаления невидимых поверхностей.

4.3.1 Удаление нелицевых граней

Если сцена состоит только из выпуклых многогранников, то внутренние грани многоугольников никогда не будут «показываться на глаза» наблюдателю. Но у такого выпуклого многогранника часть граней будет повернута от наблюдателя (так называемые нелицевые грани) и, следовательно, тоже не будет видна наблюдателю.

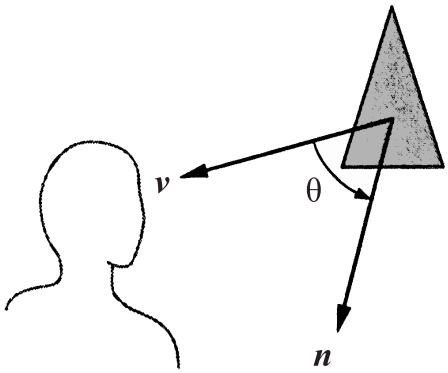


Рис. 4.11 – Анализ лицевой грани

В конечном результате, нелицевые грани будут перекрыты другими, и это сможет выявить универсальный алгоритм удаления невидимых поверхностей, но можно упростить работу, сразу же отбраковав нелицевые грани. Принцип отбраковки нелицевых граней поясняется на рис. 4.11. Внешняя сторона многоугольника будет видима наблюдателю только в том случае, если нормаль к плоскости многоугольника будет направлена к наблюдателю.

Обозначим через θ угол между внешней нормалью плоскости многоугольника и направлением на наблюдателя, тогда многоугольник будет представлять лицевую грань объекта только в том случае, если:

$$-90^\circ \leqslant \theta \leqslant 90^\circ.$$

Или другой вид записи:

$$\cos\theta \geqslant 0.$$

Последнее условие анализируется проще, поскольку знак косинуса легко определяется по скалярному произведению и условие можно сформулировать следующим образом:

$$\boldsymbol{n} \cdot \boldsymbol{v} \geqslant 0.$$

Можно еще упростить этот анализ, если учесть, что обычно он выполняется после преобразования в нормализованную систему координат устройства отображения. В этой системе координат все виды являются ортогональными, следовательно, направление наблюдателя задается координатной осью z. Таким образом, в однородных координатах вектор $\boldsymbol{v}$ имеет вид:

$$\boldsymbol{v} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

Если уравнение плоскости, в которой лежит многоугольник, имеет вид:

$$a \cdot x + b \cdot y + c \cdot z + d = 0,$$

то для определения, является ли грань лицевой или нелицевой, потребуется проверить только знак коэффициента c.

4.3.2 Алгоритм Z-буфера

Одним из самых распространенных алгоритмов удаления невидимых поверхностей является алгоритм Z-буфера (буфер глубины). Алгоритм Z-буфера позволяет определить, какие пиксели граней сцены видимы, а какие заслонены гранями других объектов.

Буфер глубины *(Z-buffer, depth buffer) — дополнительный объем памяти, где хранится значение глубины примитивов передней поверхности (расстояние от наблюдателя до поверхности изображаемого объекта) для каждого пикселя.*

Размеры Z-буфера равны размерам окна, таким образом, каждому пикселю окна соответствует ячейка Z-буфера. В этой ячейке хранится значение глубины пикселя (рис. 4.12). Перед растеризацией сцены Z-буфер заполняется значением, соответствующим максимальной глубине. В случае, когда глубина характеризуется значением w, максимальной глубине соответствует нулевое значение. Анализ видимости происходит при растеризации граней, для каждого пикселя рассчитывается глубина и сравнивается со значением в Z-буфере, если рисуемый пиксель ближе (его w больше значения в Z-буфере), то пиксель рисуется, а значение в Z-буфере заменяется его глубиной. Если пиксель дальше, то пиксель не рисуется и Z-буфер не изменяется, текущий пиксель дальше того, что нарисован ранее, а значит, невидим.

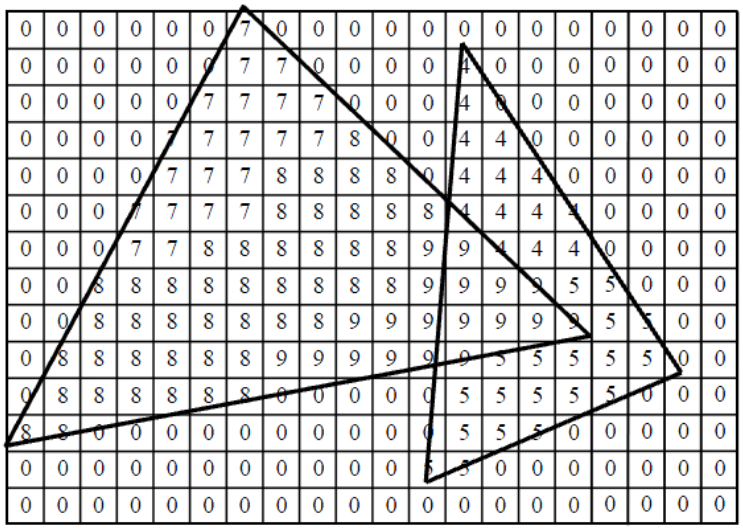


Рис. 4.12 – Организация Z-буфера

Алгоритм довольно просто реализуется как программно, так и аппаратно, прекрасно сочетается с конвейерной архитектурой графической системы и может выполняться со скоростью, соответствующей скорости обработки вершин остальными модулями конвейера. Хотя алгоритм отталкивается от пространства изображения, он включает цикл просмотра многоугольников, а не пикселей и может быть реализован в процессе растрового преобразования.

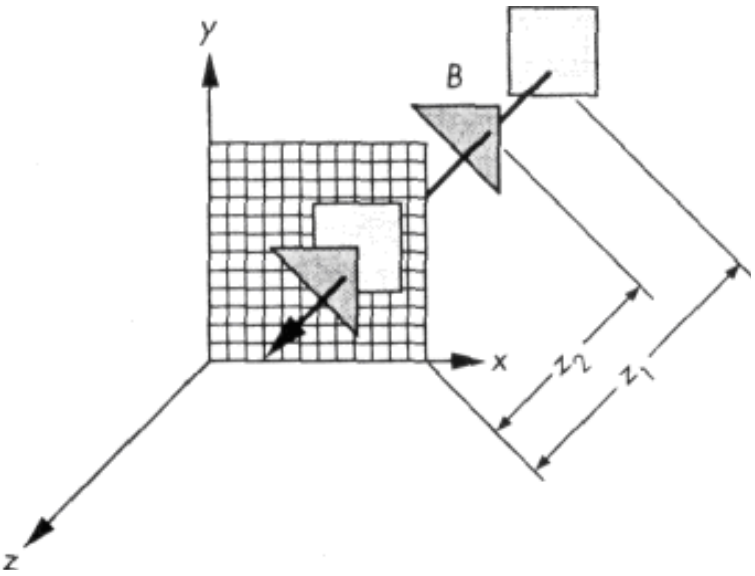


Рис. 4.13 – Алгоритм *Z*-буфера

Предположим, что выполняется растровое преобразование одного из двух многоугольников, показанных на рис. 4.13.

Используя принятую в графической системе модель закрашивания, можно вычислить цвет точек пересечения с каждым из многоугольников луча, исходящего из центра проецирования и проходящего через заданный пиксель картинной плоскости. Кроме того, одновременно нужно определить, является ли данная точка пересечения видимой наблюдателю, из двух анализируемых видимой будет только точка, ближайшая к наблюдателю. Следовательно, если преобразуется многоугольник *B*, его образ должен появиться на экране только в случае, если расстояние z_2 меньше расстояния z_1 до многоугольника *A*. И наоборот, если преобразуется многоугольник *A*, то его цвет не должен никак повлиять на формируемый цвет пикселя и изображение этого многоугольника в этой области экрана не должно появиться. Но есть небольшая загвоздка в реализации этой простой идеи — обработка многоугольников выполняется последовательно, а потому, преобразуя в растр один многоугольник, мы не располагаем информацией о том, как по отношению к нему расположены другие. Решается эта проблема с помощью промежуточного буфера, в который записывается информация о глубине размещения каждого многоугольника.

Предположим, что в нашем распоряжении имеется буфер — назовем его *Z*-буфером, который имеет такую же организацию, как и буфер кадра, а разрядность каждой ячейки достаточна для хранения информации о глубине с приемлемой для качества изображения точностью. Например, если формат изображения

1024×1280 пикселей и расстояние в графической системе представляется действительным числом с однократной точностью, то в Z-буфере должно быть 1024×1280 32-разрядных ячеек. Перед началом растрового преобразования объектов сцены в каждую ячейку заносится код, соответствующий максимальному расстоянию от центра проецирования. Буфер кадра в исходном состоянии заполняется кодами цвета фона. В процессе растрового преобразования объектов каждая ячейка Z-буфера содержит значение расстояния до ближайшего из обработанных ранее многоугольников вдоль проецирующего луча, проходящего через соответствующий пиксель пространства изображения.

Процесс заполнения Z-буфера выглядит в первом приближении следующим образом. Все многоугольники в описании сцены последовательно, один за другим, подвергаются растровому преобразованию. Для каждой точки на многоугольнике, которая проецируется на определенный пиксель, вычисляется расстояние до центра проецирования. Это расстояние сравнивается с уже хранящимся в той ячейке Z-буфера, которая соответствует формируемому пикселю. Если расстояние до текущего многоугольника больше хранящегося в Z-буфере, значит, ранее был обработан многоугольник, расположенный ближе к наблюдателю, который, таким образом, загораживает текущий. Следовательно, данная точка текущего многоугольника не будет видима и информацию о ее цвете не нужно заносить в буфер кадра. Если же вычисленное расстояние меньше того, что хранится в Z-буфере, значит, текущий многоугольник сам загораживает те, что были обработаны ранее, и его код цвета должен заменить в буфере кадра код цвета, сформированный ранее. Одновременно вычисленное только что расстояние заносится в соответствующую ячейку Z-буфера.

4.3.3 Алгоритм Робертса

Самым первым алгоритмом, предназначенным для удаления невидимых линий, был алгоритм Робертса. Сначала в нем отбрасываются все ребра, обе определяющие грани которых являются нелицевыми. Следующим шагом является проверка оставшихся ребер со всеми гранями многогранника на закрывание.

Возможны следующие случаи:

1) грань не закрывает ребро;
2) грань полностью закрывает ребро;
3) грань частично закрывает ребро (в этом случае ребро разбивается на несколько частей, из которых видимыми являются не более двух).

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это математический метод, работающий в объектном пространстве. Алгоритм, прежде всего, удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически как квадрат числа объектов. Однако математические методы, используемые в этом алгоритме, просты, мощны и точны.

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

Алгоритм Робертса делится на три этапа:

- На первом этапе каждое тело анализируется индивидуально с целью удаления нелицевых плоскостей.
- На втором этапе проверяется экранирование оставшихся в каждом теле ребер всеми другими телами с целью обнаружения их невидимых отрезков.
- На третьем этапе вычисляются отрезки, которые образуют новые ребра при протыкании телами друг друга.

В данном алгоритме предполагается, что тела состоят из плоских полигональных граней, которые, в свою очередь, состоят из рёбер, а ребра — из отдельных вершин. Все вершины, ребра и грани связаны с конкретным телом.

4.3.4 Алгоритм построчного сканирования

В 60-х годах метод построчного сканирования считался наиболее эффективным алгоритмом удаления невидимых поверхностей. Алгоритм построчного сканирования сочетает удаление невидимых поверхностей с растровым преобразованием.

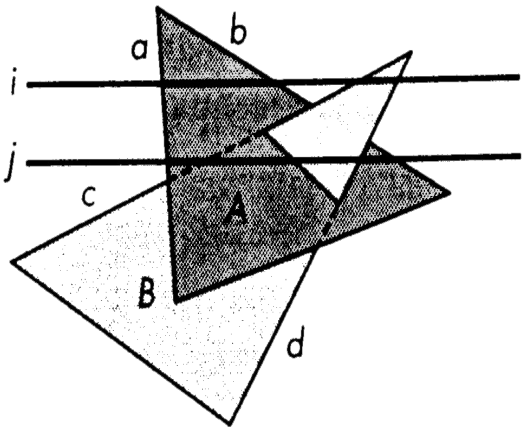


Рис. 4.14 – Метод построчного сканирования

Идея алгоритма построчного сканирования представлена на рис. 4.14. На рисунке отображены два многоугольника, пересекающихся в пространстве. При растровом преобразовании многоугольников, которое выполняется последовательно, строка за строкой, можно воспользоваться методом вычисления глубины в приращениях, рассмотренным ранее. Однако существует еще более эффективный способ обработки.

Перемещаясь вдоль строки растра *i*, мы пересекаем ребро *a* многоугольника *A*. Поскольку это первый многоугольник, который пересекается строкой, нет смысла выполнять какие-либо вычисления, анализирующие его глубину. Начиная с этого пикселя, последующим присваивается код цвета, соответствующий окраске многоугольника *A*. Но когда по мере перемещения по строке мы встретимся со следующим ребром этого многоугольника, *b*, то это будет означать, что с обработкой

многоугольника *A* на этой строке покончено и можно присваивать последующим пикселям код цвета фона. Так будет продолжаться до тех пор, пока не встретится ребро *c* многоугольника *B*; поскольку перед этим на строке отображался фон, можно опять не принимать во внимание информацию о глубине и считать дальнейший участок строки принадлежащим образу многоугольника *B*.

Более сложная ситуация возникает на строке *j*. Сначала мы опять обнаруживаем ребро *a* и, не прибегая к анализу глубины, можем, начиная с этого пикселя, присваивать последующим код цвета, соответствующий окраске многоугольника *A*. Но далее встречается ребро *c* «конкурирующего» многоугольника, и здесь без анализа глубины не обойтись. Этот анализ придется проводить для всех пикселей строки *j*, пока не встретится ребро *d*.

Хотя на первый взгляд кажется, что этот алгоритм во многом напоминает алгоритм *Z*-буфера, между ними есть одно принципиальное отличие — алгоритм построчного сканирования при обработке каждого пикселя должен просматривать все многоугольники, претендующие на отображение в данной зоне экрана, а алгоритм *Z*-буфера в каждый момент времени имеет дело только с одним многоугольником.

Реализация алгоритма построчного сканирования требует тщательно продуманной организации данных о многоугольниках, которая позволит быстро отбирать среди них те, которые претендуют на отображение в текущей строке. Чаще всего структура данных базируется на массиве указателей, по одному на каждую строку, каждый из которых указывает на специальную структуру описания ребер многогранников, пересекаемых этой строкой.

4.4 Закрашивание поверхностей

В этом разделе будут рассмотрены методы, которые позволяют получить наиболее реалистичные изображения для объектов, моделируемых многогранниками и полигональными сетками.

4.4.1 Модели отражения света

Рассмотрим, как можно определить цвет пикселей изображения поверхности согласно интенсивности отраженного света при учете взаимного расположения поверхности, источника света и наблюдателя.

Зеркальное отражение *— отражение, при котором угол между нормалью и падающим лучом* (θ) *равен углу между нормалью и отраженным лучом.*

Падающий луч, отраженный, и нормаль располагаются в одной плоскости (рис. 4.15).

Поверхность считается *идеально зеркальной*, если на ней отсутствуют какие-либо неровности, шероховатости. Собственный цвет у такой поверхности не наблюдается. Световая энергия падающего луча отражается только по линии отраженного луча. Какое-либо рассеяние в стороны от этой линии отсутствует. В природе нет идеально гладких поверхностей, поэтому полагают, что если глубина ше-

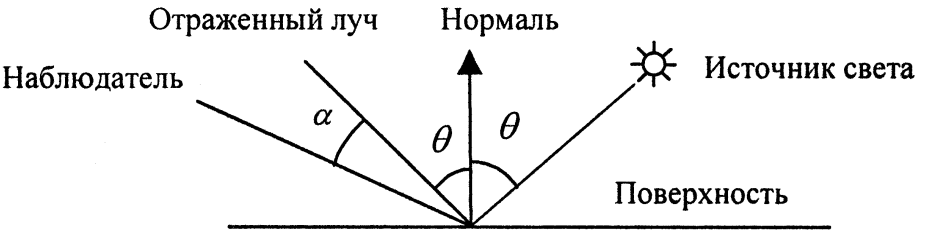


Рис. 4.15 – Зеркальное отражение света

роховатостей существенно меньше длины волны излучения, то рассеивания не наблюдается. Для видимого спектра можно принять, что глубина шероховатостей поверхности зеркала должна быть существенно меньше 0.5 мкм.

Если поверхность зеркала отполирована неидеально, то наблюдается зависимость интенсивности отраженного света от длины волны — чем больше длина волны, тем лучше отражение. Например, красные лучи отражаются сильнее, чем синие.

При наличии шероховатостей имеется зависимость интенсивности отраженного света от угла падения. Отражение света максимально для углов θ, близких к 90 градусам.

Падающий луч, попадая на слегка шероховатую поверхность реального зеркала, порождает не один отраженный луч, а несколько лучей, рассеиваемых по различным направлениям. Зона рассеивания зависит от качества полировки и может быть описана некоторым законом распределения. Как правило, форма зоны рассеивания симметрична относительно линии идеального зеркально отраженного луча. К числу простейших относится эмпирическая модель распределения Фонга:

$$I_S = I \cdot K_S \cdot \cos^p \alpha,$$

где I — интенсивность излучения источника, K_S — коэффициент пропорциональности, α — угол отклонения от линии идеально отраженного луча, p — показатель из диапазона от 1 до 200, зависящий от качества полировки.

Диффузное отражение *— отражение, при котором падающий на поверхность луч рассеивается одинаково по всем направлениям.*

Диффузное отражение присуще *матовым* поверхностям. Матовой можно считать такую поверхность, размер шероховатостей которой уже настолько велик, что падающий луч рассеивается равномерно во все стороны. Такой тип отражения характерен, например, для гипса, песка, бумаги.

Диффузное отражение описывается законом Ламберта, согласно которому интенсивность отраженного света пропорциональна косинусу угла между направлением на точечный источник света и нормалью к поверхности (рис. 4.16).

$$I_d = I \cdot K_d \cdot \cos \theta,$$

где I — интенсивность источника света, K_d — коэффициент, который учитывает свойства материала поверхности.

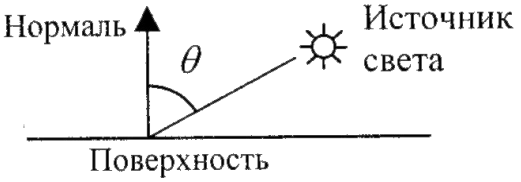


Рис. 4.16 – Матовая поверхность

Значение K_d находится в диапазоне от 0 до 1. Интенсивность отраженного света не зависит от расположения наблюдателя.

Матовая поверхность имеет свой цвет. Наблюдаемый цвет матовой поверхности определяется комбинацией собственного цвета поверхности и цвета излучения источника света.

При создании реалистичных изображений следует учитывать то, что в природе не существует идеально зеркальных или полностью матовых поверхностей. При изображении объектов средствами компьютерной графики обычно моделируют сочетание зеркальности и диффузного рассеивания в пропорции, характерной для конкретного материала. В этом случае модель отражения записывают в виде суммы диффузной и зеркальной компонент:

$$I_{\text{отр}} = I(K_d \cdot \cos\theta + K_S \cdot \cos^p\alpha),$$

где константы K_d, K_S определяют отражательные свойства материала.

Согласно этой формуле интенсивность отраженного света равна нулю для некоторых углов θ и α. Однако в реальных сценах обычно нет полностью затемненных объектов, следует учитывать фоновую подсветку, освещение рассеянным светом, отраженным от других объектов. В таком случае интенсивность может быть эмпирически выражена следующей формулой:

$$I_{\text{отр}} = I_a \cdot K_a + I(K_d \cdot \cos\theta + K_S \cdot \cos^p\alpha),$$

где I_a — интенсивность рассеянного света, K_a — константа.

4.4.2 Вычисление нормалей

Определение вектора нормали к поверхности в заданной точке может быть выполнено различными способами. В значительной степени это определяется типом модели описания поверхности. Для поверхностей, заданных в аналитической форме, известны методы дифференциальной геометрии, которые основываются на вычислении частных производных функций описания [13]. Например, если поверхность задана параметрическими функциями:

$$x = x(s,t),$$
$$y = y(s,t),$$
$$z = z(s,t),$$

то координаты вектора нормали можно вычислить так:

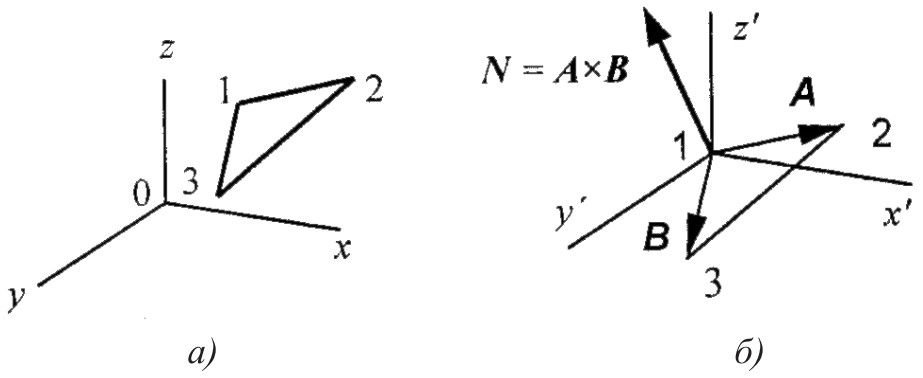
$$x_N = \begin{Vmatrix} \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial y}{\partial t} & \frac{\partial z}{\partial s} \end{Vmatrix} = \frac{\partial y}{\partial s} \cdot \frac{\partial z}{\partial t} - \frac{\partial y}{\partial t} \cdot \frac{\partial z}{\partial s},$$

$$y_N = \begin{Vmatrix} \frac{\partial z}{\partial s} & \frac{\partial x}{\partial s} \\ \frac{\partial z}{\partial t} & \frac{\partial x}{\partial t} \end{Vmatrix} = \frac{\partial z}{\partial s} \cdot \frac{\partial x}{\partial t} - \frac{\partial z}{\partial t} \cdot \frac{\partial x}{\partial s},$$

$$z_N = \begin{Vmatrix} \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} \end{Vmatrix} = \frac{\partial x}{\partial s} \cdot \frac{\partial y}{\partial t} - \frac{\partial x}{\partial t} \cdot \frac{\partial y}{\partial s}.$$

В случае описания поверхности векторно-полигональной моделью для определения нормалей можно использовать методы векторной алгебры.

Пусть в пространстве задана некоторая многогранная поверхность. Рассмотрим одну ее плоскую грань в виде треугольника (рис. 4.17, *а*). Для вычисления координат вектора нормали воспользуемся векторным произведением любых двух векторов, лежащих в плоскости грани. В качестве таких векторов могут служить и ребра грани, например ребра 1–2 и 1–3. Однако формулы для векторного произведения были определены только для радиус-векторов.



а) *б)*

Рис. 4.17 – Пример плоской грани поверхности: *а* — треугольная грань, *б* — радиус-векторы

Чтобы перейти к радиус-векторам, введем новую систему координат, центр которой совпадает с вершиной 1, а оси параллельны осям прежней системы. Координаты вершин в новой системе:

$$x'_i = x_i - x_1,$$
$$y'_i = y_i - y_1,$$
$$z'_i = z_i - z_1.$$

Теперь назовем ребро (1–2) вектором $\boldsymbol{A}$, а ребро (1–3) — вектором $\boldsymbol{B}$, как показано на рис. 4.17, *б*. Таким образом, положение нормали к грани в пространстве

будет описываться радиус-вектором $\boldsymbol{N}$. Его координаты в системе (x', y', z') выразим формулами для векторного произведения:

$$x'_N = (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1),$$
$$y'_N = (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1),$$
$$z'_N = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1).$$

Здесь использованы координаты вершин грани до переноса.

Плоская грань может изображаться в различных ракурсах. В каждой конкретной ситуации необходимо выбирать направление нормали, соответствующее *видимой стороне* грани. Если плоская грань может быть видна с обратной стороны, то тогда в расчетах отраженного света необходимо выбирать в качестве нормали обратный вектор, то есть $(-N)$.

Если полигональная поверхность имеет не треугольные грани, а, например, плоские четырехугольные, то расчет нормали можно выполнять по любым трем вершинам грани.

4.4.3 Метод Гуро

Этот метод предназначен для создания иллюзии гладкой криволинейной поверхности, описанной в виде многогранников или полигональной сетки с плоскими гранями. Если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник, зрение человека имеет способность подчеркивать перепады яркости на границах смежных граней — такой эффект называется эффектом полос Маха. Поэтому для создания иллюзии гладкости нужно намного увеличить количество граней, что приводит к существенному замедлению визуализации [13].

Метод Гуро основывается на идее закрашивания каждой плоской грани не одним цветом, а плавно изменяющимися оттенками, вычисляемыми путем интерполяции цветов примыкающих граней. Закрашивание граней по методу Гуро осуществляется в четыре этапа.

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

4.4.4 Метод Фонга

Аналогичен методу Гуро, но при использовании метода Фонга для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей. Закрашивание граней по методу Фонга осуществляется в три этапа:

1) Определяются нормали к граням.
2) По нормалям к граням определяются нормали в вершинах.
3) В каждой точке закрашиваемой грани определяется интерполированный вектор нормали.

Метод Фонга сложнее, чем метод Гуро. Для каждой точки (пикселя) поверхности необходимо выполнять намного больше вычислительных операций. Тем не менее он дает значительно лучшие результаты, в особенности при имитации зеркальных поверхностей [13].

4.4.5 Преломление света

Законы преломления света следует учитывать при построении изображений прозрачных объектов.

Согласно этой модели идеального преломления луч отклоняется на границе двух сред. Причем падающий луч, преломленный луч и нормаль лежат в одной плоскости (в этой же плоскости лежит и зеркально отраженный луч). Обозначим угол между падающим лучом и нормалью как α_1, а угол между нормалью и преломленным лучом как α_2. Для этих углов известен закон Снеллиуса, согласно которому:

$$n_1 \cdot \sin\alpha_1 = n_2 \cdot \sin\alpha_2,$$

где n_1 и n_2 — абсолютные показатели преломления соответствующих сред.

На рис. 4.18 изображен пример отклонения луча при преломлении. В данном случае границами раздела сред являются две параллельные плоскости, например при прохождении луча через толстое стекло. Очевидно, что угол α_1 равен углу α_4, а угол α_2 равен углу α_3. Иными словами, после прохождения сквозь стекло луч параллельно смещается. Это смещение зависит от толщины стекла и соотношения показателей преломления сред.

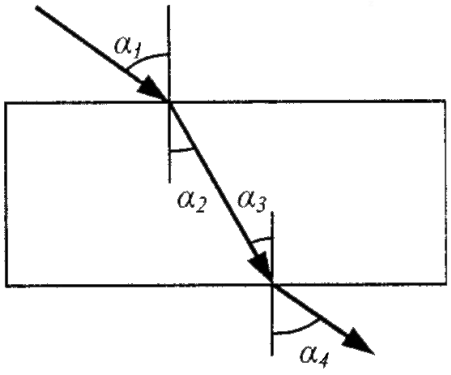


Рис. 4.18 – Преломление луча

Принято считать, что для вакуума абсолютный показатель преломления равен единице. Для воздуха он составляет 1.00029, для воды — 1.33, для стекла разных сортов: 1.52 (легкий крон), 1.65 (тяжелый крон). Показатель преломления зависит от состояния вещества, например от температуры. На практике обычно используют отношение показателей преломления двух сред (n_1/n_2), называемое относительным показателем преломления.

Еще одним важным аспектом преломления является зависимость отклонения луча от длины волны. Чем меньше длина волны, тем больше отклоняется луч при преломлении. Благодаря этому свойству преломления мы и наблюдаем радугу. Фиолетовый (1–0.4 мкм) луч отклоняется больше всего, а красный ($\lambda = 0.7$ мкм) — меньше всего. Например, для стекла показатель преломления в видимом спектре изменяется от 1.53 до 1.51.

Таким образом, каждый прозрачный материал описывается показателем преломления, зависящим от длины волны. Кроме того, необходимо учитывать, какая часть световой энергии отражается, а какая часть проходит через объект и описывается преломлением света.

4.4.6 Вычисление вектора преломленного луча

Сформулируем задачу следующим образом. Заданы два единичных вектора: S_i — радиус-вектор, направленный на источник, и N_i — радиус-вектор нормали к границе раздела двух сред. Также должны быть известны два коэффициента преломления для данных сред — n_1 и n_2 (или же их отношение).

Требуется найти единичный радиус-вектор преломленного луча T_1. Для решения этой задачи выполним некоторые геометрические построения (рис. 4.19). Нанесем еще несколько радиус-векторов (далее их для краткости будем называть векторами). Искомый вектор T_1 равен сумме двух векторов:

$$T_1 = N_T + B.$$

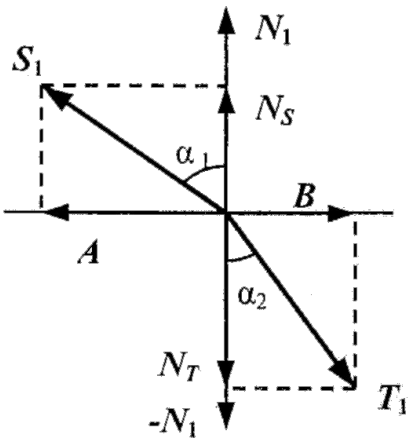


Рис. 4.19 – Преломление

Найдем вначале вектор N_T. Он противоположен по направлению вектору нормали, а поскольку по условию задачи T_1 — единичный, то его длина равна:

$$|T_1| \cdot \cos\alpha_2 = \cos\alpha_2.$$

Откуда:

$$N_T = -N_1 \cdot \cos\alpha_2.$$

Необходимо определить $\cos\alpha_2$. Запишем закон преломления в виде:

$$\sin\alpha_2 = n \cdot \sin\alpha_1,$$

где $n = n_1/n_2$.

Применив известные, тождества получим:

$$\cos\alpha_2 = \sqrt{1 + n^2(\cos^2\alpha_1 - 1)}.$$

Значение $\cos\alpha_1$ можно выразить через скалярное произведение единичных векторов S_1 и N_1:

$$\cos\alpha_1 = S_1 \cdot N_1.$$

Тогда мы можем записать такое выражение для вектора N_T:

$$N_T = -N_1\sqrt{1 + n^2\left((S_1 \cdot N_1)^2 - 1\right)}.$$

Осталось найти выражение для вектора $\boldsymbol{B}$. Он располагается на одной прямой с вектором $\boldsymbol{A}$, причем $A = S_1 - N_S$. Учитывая, что вектор $N_S = N_1 \cdot \cos\alpha_1$, то $A = S_1 - N_1 \cdot \cos\alpha_1$. Отсюда:

$$\frac{|B|}{|F|} = \frac{\sin\alpha_2}{\sin\alpha_1} = \frac{n_2}{n_1} = n.$$

Учитывая взаимное расположение векторов $\boldsymbol{A}$ и $\boldsymbol{B}$, получим:

$$B = -n \cdot F = n\left(N_1(S_1 \cdot N_1) - S_1\right).$$

Теперь можно записать искомое выражение для единичного радиус-вектора луча преломления T_1:

$$T_1 = n \cdot N_1(S_1 \cdot N_1) - nS_1 - N_1\sqrt{1 + n^2\left((S_1 \cdot N_1)^2 - 1\right)}.$$

Если подкоренное выражение отрицательно, то преломленный луч не существует. Это соответствует так называемому полному внутреннему отражению.

4.4.7 Трассировка лучей

Методы трассировки лучей на сегодняшний день считаются наиболее мощными и универсальными методами создания реалистичных изображений. Известно много примеров реализации алгоритмов трассировки для качественного отображения самых сложных трехмерных сцен. Можно отметить, что универсальность методов трассировки в значительной степени обусловлена тем, что в их основе лежат простые и ясные понятия, отражающие наш опыт восприятия окружающего мира.

Как мы видим окружающую нас реальность? Во-первых, нужно определиться с тем, что мы вообще способны видеть. Это изучается в специальных дисциплинах, а в некоторой степени, это вопрос философский. Но здесь мы будем полагать, что окружающие нас объекты обладают по отношению к свету такими свойствами:

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

Каждое из этих свойств можно описать некоторым набором характеристик. Например, излучение можно охарактеризовать интенсивностью, направленностью, спектром. Излучение может исходить от условно точечного источника (далекая звезда) или протяженного (скажем, от извергающейся из кратера вулкана расплавленной лавы). Распространение излучения может осуществляться вдоль достаточно узкого луча (сфокусированный луч лазера), конусом (прожектор), равномерно во все стороны (Солнце) либо еще как-нибудь. Свойство отражения (поглощения) можно описать характеристиками диффузного рассеивания и зеркального отражения. Прозрачность можно описать ослаблением интенсивности и преломлением.

Распределение световой энергии по возможным направлениям световых лучей можно отобразить с помощью векторных диаграмм, в которых длина векторов соответствует интенсивности

Рассмотрим то, как формируется изображение некоторой сцены, включающей в себя несколько пространственных объектов. Будем полагать, что из точек поверхности (объема) излучающих объектов исходят лучи света. Можно назвать такие лучи первичными — они освещают все остальное. Важным моментом является предположение, что световой луч в свободном пространстве распространяется вдоль прямой линии (хотя в специальных разделах физики изучаются также и причины возможного искривления). Но в геометрической оптике полагают, что луч света распространяется прямолинейно до тех пор, пока не встретится отражающая поверхность или граница среды преломления.

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

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

Цвет отдельных точек изображения определяется спектром и интенсивностью первичных лучей источников излучения, а также поглощением световой энергии в объектах, встретившихся на пути соответствующих лучей.

Трассировка лучей *— метод реалистической визуализации, моделирующий движение светового луча в изображаемой сцене; яркость точки экрана (пикселя) определяется интенсивностью проходящего через нее луча.*

Непосредственная реализация данной лучевой модели формирования изображения представляется затруднительной. Можно попробовать построить алгоритм построения изображения указанным способом. В таком алгоритме необходимо предусмотреть перебор всех первичных лучей и определить те из них, которые попадают в объекты и в камеру. Затем выполнить перебор всех вторичных лучей, и также учесть только те, которые попадают в объекты и в камеру. И так далее. Можно назвать такой метод прямой трассировкой лучей. Практическая ценность такого метода вызывает сомнения. Очевидно, что полный перебор бесконечного числа лучей в принципе невозможен. Даже если каким-то образом свести это к конечному числу операций (например, разделить всю сферу направлений на угловые секторы и оперировать уже не бесконечно тонкими линиями, а секторами), все равно остается главный недостаток метода — много лишних операций, связанных с расчетом лучей, которые затем не используются.

Метод обратной трассировки лучей позволяет значительно сократить перебор световых лучей. Метод разработан в 80-х годах, основополагающими считаются работы Уиттеда и Кэя. Согласно этому методу отслеживание лучей производится не от источников света, а в обратном направлении — от точки наблюдения. Так учитываются только те лучи, которые вносят вклад в формирование изображения.

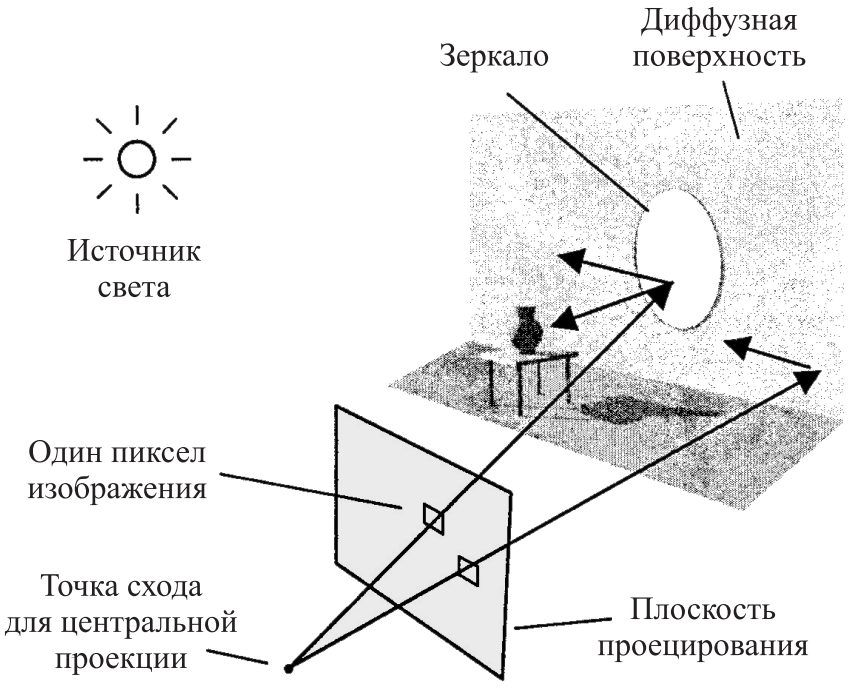


Рис. 4.20 – Схема обратной трассировки

Рассмотрим, как можно получить растровое изображение некоторой трехмерной сцены методом обратной трассировки. Предположим, что плоскость проецирования разбита на множество пикселей. Выберем центральную проекцию с центром схода на некотором расстоянии от плоскости проецирования. Проведем прямую линию из центра схода через середину пикселя плоскости проецирования (рис. 4.20).

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

При практической реализации метода обратной трассировки вводят ограничения. Некоторые их них необходимы, чтобы можно было в принципе решить задачу синтеза изображения, а некоторые ограничения позволяют значительно повысить быстродействие трассировки.

Рассмотрим примеры таких ограничений.

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

4.5 Примеры изображения трехмерных объектов

Рассмотрим примеры каркасного изображения нескольких геометрических объектов.

Шар

Для каркасного изображения шара можно отобразить сетку меридианов и параллелей.

Для этого удобно воспользоваться известными формулами параметрического описания. Координаты точек поверхности шара определяются как функции от двух переменных (параметров) — широты и долготы (рис. 4.21):

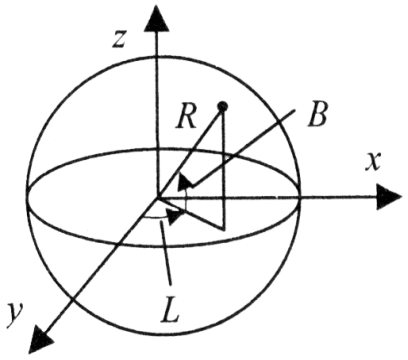


Рис. 4.21 – Широта и долгота

$$x = R \cdot \cos B \cdot \sin L,$$
$$y = R \cdot \cos B \cdot \cos L,$$
$$z = R \sin B,$$

где R — радиус шара, B — широта (−90° до +90°), L — долгота (от −180° до +180°).

Цилиндр

Используем формулы параметрического описания поверхности цилиндра. В одной из возможных разновидностей такого описания применяются следующие параметры — долгота (l) и высота (h).

$$x = R \cdot \sin l,$$
$$y = R \cdot \cos l,$$
$$z = Hh,$$

где $l = 0 \ldots 360°$, $h = -0.5 \ldots 0.5$.

Величинами R и H обозначим соответственно радиус и общую высоту цилиндра (рис. 4.22).

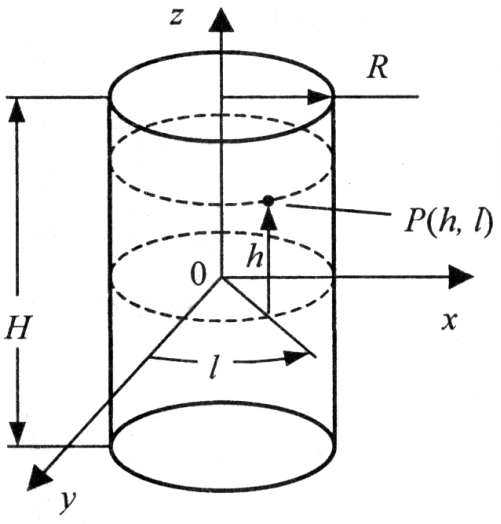


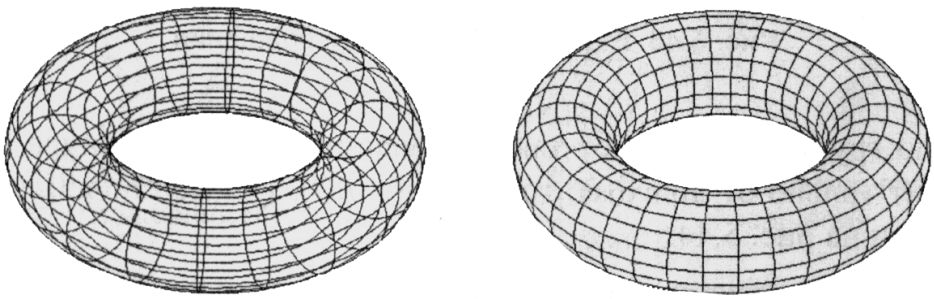
Рис. 4.22 – Цилиндр

Тор

Функции параметрического описания поверхности тора запишем в следующем виде:

$$x = (R + r \cdot \cos\varphi) \cdot \sin\omega,$$
$$y = (R + r \cdot \cos\varphi) \cdot \cos\omega$$
$$z = r \sin\varphi,$$

где R и r — большой и малый радиусы, φ и ψ — широта и долгота (рис. 4.23).



а) *б)*

Рис. 4.23 – Простейшее изображение тора: *а* — каркас, *б* — поверхность с удаленными невидимыми точками

Для замкнутой поверхности углы φ и ψ должны изменяться в полном круговом диапазоне, например от 0 до 360° или от −180° до +180°.

На рис. 4.23 показаны различные способы изображения тора: *а* — каркасное изображение; *б* — поверхность с удаленными невидимыми точками.

Контрольные вопросы по главе 4

1) Какие виды проекций относятся к параллельным проекциям?
2) Какое преобразование реализует матрица $[A] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \cos\frac{\pi}{4} & \cos\frac{\pi}{4} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$?
3) Какой вид отражения описывает эмпирическая модель Фонга?
4) Какому методу закрашивания характерен минимальный максимальный эффект полос Маха?
5) Какой вид трассировки лучей позволяет значительно сократить перебор световых лучей?

Глава 5

КРИВЫЕ И КРИВОЛИНЕЙНЫЕ ПОВЕРХНОСТИ

Современная графическая система способна с высокой скоростью выполнять операции закраски проекций плоских многоугольников, удаления невидимых поверхностей, представленных в форме множества плоских многоугольников, и наложения на них разного рода текстур. Поэтому если возникает необходимость включить в состав сцены криволинейный объект, например сферу, его зачастую стремятся с самого начала приближенно представить (аппроксимировать) множеством плоских многоугольников. Альтернативный способ, который будет представлен в этой главе, состоит в том, чтобы предоставить пользователю средства работы с криволинейными объектами. Рассмотрим три метода моделирования (форм представления) кривых линий и криволинейных поверхностей, причем основное внимание будет уделено параметрической полиномиальной форме представления [21].

5.1 Представление кривых линий и поверхностей

Рассмотрим три главных формы математического представления кривых и поверхностей — явную, неявную и параметрическую. Проанализируем достоинства и недостатки каждой из них. Для иллюстрации тех или иных положений будем использовать простейшие геометрические объекты — прямые линии, окружности, плоскости и сферические поверхности [10].

5.1.1 Представление в явной форме

Явная форма представления кривой в двухмерном пространстве (иногда говорят «представление кривой в виде явной функции») представляет собой уравнение, в левой части которого стоит зависимая переменная, а в правой части — функция, аргументом которой является независимая переменная.

В пространстве переменных x, y уравнение линии в явной форме имеет вид:

$$y = f(x).$$

Некоторые функции f имеют обратную g, которая позволяет изменить назначение зависимой и независимой переменных в уравнении, т. е. выразить x как функцию от y:

$$x = g(y).$$

Нет никакой гарантии, что для определенной линии существует явное уравнение в том или ином виде.

В трехмерном пространстве линия описывается системой из двух уравнений в явной форме. Если, например, переменная x выбрана в качестве независимой, то имеем два уравнения для зависимых переменных:

$$y = f(x),$$
$$z = g(x).$$

Для описания поверхности потребуется использовать две независимые переменные, и уравнение поверхности в явном виде будет выглядеть так:

$$z = f(x,y).$$

Как и в двухмерном пространстве, в трехмерном пространстве также не все кривые и поверхности могут быть описаны уравнениями в явной форме. Например, система уравнений:

$$y = a \cdot x + b,$$
$$z = c \cdot x + d.$$

Данная система уравнений описывает прямую в трехмерном пространстве, но таким способом нельзя описать прямую, лежащую в вертикальной плоскости x = const. Точно так же сферу нельзя представить уравнением вида $z = f(x,y)$, поскольку заданным значениям x и y соответствуют две точки на сфере.

5.1.2 Неявная форма представления

Большинство кривых и поверхностей, с которыми приходится работать на практике, можно описать с помощью уравнений в неявной форме. В двухмерном пространстве неявная форма уравнения линии имеет вид:

$$f(x,y) = 0.$$

Прямая и окружность с центром в начале координат описываются соответственно уравнениями:

$$a \cdot x + b \cdot y + c = 0,$$
$$x^2 + y^2 - r^2 = 0.$$

Функция f, по сути, выделяет из всех точек пространства те, которые принадлежат описываемой линии. Значение этой функции позволяет проверить для

каждой пары значений координат (x,y), лежит ли описываемая ими точка на данной линии. Неявная форма представления является менее зависимой от системы координат, поскольку позволяет представлять прямые или окружности во всех вариантах. В трехмерном пространстве уравнение в неявной форме, описывающее поверхность, имеет вид:

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

Плоскость описывается уравнением:

$$a \cdot x + b \cdot y + c \cdot z + d = 0,$$

где a, b, c, d — константы. Сфера радиуса r с центром в начале координат описывается уравнением:

$$x^2 + y^2 + z^2 - r^2 = 0.$$

Описать линию в трехмерном пространстве не так просто. Она может быть представлена только системой уравнений, описывающих поверхности, пересечение которых и образует эту линию, если таковые существуют:

$$f(x,y,z) = 0,$$
$$g(x,y,z) = 0.$$

Следовательно, задавшись точкой с координатами (x,y,z), нужно проверить, лежит ли она на обеих поверхностях, и, если лежит, считать ее точкой, принадлежащей формируемой линии.

Алгебраическая поверхность *— поверхность, для которой функция $f(x,y,z)$ есть сумма полиномов трех переменных.*

Частный случай алгебраической поверхности — квадратичные поверхности, в функции f представления которых отсутствуют степени переменных выше 2. Квадратичные поверхности представляют для нас особый интерес не только потому, что к ним относятся такие распространенные поверхности, как сфера, цилиндр и конус, но и потому, что такая поверхность пересекается прямой не более чем в двух точках.

5.1.3 Параметрическая форма представления

В параметрической форме значение каждой координаты точки, принадлежащей кривой, представляется функцией независимой переменной u, которая называется параметром этой кривой. В трехмерном пространстве кривая описывается системой из трех параметрических уравнений:

$$x = x(u),$$
$$y = y(u),$$
$$z = z(u).$$

Одно из главных достоинств параметрической формы представления — ее единообразие в двух- и трехмерном пространствах. В первом случае нужно просто отбросить третье уравнение для координаты z. Параметрическую форму представления можно интерпретировать как способ представления на экране образа кривой, изменяя значения параметра, как показано на рис. 5.1.

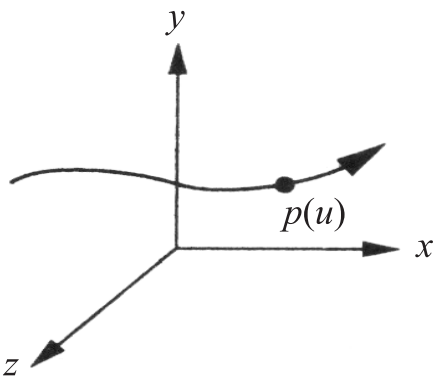


Рис. 5.1 – Сегмент кривой

Производную от параметрически заданной векторной функции можно рассматривать как вектор скорости обхода кривой, который в каждой точке этой кривой направлен по касательной к ней:

$$\frac{dp(u)}{du} = \begin{bmatrix} \dfrac{dx(u)}{du} \\ \dfrac{dy(u)}{du} \\ \dfrac{dz(u)}{du} \end{bmatrix}.$$

Для описания поверхности в параметрической форме требуется использовать два параметра. Система уравнений поверхности имеет вид:

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

Параметрическая форма описания кривых и поверхностей является, во-первых, наиболее гибкой, а во-вторых, устойчивой к любым вариациям формы и ориентации объектов, что делает ее особенно удобной для использования в математическом обеспечении систем компьютерной графики.

5.2 Общая характеристика полиномиальной параметрической формы представления

В компьютерной графике и прикладных системах автоматизации проектирования, которые строятся на базе систем компьютерной графики, кривые и поверхности используются довольно специфическим образом, в корне отличном от других

применений. Существует множество доводов в пользу использования в этой области именно параметрической формы представления криволинейных объектов. Среди них рассмотрим основные:

- возможность локального контроля формы объекта;
- гладкость и непрерывность в математическом смысле;
- возможность аналитического вычисления производных;
- устойчивость к малым возмущениям.

Наименование класса составных полиномиальных кривых — сплайн-кривые, или сплайны — произошло от английского наименования гибкой деревянной рейки или упругой стальной ленты, с помощью которой в кораблестроении вычерчивались гладкие контуры. Такая лента вставлялась между штифтами, соответствующими положению контрольных точек на кривой, и за счет упругости образовывала гладкую линию контура. Оказалось, что эта линия математически описывается сегментами полиномиальных кривых степени не выше третьей. За такими составными полиномиальными кривыми и закрепилось название сплайнов.

Сплайн *— кривая или поверхность, используемая для представления сложных гладких кривых или поверхностей.*

Возвращаясь к задачам компьютерной графики, отметим, что, помимо собственно математического описания кривых, необходимо разработать и методы их отображения.

5.3 Параметрически заданные кубические сплайны

Остановившись на классе полиномиальных кривых в параметрической форме, необходимо выбрать степень полинома для описания кривой. Если воспользоваться полиномом высокой степени, то при конструировании кривой определенной формы будут в значительной степени «развязаны руки», поскольку в распоряжении будет больше коэффициентов, но процесс расчета координат точек на кривой потребует большего количества вычислений. Кроме того, при работе с полиномами высоких степеней возрастает опасность получить кривую волнистой формы. С другой стороны, выбор полинома низкой степени может привести к тому, что в распоряжении окажется слишком мало регулируемых параметров — коэффициентов полинома — и, следовательно, не удастся воспроизвести с требуемой точностью форму кривой. Выход можно найти в том, чтобы не описывать единым полиномом высокой степени всю кривую, а разбить ее на сегменты небольшой длины, которые можно описывать полиномами низкой степени. Хотя такой полином и обладает малым количеством степеней свободы, их может оказаться вполне достаточно, чтобы воспроизвести форму кривой на отрезке небольшой длины.

Описать кубическую полиномиальную кривую можно следующим образом, воспользовавшись матрицами-строками и матрицами-столбцами:

$$p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3 = \sum_{k=0}^{3} c_k u^k = u^T c,$$

где

$$c = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}, \quad u = \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}, \quad c_k = \begin{bmatrix} c_{kx} \\ c_{ky} \\ c_{kz} \end{bmatrix}.$$

В этих выражениях c представляет матрицу коэффициентов полинома, которую можно вычислить по заданному набору опорных точек.

Рассмотрим несколько классов кубических кривых, которые отличаются характером сопоставления с опорными точками.

5.3.1 Интерполяция

Первым рассмотрим класс кубических интерполяционных полиномов. Пусть имеется четыре опорные точки в трехмерном пространстве p_0, p_1, p_2, p_3.

Каждая точка представлена тройкой своих координат:

$$p_k = \begin{bmatrix} x_k \\ y_k \\ z_k. \end{bmatrix}$$

Необходимо отыскать элементы матрицы коэффициентов c, такие, что полином $p(u) = u^T c$ будет проходить через заданные четыре опорные точки. Будем считать, что значения u_k (k = 0, …, 3) распределены равномерно на интервале $[0, 1]$, т. е. u = 0, 1/3, 2/3, 1. Соответствующие четыре уравнения в векторной форме имеют вид:

$$p_0 = p(0) = c_0,$$
$$p_1 = p\left(\frac{1}{3}\right) = c_0 + \frac{1}{3}c_1 + \left(\frac{1}{3}\right)^2 c_2 + \left(\frac{1}{3}\right)^3 c_3,$$
$$p_2 = p\left(\frac{2}{3}\right) = c_0 + \frac{2}{3}c_1 + \left(\frac{2}{3}\right)^2 c_2 + \left(\frac{2}{3}\right)^3 c_3,$$
$$p_3 = p\,(1) = c_0 + c_1 + c_2 + c_3.$$

Перепишем эти уравнения в матричной форме:

$$p = A\,c,$$

где $p = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$, $A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{3} & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\ 1 & \frac{2}{3} & \left(\frac{2}{3}\right)^2 & \left(\frac{2}{3}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$.

Матрица A является невырожденной, следовательно, ее можно обратить и получить базисную интерполяционную матрицу:

$$M_I = A^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{bmatrix}.$$

Располагая значениями элементов M_I, можно вычислить искомые значения коэффициентов:

$$c = M_I p.$$

Введем понятие функций смешивания (полиномиальных весовых функций при опорных точках), позволяющих проводить анализ гладкости интерполяционных полиноминальных кривых. Для этого перепишем полученные выше уравнения в виде:

$$p(u) = u^T \cdot c = u^T \cdot M_I \cdot p.$$

Или:

$$p(u) = b(u)^T \cdot p,$$

где $b(u) = M_I^T \cdot u$ есть матрица-столбец из четырех полиномиальных функций смешивания:

$$b(u) = \begin{bmatrix} b_0(u) \\ b_1(u) \\ b_2(u) \\ b_3(u) \end{bmatrix}.$$

В каждой функции смешивания полином является кубическим. Выразив $p(u)$ как сумму полиномов смешивания, получим:

$$p(u) = b_0(u) \cdot p_0 + b_1(u) \cdot p_1 + b_2(u) \cdot p_2 + b_3(u) \cdot p_3 = \sum_{i=0}^{3} b_i(u) \cdot p_i.$$

Из этого соотношения следует, что полиномиальные функции смешивания характеризуют вклад, который вносит каждая опорная точка, и таким образом позволяют оценить, насколько скажется на виде конечной кривой изменение положения той или иной опорной точки.

5.3.2 Кривые Эрмита

В этом разделе будет рассмотрена форма представления кривых, предложенная Эрмитом (Hermite). Предположим, что анализ сегмента кривой мы начинаем, имея в своем распоряжении только точки p_0 и p_3. Сегменту соответствует интервал изменения параметра $[0, 1]$, т. е. имеющиеся точки соответствуют значениям параметра $u = 0$ и $u = 1$. Используя те же обозначения, что и раньше, можно записать два условия:

$$p_0 = p(0) = c_0,$$
$$p_1 = p_3 = c_0 + c_1 + c_2 + c_3.$$

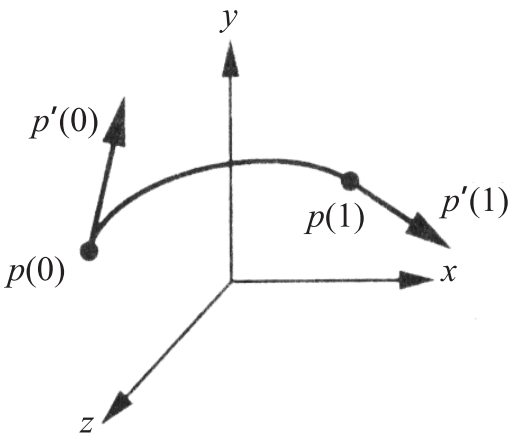


Рис. 5.2 – Определение формы Эрмита для кубической кривой

Два других условия получим, определив значения производных функции в крайних точках сегмента $u = 0$ и $u = 1$.

Производная кубического полинома есть квадратный полином:

$$p'(u) = \begin{bmatrix} \frac{du}{dx} \\ \frac{du}{dy} \\ \frac{du}{dz} \end{bmatrix}.$$

Обозначим известные значения производных в конечных точках p'_0 и p'_3 (рис. 5.2). Тогда два дополнительных условия примут вид:

$$p'_0 = p'(0) = c_1,$$
$$p'_1 = p'(1) = c_1 + 2c_2 + 3c_3.$$

Эти же уравнения можно записать и в матричной форме:

$$\begin{bmatrix} p_0 \\ p_3 \\ p'_0 \\ p'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \cdot c.$$

Обозначим через q матрицу имеющихся в нашем распоряжении данных, т. е.:

$$q = \begin{bmatrix} p_0 \\ p_3 \\ p'_0 \\ p'_3 \end{bmatrix}.$$

Таким образом, можно записать решение уравнения в виде:

$$c = M_H q,$$

где M_H называется базисной матрицей Эрмита (Hermite geometry matrix):

$$M_H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}.$$

В результате получим представление полиномиальной кривой в форме Эрмита:

$$p(u) = u^T \cdot M_H \cdot q.$$

Данную форму представления используем для представления сегментов составной кривой, как показано на рис. 5.3.

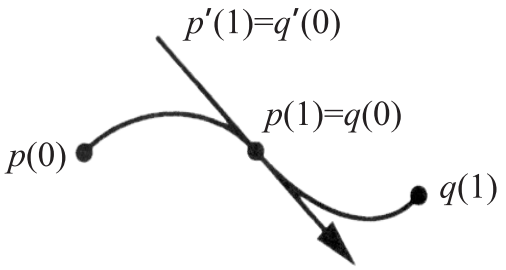


Рис. 5.3 – Применение формы представления по Эрмиту к стыковке сегментов

Точка сопряжения является общей для обоих сегментов, и, кроме того, производные к кривой в точке сопряжения для обоих сегментов также равны. В результате получаем составную кривую, непрерывную по первой производной на всем протяжении.

5.3.3 Кривые и порции поверхности в форме Безье

Кривая в форме Безье (Bezier) является очень хорошим приближением кривой в форме Эрмита, которую можно сравнивать с интерполяционным полиномом, сформированным на том же ансамбле опорных точек. Кроме того, поскольку определение кривой в форме Безье не требует задания производных, такая процедура идеально подходит для интерактивного построения криволинейных объектов в системах компьютерной графики и автоматизации проектирования.

Рассмотрим ансамбль из четырех опорных точек p_0, p_1, p_2, p_3. Пусть конечные точки формируемой кривой $p(u)$ совпадают с опорными точками p_0 и p_3:

$$p_0 = p(0), \quad p_3 = p(1).$$

Безье предложил использовать две другие опорные точки p_1 и p_2 не для интерполяции, а для задания производных в крайних точках сегмента $u = 0$ и $u = 1$. Воспользуемся линейной аппроксимацией в пространстве параметра (рис. 5.4):

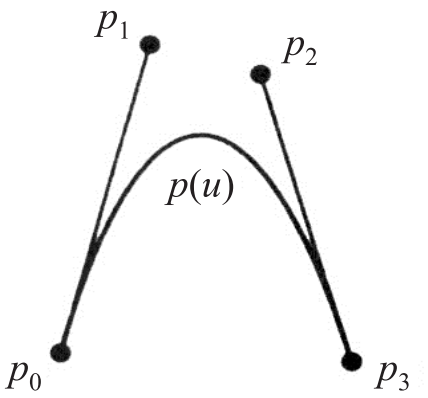


Рис. 5.4 – Аппроксимация векторов касательных

$$p'(0) = \frac{p_1 - p_0}{\frac{1}{3}} = 3(p_1 - p_0),$$

$$p'(1) = \frac{p_3 - p_2}{\frac{1}{3}} = 3(p_3 - p_2).$$

Применив эту аппроксимацию к касательным в двух крайних точках к параметрической полиномиальной кривой $p(u) = u^T c$, получим два условия:

$$3p_1 - 3p_0 = c_1,$$
$$3p_3 - 3p_2 = c_1 + 2c_2 + 3c_3.$$

Добавим их к уже имеющимся условиям совпадения кривой в конечных точках с p_0 и p_3:

$$p_0 = c_0,$$
$$p_1 = c_0 + c_1 + c_2 + c_3.$$

Таким образом, получены три набора по четыре уравнения относительно четырех неизвестных в каждом. Решая их по той же методике, что и в разделе 5.3.2, получим:

$$c = M_B q,$$

где M_B называется базисной матрицей Безье (Bezier geometry matrix). Матрица M_B имеет вид:

$$M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 1 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}.$$

Следовательно, кубический полином Безье определяется соотношением:

$$p(u) = u^T \cdot M_B \cdot p.$$

Эту формулу можно использовать точно так же, как и аналогичную формулу для составной кривой, сегменты которой являются интерполяционными полиномами. Если имеется ансамбль опорных точек $p_0, \ldots, p_n$, то первую четверку p_0, p_1, p_2, p_3 будем использовать для определения первого сегмента составной кривой, конечную точку первого сегмента p_3 и следующие три опорные точки p_4, p_5, p_6 — для определения второго сегмента и т. д.

Порции поверхностей Безье (Bezier surface patches) можно сформировать с помощью функций смешивания.Если $P = [p_{ij}]$ — массив опорных точек с размерами 4×4, то соответствующая порция поверхности в форме Безье описывается соотношением:

$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u) \cdot b_j(v) \cdot p_{ij} = u^T \cdot M_B \cdot P \cdot M_B^T \cdot v.$$

Порция поверхности проходит через угловые точки $p_{00}, p_{03}, p_{30}, p_{33}$ и не выходит за пределы выпуклого многогранника, вершинами которого являются опорные точки (рис. 5.5). Двенадцать опорных точек из 16 можно интерпретировать как данные, определяющие направление производных по разным параметрам в угловых точках формируемой порции поверхности.

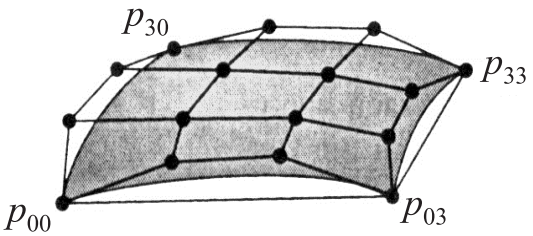


Рис. 5.5 – Порция поверхности Безье

5.4 Кубические B-сплайны

Кубические кривые Безье и порции кубических поверхностей Безье довольно широко используются в задачах компьютерной графики и автоматизации проектирования.

Кубические B-сплайны представляют собой усовершенствованную методику построения кубических кривых. Здесь снимается требование, чтобы формируемая кривая проходила через опорные точки, и накладывается новое — чтобы она проходила близко к ним. При этих условиях довольно просто обеспечить непрерывность не только самой составной кривой, но и ее первой и второй производных в точках сопряжения сегментов.

5.4.1 Обобщенные B-сплайны

Предположим, что имеется ансамбль опорных точек $p_0, \ldots, p_m$. В общем виде задача аппроксимации формулируется как поиск функции $p(u) = \left[x(u) \cdot y(u) \cdot z(u)\right]^T$, определенной на интервале $u_{\min} \leqslant u \leqslant u_{\max}$, которая является достаточно гладкой

и проходит достаточно близко к опорным точкам. Предположим, что имеется множество значений $\{u_k\}$, называемых узлами (knots), такое, что:

$$u_{\min} = u_0 \leqslant u_1 \leqslant \ldots \leqslant u_n = u_{\max}.$$

Последовательность $u_0, u_1, \ldots, u_n$ будем называть массивом узлов. При использовании аппроксимации сплайнами функция $p(u)$ имеет вид полинома степени d на интервале между соседними узлами:

$$p(u) = \sum_{j=0}^{d} c_{jk} \cdot u^j, \quad u_k < u < u_{k+1}.$$

Следовательно, для того чтобы определить сплайн степени d, потребуется определить $n(d+1)$ трехмерных коэффициентов c_{jk}. Необходимые для этого уравнения можно получить, рассматривая разного рода ограничения, связанные с непрерывностью функции и критерием близости к опорным точкам.

Например, если $d = 3$, то мы имеем дело с кубическим полиномом на каждом интервале, т. е. при данном n нужно сформулировать $4n$ ограничений. Существует $(n-1)$ внутренний узел, и, формулируя ограничения, связанные с непрерывностью в этих узлах, получим $(3n-3)$ уравнения. Если же требуется, чтобы функция еще и проходила через $(n+1)$ опорную точку, то общее число уравнений составит $(4n-2)$. Остается сформулировать еще два ограничения (получить еще два уравнения). Их можно получить, задавшись определенным направлением касательной к формируемой кривой в начальной и конечной точках. Такой подход к формированию сплайна является глобальным — нужно решить систему из $4n$ уравнений относительно $4n$ неизвестных, а значит, каждый полученный коэффициент будет зависеть от всех опорных точек. Данная методика определения коэффициентов сплайна обеспечивает получение гладкой кривой, проходящей через заданные опорные точки.

5.5 Построение кривых и поверхностей

Если уж в состав сцены включены объекты, состоящие из кривых и поверхностей, необходимо применить методы их воспроизведения. Существует несколько подходов к выполнению подобной процедуры.

Первый подход — можно вычислить точки пересечения с таким объектом лучей, исходящих из центра проецирования и проходящих через определенные пиксели картинной плоскости. Однако вычисление пересечений с криволинейными объектами требует решения нелинейных уравнений, что не так-то просто сделать, учитывая, что все вычисления следует выполнять в реальном масштабе времени.

Второй подход состоит в том, чтобы вычислить массив вершин, принадлежащих криволинейному объекту, и построить на основе этого массива приближение криволинейного объекта множеством плоских из стандартного набора.

Контрольные вопросы по главе 5

1) Чем отличается кривая Безье от кривой Эрмита?
2) С помощью каких функций можно сформировать порции поверхностей Безье?
3) Как называют наименьший элемент растровой графики?
4) С помощью каких функций можно сформировать порции поверхностей Безье?
5) Какая задача формулируется как поиск функции $p(u) = \left[x(u) \cdot y(u) \cdot z(u)\right]^T$, определенной на интервале $u_{\min} \leqslant u \leqslant u_{\max}$, которая является достаточно гладкой и проходит достаточно близко к опорным точкам?

Глава 6

ГРАФИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

6.1 OpenGl. Архитектура и особенности синтаксиса

OpenGL*(Open Graphic Library) — библиотека графических функций, интерфейс для графических прикладных программ. Разработана Silicon Graphics.*

OpenGL является на данный момент одним из самых популярных программных интерфейсов (API) для разработки приложений в области двумерной и трехмерной графики. Стандарт OpenGL был разработан и утвержден в 1992 году ведущими фирмами в области разработки программного обеспечения.

Библиотека насчитывает около 120 различных команд, которые программист использует для задания объектов и операций, необходимых для написания интерактивных графических приложений.

Характерными особенностями OpenGL, которые обеспечили распространение и развитие этого графического стандарта, являются [7, 18]:

- Стабильность — дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.
- Надежность и переносимость — приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации. Кроме того, эти приложения могут выполняться как на персональных компьютерах, так и на рабочих станциях и суперкомпьютерах.
- Легкость применения — стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет с меньшими затратами

создавать эффективные приложения, содержащие меньше строк кода, чем с использованием других графических библиотек. Необходимые функции для обеспечения совместимости с различным оборудованием реализованы на уровне библиотеки и значительно упрощают разработку приложений.

Основные возможности OpenGL:

- Набор базовых примитивов: точки, линии, многоугольники и др.
- Видовые и координатные преобразования.
- Удаление невидимых линий и поверхностей (Z-буфер).
- Использование сплайнов для построения линий и поверхностей.
- Наложение текстуры и применение освещения.
- Добавление специальных эффектов: туман, изменение прозрачности, сопряжение цветов (blending), устранение ступенчатости (anti-aliasing).

Описывать возможности OpenGL мы будем через функции его библиотеки. Все функции можно разделить на пять категорий:

- Функции описания примитивов определяют объекты нижнего уровня иерархии (примитивы), которые способна отображать графическая подсистема. В OpenGL в качестве примитивов выступают точки, линии, многоугольники и т. д.
- Функции описания источников света служат для описания положения и параметров источников света, расположенных в трехмерной сцене.
- Функции задания атрибутов. С помощью задания атрибутов программист определяет, как будут выглядеть на экране отображаемые объекты. Другими словами, если с помощью примитивов определяется, что появится на экране, то атрибуты определяют способ вывода на экран. В качестве атрибутов OpenGL позволяет задавать цвет, характеристики материала, текстуры, параметры освещения.
- Функции визуализации позволяют задать положение наблюдателя в виртуальном пространстве, параметры объектива камеры. Зная эти параметры, система сможет не только правильно построить изображение, но и отсечь объекты, оказавшиеся вне поля зрения.
- Набор функций геометрических преобразований позволяет программисту выполнять различные преобразования объектов — поворот, перенос, масштабирование.

При этом OpenGL может выполнять дополнительные операции, такие, как использование сплайнов для построения линий и поверхностей, удаление невидимых фрагментов изображений, работа с изображениями на уровне пикселей и т. д.

6.1.1 Интерфейс OpenGL

OpenGL состоит из набора библиотек (рис. 6.1). Все базовые функции хранятся в основной библиотеке opengl32.dll. Помимо основной, OpenGL включает в себя несколько дополнительных библиотек [21].

Первая из них — библиотека утилит GLU — GL Utility. Все функции этой библиотеки определены через базовые функции opengl32.dll. В состав GLU вошла реализация более сложных функций, таких, как набор популярных геометрических примитивов (куб, шар, цилиндр, диск), функции построения сплайнов, реализация дополнительных операций над матрицами и т. п.

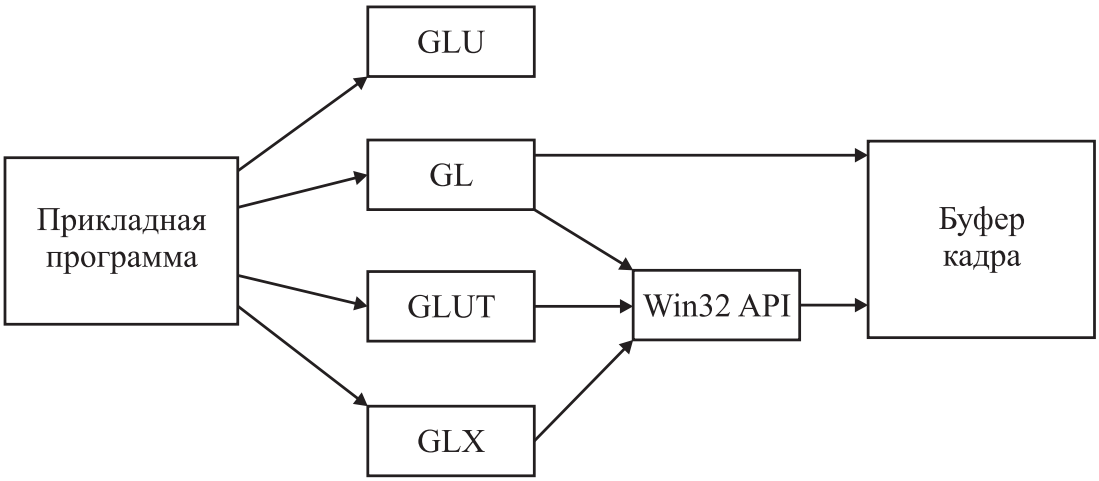


Рис. 6.1 – Организация библиотеки OpenGL

OpenGL не включает в себя никаких специальных команд для работы с окнами или ввода информации от пользователя. Поэтому были созданы специальные переносимые библиотеки для обеспечения часто используемых функций взаимодействия с пользователем и для отображения информации с помощью оконной подсистемы. Наиболее популярной является библиотека GLUT (GL Utility Toolkit). Формально GLUT не входит в OpenGL, но включается почти во все его дистрибутивы и имеет реализации для различных платформ. GLUT предоставляет только минимально необходимый набор функций для создания OpenGL-приложения. Функционально аналогичная библиотека GLX менее популярна. В дальнейшем, в этом пособии в качестве основной будет рассматриваться GLUT.

Кроме того, функции, специфичные для конкретной оконной подсистемы, обычно входят в ее прикладной программный интерфейс. Так, функции, поддерживающие выполнение OpenGL, есть в составе Win32 API и X Window. На рис. 6.1 схематически представлена организация системы библиотек в версии, работающей под управлением системы Windows. Аналогичная организация используется и в других версиях OpenGL.

6.1.2 Архитектура OpenGL

Функции OpenGL реализованы в модели клиент-сервер. Приложение выступает в роли клиента — оно вырабатывает команды, а сервер OpenGL интерпретирует и выполняет их. Сам сервер может находиться как на том же компьютере, на котором находится клиент (например, в виде динамически загружаемой библиотеки — DLL), так и на другом (при этом может быть использован специальный протокол передачи данных между машинами).

GL обрабатывает и рисует в буфере кадра графические *примитивы* с учетом некоторого числа выбранных режимов. Каждый примитив — это точка, отрезок, многоугольник и т. д. Каждый режим может быть изменен независимо от других. Определение примитивов, выбор режимов и другие операции описываются с помощью *команд* в форме вызовов функций прикладной библиотеки.

Примитивы определяются набором из одной или более *вершин* (vertex). Вершина определяет точку, конец отрезка или угол многоугольника. С каждой вершиной ассоциируются некоторые данные (координаты, цвет, нормаль, текстурные координаты и т. д.), называемые *атрибутами*. В подавляющем большинстве случаев каждая вершина обрабатывается независимо от других.

С точки зрения архитектуры графическая система OpenGL является конвейером, состоящим из нескольких последовательных этапов обработки графических данных (рис. 6.2).

Рис. 6.2 – Функционирование конвейера OpenGL

Команды OpenGL всегда обрабатываются в том порядке, в котором они поступают, хотя могут происходить задержки перед тем, как проявится эффект от их выполнения. В большинстве случаев OpenGL предоставляет непосредственный интерфейс, т. е. определение объекта вызывает его визуализацию в буфере кадра.

С точки зрения разработчиков, OpenGL — это набор команд, которые управляют использованием графической аппаратуры. Если аппаратура состоит только из адресуемого буфера кадра, тогда OpenGL должен быть реализован полностью с ис-

пользованием ресурсов центрального процессора. Обычно графическая аппаратура предоставляет различные уровни ускорения: от аппаратной реализации вывода линий и многоугольников до изощренных графических процессоров с поддержкой различных операций над геометрическими данными.

OpenGL является прослойкой между аппаратурой и пользовательским уровнем, что позволяет предоставлять единый интерфейс на разных платформах, используя возможности аппаратной поддержки.

Кроме того, OpenGL можно рассматривать как конечный автомат, состояние которого определяется множеством значений специальных переменных и значениями текущей нормали, цвета, координат текстуры и других атрибутов и признаков. Вся эта информация будет использована при поступлении в графическую систему координат вершины для построения фигуры, в которую она входит. Смена состояний происходит с помощью команд, которые оформляются как вызовы функций.

6.2 Синтаксис команд OpenGL

Все команды (процедуры и функции) библиотеки OpenGL начинаются с префикса gl, все константы — с префикса GL_.

Соответствующие команды и константы библиотек GLU и GLUT аналогично имеют префиксы glu (GLU_) и glut (GLUT_).

Кроме того, в имена команд входят суффиксы, несущие информацию о числе и типе передаваемых параметров. В OpenGL полное имя команды имеет вид:

type **glCommand_name[1 2 3 4][b s i f d ub us ui][v]**
(type1 *arg*1, …, type*N* *argN*)

Имя состоит из нескольких частей:

1) **gl** имя библиотеки, в которой описана эта функция: для базовых функций OpenGL, функций из библиотек GL, GLU, GLUT, GLAUX, это gl, glu, glut, aux соответственно.
2) **Command_name** имя команды (процедуры или функции) **[1 2 3 4]** число аргументов команды.
3) **[b s i f d ub us ui]** тип аргумента:
   - символ b — GLbyte (аналог char),
   - символ i — GLint (аналог int),
   - символ f — GLfloat (аналог float) и так далее.
4) **[v]** — наличие этого символа показывает, что в качестве параметров функции используется указатель на массив значений.

Символы в квадратных скобках в некоторых названиях не используются. Например, команда glVertex2i() описана в библиотеке GL и использует в качестве параметров два целых числа, а команда glColor3fv() использует в качестве параметра указатель на массив из трех вещественных чисел.

6.3 Отрисовка примитивов

Под вершиной в OPenGl понимается точка в трехмерном пространстве, координаты которой можно задавать следующим образом:

glVertex[2 3 4][s i f d](cords: type);

glVertex[2 3 4][s i f d]v(cords: ^type).

Координаты точки задаются максимум четырьмя значениями: x, y, z, w, при этом можно указывать два (x,y) или три (x,y,z) значения, а для остальных переменных в этих случаях используются значения по умолчанию: $z = 0$, $w = 1$. Как уже было сказано выше, число в названии команды соответствует числу явно задаваемых значений, а последующий символ — их типу.

Координатные оси расположены так, что точка $(0,0)$ находится в левом нижнем углу экрана, ось x направлена влево, ось y — вверх, а ось z — из экрана.

Чтобы задать какую-нибудь фигуру, одних координат вершин недостаточно — эти вершины надо объединить в одно целое, определив необходимые свойства. Для этого в OpenGL используется понятие примитивов, к которым относятся точки, линии, связанные или замкнутые линии, треугольники и так далее. Задание примитива происходит внутри командных скобок:

glBegin(mode: GLenum)

...

glEnd

Параметр mode определяет тип примитива, который задается внутри и может принимать следующие значения (рис. 6.3):

Отрезки:

Значение <тип>	Описание
GL_POINTS	Каждый вызов glVertex задает отдельную точку
GL_LINES	Каждая пара вершин задает отрезок
GL_LINE_STRIP	Рисуется ломанная
GL_LINE_LOOP	Рисуется ломанная, причем ее последняя точка соединяется с первой

Многоугольники:

Значение <тип>	Описание
GL_TRIANGLES	Тройки вершин образуют треугольник
GL_TRIANGLE_STRIP	Связанные треугольники
GL_TRIANGLE_FAN	Связанные треугольники с общей первой вершиной
GL_QUADS	Каждые четыре вершины образуют четырехугольники
GL_QUAD_STRIP	Связанные четырехугольники
GL_POLYGON	Один выпуклый многоугольник

```
glBegin(<тип>); // указываем, что будем рисовать
    glVertex[2 3][i f d](...); // первая вершина
    ...                        // остальные вершины
    glVertex[2 3][i f d](...); // последняя вершина
glEnd; // закончили рисовать примитив
```

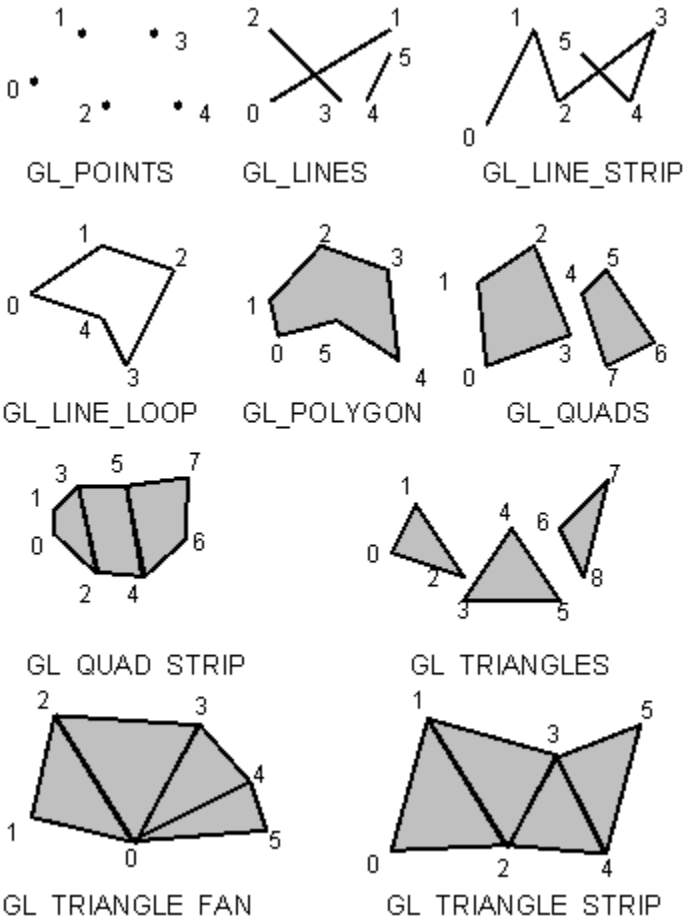
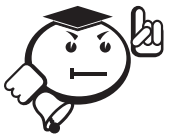


Рис. 6.3 – Примитивы OpenGL



Отметим, что в OpenGL левый нижний угол области вывода имеет координаты $[-1;-1]$, правый верхний $[1;1]$.

Рисуем четыре точки:

```
glPointSize(20); // размер точки
glColor3f (1.0, 1.0, 1.0); // цвет точки
glBegin (GL_POINTS);
      glVertex2f (-1, -1);
      glVertex2f (-1, 1);
      glVertex2f (0, 0);
      glVertex2f (1, -1);
      glVertex2f (1, 1);
glEnd;
```

Рисуем два отрезка, соединяющие углы окна по диагоналям:

```
glLineWidth(2.5); //толщина линии
glBegin (GL_LINES);
      glVertex2f (-1, -1);
```

```
    glVertex2f (1, 1);
    glVertex2f (1, -1);
    glVertex2f (-1, 1);
glEnd;
```

Если вершин много, то, чтобы не вызывать для каждой команду **glVertex**..(), удобно объединять вершины в массивы, используя команду:

glVertexPointer(size: GLint, type: GLenum, stride: GLsizei, ^ptr),

которая определяет способ хранения и координаты вершин.

При этом *size* определяет число координат вершины (может быть равен 2, 3, 4), *type* определяет тип данных (может быть равен **GL_SHORT**, **GL_INT**, **GL_FLOAT**, **GL_DOUBLE**).

Иногда удобно хранить в одном массиве другие атрибуты вершины, и тогда параметр *stride* задает смещение от координат одной вершины до координат следующей; если *stride* равен нулю, это значит, что координаты расположены последовательно. В параметре *ptr* указывается адрес, где находятся данные.

Для задания текущего цвета вершины используются команды:

glColor[3 4][b s i f](components: GLtype);

glColor[3 4][b s i f]v(components: ^GLtype).

Первые три параметра задают *R*, *G*, *B* — компоненты цвета, а последний параметр определяет alpha-компоненту, которая задает уровень прозрачности объекта. Если в названии команды указан тип f (float), то значения всех параметров должны принадлежать отрезку $[0,1]$, при этом по умолчанию значение alpha-компоненты устанавливается равным 1.0, что соответствует полной непрозрачности. Если указан тип ub (unsigned byte), то значения должны лежать в отрезке $[0,255]$.

Разным вершинам можно назначать различные цвета, и тогда будет проводиться линейная интерполяция цветов по поверхности примитива.

Для управления режимом интерполяции цветов используется команда **glShadeModel**(mode: GLenum) вызов которой с параметром **GL_SMOOTH** включает интерполяцию (установка по умолчанию), а с **GL_FLAT** отключает.

Пример

Например, чтобы нарисовать треугольник с разными цветами в вершинах, достаточно написать:

```
glBegin(GL_TRIANGLE);
    glColor3f(1.0, 0.0, 0.0); //красный
    glVertex3f(0.0, 0.0, 0.0);
    glColor3ub(0,255,0); //зеленый
    glVertex3f(1.0, 0.0, 0.0);
    glColor3f(0.0, 0.0, 1.0); //синий
    glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Для задания цвета фона используется команда **glClearColor**(red, green, blue, alpha: GLclampf). Значения должны находиться в отрезке $[0,1]$ и по умолчанию равны нулю. После этого вызов команды void **glClear**(mask: GLbitfield) с параметром **GL_COLOR_BUFFER_BIT** устанавливает цвет фона во все буфера, доступные для записи цвета (иногда удобно использовать несколько буферов цвета).

Кроме цвета, аналогичным образом можно определить нормаль в вершине, используя команды:

glNormal3[b s i f d](coords: type);

glNormal3[b s i f d]v(coords: ^type).

Задаваемый вектор может не иметь единичной длины, но он будет нормироваться автоматически в режиме нормализации, который включается вызовом команды **glEnable**(GL_NORMALIZE). Команды:

glEnable(mode: GLenum),

glDisable(mode: GLenum),

производят включение и отключение того или иного режима работы конвейера OpenGL. Эти команды применяются достаточно часто, и их влияние будет рассматриваться в конкретных случаях.

Вообще, внутри командных скобок **glBegin**() и **glEnd** можно производить вызов лишь нескольких команд, в которые входят **glVertex**..(), **glColor**..(), **glNormal**..(), **glRect**..(), **glMaterial**..() и **glTexCoord**..().

Последние две команды будут рассматриваться ниже, а с помощью команды **glRect**[s i f d](x_1, y_1, x_2, y_2: GLtype), void **glRect**[s i f d]v(v_1, v_2: ^GLtype) можно нарисовать прямоугольник в плоскости $z = 0$ с координатами противоположных углов (x_1, y_1) и (x_2, y_2) либо набор прямоугольников с координатами углов в массивах v_1 и v_2.

Кроме задания самих примитивов, можно определить метод их отображения на экране, где под примитивами в данном случае понимаются многоугольники.

Однако сначала надо определить понятие лицевых и обратных граней. Под гранью понимается одна из сторон многоугольника, и по умолчанию лицевой считается та сторона, вершины которой обходятся против часовой стрелки. Направление обхода вершин лицевых сторон можно изменить вызовом команды **glFrontFace**(mode: GLenum) со значением параметра *mode* равным **GL_CW**, а отменить — с **GL_CCW**.

Чтобы изменить метод отображения многоугольника, используется команда **glPolygonMode**(face, mode: GLenum).

Параметр *mode* определяет, как будут отображаться многоугольники, а параметр *face* устанавливает тип многоугольников, к которым будет применяться эта команда, и может принимать следующие значения:

- **GL_FRONT** для лицевых граней.
- **GL_BACK** для обратных граней.
- **GL_FRONT_AND_BACK** для всех граней.

Параметр *mode* может быть равен:

- **GL_POINT** — при таком режиме будут отображаться только вершины многоугольников.
- **GL_LINE** — при таком режиме многоугольник будет представляться набором отрезков.

- **GL_FILL** — при таком режиме многоугольники будут закрашиваться текущим цветом с учетом освещения и этот режим установлен по умолчанию.

Кроме того, можно указывать, какой тип граней отображать на экране. Для этого сначала надо установить соответствующий режим вызовом команды **glEnable**(**GL_CULL_FACE**), а затем выбрать тип отображаемых граней с помощью команды **glCullFace**(mode: GLenum)

Вызов с параметром **GL_FRONT** приводит к удалению из изображения всех лицевых граней, а с параметром **GL_BACK** — обратных (установка по умолчанию).

6.4 Матрицы преобразований в OpenGL

В этом разделе речь пойдет о реализации механизма преобразований в однородных координатах в виде программного пакета и об интерфейсе между этим пакетом и прикладной программой. В OpenGL существуют три матрицы, которые входят в состав параметров, характеризующих текущее состояние графической системы [7]. Здесь мы рассмотрим только матрицу вида (model-view matrix). Всеми тремя матрицами можно манипулировать с помощью одного и того же набора функций, а для выбора, с какой именно матрицей выполняются операции, используется функция glMatrixMode().

6.4.1 Текущая матрица преобразования

В большинстве графических систем используется текущая матрица преобразования — ТМП (CTM — current transformation matrix). Эта матрица применяется для преобразования всех вершин. Если изменяется ТМП, изменяется текущее состояние системы. Умножение на ТМП является одной из стадий конвейерного процесса обработки информации в графической системе. Обозначим матрицу ТМП через C. Тогда, если p — это вершина, то при «перемещении» ее по конвейеру формируется произведение Cp. Матрица ТМП имеет размер 4×4 и может быть изменена функциями, которые входят в состав графического пакета.

В исходном состоянии ТМП является единичной матрицей размера 4×4; при необходимости в любой момент прикладная программа может ее реинициализировать. Будем использовать символ $\leftarrow$ для обозначения процедуры замены содержимого матрицы. Операция инициализации в нашей системе обозначений будет выражаться следующим образом:

$$C \leftarrow I.$$

Функции изменения C разделены на две группы: функции присвоения новых значений элементам матрицы и функции преобразования матрицы умножением ее на другую матрицу справа или слева. В подавляющем большинстве графических систем поддерживаются три вида преобразований: сдвиг, масштабирование с фиксированной точкой в начале координат и поворот с фиксированной точкой в начале координат.

6.4.2 Преобразования координат и проекции

В OpenGL используются три системы координат: левосторонняя, правосторонняя и оконная. Первые две системы являются трехмерными и отличаются друг от друга направлением оси *z* (в правосторонней она направлена на наблюдателя, а в левосторонней — вглубь экрана). Левосторонняя система используется для задания значений параметрам команды **gluPerspective**(), **glOrtho**(), которые будут рассмотрены ниже, а правосторонняя, или мировая, система координат во всех остальных случаях. Отображение трехмерной информации происходит в двумерную оконную систему координат.

Для задания различных преобразований объектов сцены в OpenGL используются операции над матрицами, при этом различают три типа матриц: *видовая*, *проекций* и *текстуры*. Все они имеют размер 4×4.

Видовая матрица *определяет преобразования объекта в мировых координатах, такие как параллельный перенос, изменение масштаба и поворот.*

Матрица проекций *задает, как будут проецироваться трехмерные объекты на плоскость экрана.*

Матрица текстуры *определяет наложение текстуры на объект.*

Для того чтобы выбрать, какую матрицу надо изменить, используется команда **glMatrixMode**(mode: GLenum), вызов которой со значением параметра mode равным **GL_MODELVIEW**, **GL_PROJECTION**, **GL_TEXTURE** включает режим работы с видовой, матрицей проекций и матрицей текстуры соответственно. Для вызова команд, задающих матрицы того или иного типа, необходимо сначала установить соответствующий режим.

Для определения элементов матрицы текущего типа вызывается команда **glLoadMatrix**[f d](m: ^GLtype), где *m* указывает на массив из 16 элементов типа ***float*** или ***double*** в соответствии с названием команды, при этом сначала в нем должен быть записан первый столбец матрицы, затем второй, третий и четвертый.

Команда **glLoadIdentity** заменяет текущую матрицу на единичную.

Часто нужно сохранить содержимое текущей матрицы для дальнейшего использования, для чего используют команды **glPushMatrix** и **glPopMatrix**. Они записывают и восстанавливают текущую матрицу из стека, причем для каждого типа матриц стек свой. Для видовых матриц его глубина равна как минимум 32, а для двух оставшихся типов как минимум 2.

Для умножения текущей матрицы слева на другую матрицу используется команда **glMultMatrix**[f d](m: ^GLtype), где *m* должен задавать матрицу размером

4×4 в виде массива с описанным расположением данных. Обычно для изменения матрицы того или иного типа удобно использовать специальные команды, которые по значениям своих параметров создают нужную матрицу и перемножают ее с текущей. Чтобы сделать текущей созданную матрицу, надо перед вызовом этой команды вызвать **glLoadIdentity**().

Для выполнения поворота, сдвига и масштабирования служат три специальные функции:

- функция поворота **glRotetef**(angle, vx, vy, vz), первый аргумент *angle* задает угол поворота в градусах, а три последующих — *vx*, *vy* и *vz* — компоненты вектора оси поворота.
- Функция сдвига **glTranslatef**(dx, dy, dz), аргументы функции сдвига — компоненты вектора смещения.
- функции масштабирования **glScalef**(sx, sy, sz), аргументы функции масштабирования — масштабные коэффициенты по координатным осям.

Каждая из этих функций изменяет выбранную с помощью **glMatrixMode**() матрицу текущего состояния, домножая ее справа.

6.4.3 Проекции в OpenGl

В OpenGL существуют ортографическая (параллельная) и перспективная проекции. Первый тип проекции может быть задан командами:

glOrtho(left, right, bottom, top, znear, far: GLdouble);

gluOrtho2D(left, right, bottom, top: GLdouble).

Первая команда создает матрицу проекции в усеченный объем видимости (параллелограмм видимости) в левосторонней системе координат. Параметры команды задают точки $(left, bottom, -znear)$ и $(right, top, -znear)$, которые отвечают левому нижнему и правому верхнему углам окна вывода. Параметры *znear* и *far* задают расстояние до ближней и дальней плоскостей отсечения по дальности от точки $(0, 0, 0)$ и могут быть отрицательными (рис. 6.4).

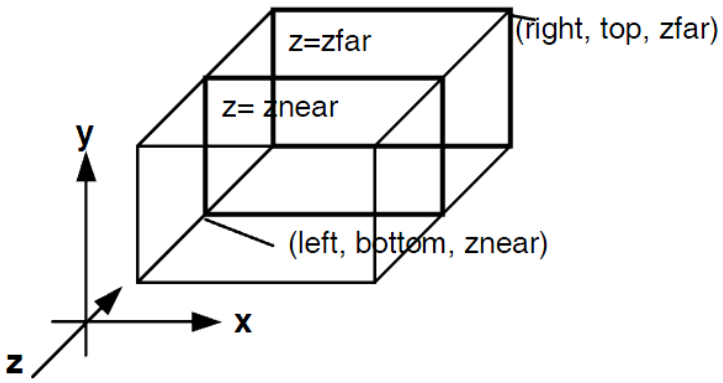


Рис. 6.4 – Ортографическая проекция в OpenGL

Во второй команде, в отличие от первой, значения *znear* и *far* устанавливаются равными -1 и 1 соответственно.

Перспективная проекция определяется командой:

gluPerspective(angley, aspect, znear, zfar: GLdouble).

Данная команда задает усеченную пирамиду видимости в левосторонней системе координат (рис. 6.5).

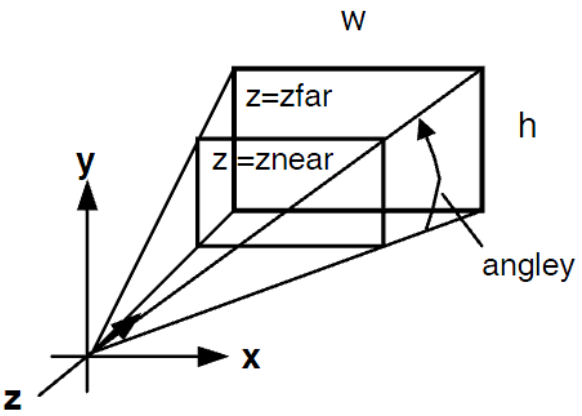


Рис. 6.5 – Перспективная проекция в OpenGL

Параметр *angley* определяет угол видимости в градусах по оси *y* и должен находиться в диапазоне от 0° до 180°. Угол видимости вдоль оси *x* задается параметром *aspect*, который обычно задается как отношение сторон области вывода. Параметры *zfar* и *znear* задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными. Чем больше отношение *zfar/znear*, тем хуже в буфере глубины будут различаться расположенные рядом поверхности, так как по умолчанию в него будет записываться «сжатая» глубина в диапазоне от 0 до 1.

6.4.4 Область вывода

После применения матрицы проекций на вход следующего преобразования подаются так называемые усеченные (clip) координаты, для которых значения всех компонент $(xc, yc, zc, wc)T$ находятся в отрезке $[-1, 1]$. После этого находятся нормализованные координаты вершин по формуле:

$$(xn, yn, zn)T = \left(\frac{xc}{wc}, \frac{yc}{wc}, \frac{zc}{wc}\right)T.$$

Область вывода представляет из себя прямоугольник в оконной системе координат, размеры которого задаются командой:

glViewPort(x, y, width, height: GLint).

Значения всех параметров задаются в пикселях и определяют ширину и высоту области вывода с координатами левого нижнего угла (x, y) в оконной системе координат. Размеры оконной системы координат определяются текущими размерами окна приложения, точка $(0, 0)$ находится в левом нижнем углу окна.

Используя параметры команды **glViewPort**(), вычисляются оконные координаты центра области вывода (ox, oy) по формулам $ox = x + width/2$, $oy = y + height/2$.

Для отработки буфера глубины (для корректного отображения трехмерных объектов и сцен) данную возможность необходимо инициализировать, вызвав команду **glEnable(GL_DEPTH_TEST)**.

6.5 Визуальные эффекты в OpenGl

Этот раздел посвящен тому, как повысить качество получаемых образов и как получить некоторые специальные эффекты.

6.5.1 Материалы и освещение

Для создания реалистических изображений необходимо определить как свойства самого объекта, так и свойства среды, в которой он находится. Первая группа свойств включает в себя параметры материла, из которого сделан объект, способы нанесения текстуры на его поверхность, степень прозрачности объекта. Ко второй группе можно отнести количество и свойства источников света, уровень прозрачности среды. Все эти свойства можно задавать, используя соответствующие команды OpenGL.

Для задания параметров текущего материала используются команды:

glMaterial[i f](face, pname: GLenum, param: GLtype);

glMaterial[i f]v(face, pname: GLenum, params: ^GLtype).

С их помощью можно определить рассеянный, диффузный и зеркальный цвета материала, а также цвет степени зеркального отражения и интенсивность излучения света, если объект должен светиться. Какой именно параметр будет определяться значением *param*, зависит от значения *pname*:

- **GL_AMBIENT** параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют рассеянный цвет материала (цвет материала в тени). Значение по умолчанию: (0.2, 0.2, 0.2, 1.0).
- **GL_DIFFUSE** параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет диффузного отражения материала. Значение по умолчанию:(0.8, 0.8, 0.8, 1.0).
- **GL_SPECULAR** параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения материала. Значение по умолчанию: (0.0, 0.0, 0.0, 1.0).
- **GL_SHININESS** параметр *params* должен содержать одно целое или вещественное значение в диапазоне от 0 до 128, которое определяет степень зеркального отражения материала. Значение по умолчанию: 0.
- **GL_EMISSION** параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют интенсивность излучаемого света материала. Значение по умолчанию: (0.0, 0.0, 0.0, 1.0).
- **GL_AMBIENT_AND_DIFFUSE** эквивалентно двум вызовам команды **glMaterial**..() со значением pname **GL_AMBIENT** и **GL_DIFFUSE** и одинаковыми значениями *params*.

В большинстве моделей учитывается диффузный и зеркальный отраженный свет; первый определяет естественный цвет объекта, а второй — размер и форму бликов на его поверхности.

Параметр *face* определяет тип граней, для которых задается этот материал и может принимать значения **GL_FRONT**, **GL_BACK** или **GL_FRONT_AND_BACK**.

Если в сцене материалы объектов различаются лишь одним параметром, рекомендуется сначала установить нужный режим, вызвав **glEnable**() с параметром **GL_COLOR_MATERIAL**, а затем использовать команду:

glColorMaterial(face, pname: GLenum),

где параметр *face* имеет аналогичный смысл, а параметр pname может принимать все перечисленные значения. После этого, значения выбранного с помощью *pname* свойства материала для конкретного объекта (или вершины) устанавливаются вызовом команды **glColor**..(), что позволяет избежать вызовов более ресурсоемкой команды **glMaterial**..() и повышает эффективность программы.

Добавить в сцену источник света можно с помощью команд:

glLight[i f](light, pname: GLenum, param: GLfloat);

glLight[i f]v(light, pname: GLenum, params: ^GLfloat).

Параметр *light* однозначно определяет источник и выбирается из набора специальных символических имен вида **GL_LIGHT*i***, где *i* должно лежать в диапазоне от 0 до **GL_MAX_LIGHT**, которое не превосходит восьми.

Оставшиеся два параметра имеют аналогичный смысл, что и в команде **glMaterial**..(). Рассмотрим их назначение:

- **GL_SPOT_EXPONENT** параметр *param* должен содержать целое или вещественное число от 0 до 128, задающее распределение интенсивности света. Этот параметр описывает уровень сфокусированности источника света. Значение по умолчанию: 0 (рассеянный свет).
- **GL_SPOT_CUTOFF** параметр *param* должен содержать целое или вещественное число между 0 и 90 или равное 180, которое определяет максимальный угол разброса света. Значение этого параметра есть половина угла в вершине конусовидного светового потока, создаваемого источником. Значение по умолчанию: 180 (рассеянный свет).
- **GL_AMBIENT** параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет фонового освещения. Значение по умолчанию: (0.0, 0.0, 0.0, 1.0).
- **GL_DIFFUSE** параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет диффузного освещения. Значение по умолчанию: (1.0, 1.0, 1.0, 1.0) для **LIGHT0** и (0.0, 0.0, 0.0, 1.0) для остальных.
- **GL_SPECULAR** параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения. Значение по умолчанию: (1.0, 1.0, 1.0, 1.0) для **LIGHT0** и (0.0, 0.0, 0.0, 1.0) для остальных.
- **GL_POSITION** параметр *params* должен содержать четыре целых или вещественных числа, которые определяют положение источника света. Ес-

ли значение компоненты w равно 0.0, то источник считается бесконечно удаленным и при расчете освещенности учитывается только направление на точку (x, y, z), в противном случае считается, что источник расположен в точке (x, y, z, w). Значение по умолчанию: (0.0, 0.0, 1.0, 0.0).

- **GL_SPOT_DIRECTION** параметр params должен содержать четыре целых или вещественных числа, которые определяют направление света. Значение по умолчанию: (0.0, 0.0, –1.0, 1.0).

Для использования освещения сначала надо установить соответствующий режим вызовом команды **glEnable**(**GL_LIGHTING**), а затем включить нужный источник командой **glEnable**(**GL_LIGHT*n***).

В OpenGL используется модель освещения Фонга, в соответствии с которой цвет точки определяется несколькими факторами: свойствами материала и текстуры, величиной нормали в этой точке, а также положением источника света и наблюдателя. Для корректного расчета освещенности в точке надо использовать единичные нормали, однако команды типа **glScale**..() могут изменять длину нормалей. Чтобы это учитывать, используется уже упоминавшийся режим нормализации нормалей, который включается вызовом команды **glEnable**(**GL_NORMALIZE**).

Для задания глобальных параметров освещения используются команды:

glLightModel[i f](pname, param: GLenum);

glLightModel[i f]v(pname: GLenum, const params: ^GLtype).

Аргумент *pname* определяет, какой параметр модели освещения будет настраиваться и может принимать следующие значения:

- **GL_LIGHT_MODEL_LOCAL_VIEWER** параметр *param* должен быть булевским и задает положение наблюдателя. Если он равен FALSE, то направление обзора считается параллельным оси $-z$, вне зависимости от положения в видовых координатах. Если же он равен TRUE, то наблюдатель находится в начале видовой системы координат. Это может улучшить качество освещения, но усложняет его расчет. Значение по умолчанию: FALSE.
- **GL_LIGHT_MODEL_TWO_SIDE** параметр *param* должен быть булевским и управляет режимом расчета освещенности как для лицевых, так и для обратных граней. Если он равен FALSE, то освещенность рассчитывается только для лицевых граней. Если же он равен TRUE, расчет проводится и для обратных граней. Значение по умолчанию: FALSE .
- **GL_LIGHT_MODEL_AMBIENT** параметр *params* должен содержать четыре целых или вещественных числа, которые определяют цвет фонового освещения даже в случае отсутствия определенных источников света. Значение по умолчанию:(0.2, 0.2, 0.2,1.0).

6.5.2 Наложение текстуры

Принятый в OpenGL формат хранения изображений отличается от стандартного формата Windows DIB только тем, что компоненты (R, G, B) для каждой точки хранятся в прямом порядке, а не в обратном и выравнивание задается программистом. При создании образа текстуры в памяти следует учитывать следующие требования:

1) Размеры текстуры как по горизонтали, так и по вертикали должны представлять собой степени двойки. Это требование накладывается для компактного размещения текстуры в памяти и способствует ее эффективному использованию. Использовать только текстуры с такими размерами конечно неудобно, поэтому перед загрузкой их надо преобразовать. Изменение размеров текстуры проводится с помощью команды **gluScaleImage**(format: GLenum, widthin, heightin: GLint, typein: GLenum, const ^datain, widthout, heightout GLint, typeout: GLenum, ^dataout). Параметры *widthin*, *heightin*, *widhtout*, *heightout* определяют размеры входного и выходного изображений, а с помощью *typein* и *typeout* задается тип элементов массивов, расположенных по адресам *datain* и *dataout*. Как и обычно, это может быть тип **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_INT** и так далее. Результат своей работы функция заносит в область памяти, на которую указывает параметр *dataout*.
2) Необходимо предусмотреть случай, когда объект по размерам значительно меньше наносимой на него текстуры. Чем меньше объект, тем меньше должна быть наносимая на него текстура, и поэтому вводится понятие уровней детализации текстуры. Каждый уровень детализации задает некоторое изображение, которое является, как правило, уменьшенной в два раза копией оригинала. Такой подход позволяет улучшить качество нанесения текстуры на объект. Например, для изображения размером $2^m \times 2^n$ можно построить $\max(m, n) + 1$ уменьшенных изображений, соответствующих различным уровням детализации.

Эти два этапа создания образа текстуры в памяти можно провести с помощью команды **gluBuild2DMipmaps**(target: GLenum, components, width, height: GLint, format, type: GLenum, const ^data), где параметр *target* должен быть равен **GL_TEXTURE_2D**, components определяет количество цветовых компонент текстуры, которые будут использоваться при ее наложении, и может принимать значения от 1 до 4 (1 — только красный,2 — красный и alpha, 3 — красный, синий, зеленый, 4 — все компоненты).

Параметры *width*, *height*, *data* определяют размеры и расположение текстуры соответственно, а *format* и *type* имеют аналогичный смысл, что и в команде **gluScaleImage**().

В OpenGL допускается использование одномерных текстур, то есть размера $1 \times N$, однако это всегда надо указывать, используя в качестве значения *target* константу **GL_TEXTURE_1D**. Существует одномерный аналог рассматриваемой команды — **gluBuild1DMipmaps**(), который отличается от двумерного отсутствием параметра *height*.

При использовании в сцене нескольких текстур в OpenGL применяется подход, напоминающий создание списков изображений. Вначале, с помощью команды **glGenTextures**(n^ GLsizei, textures: ^GLuint) надо создать *n* идентификаторов для используемых текстур, которые будут записаны в массив *textures*. Перед началом определения свойств очередной текстуры следует вызвать команду **glBindTexture**(target: GLenum, texture: GLuint), где *target* может принимать значения **GL_TEXTURE_1D** или **GL_TEXTURE_2D**, а параметр *texture* должен быть равен идентификатору той текстуры, к которой будут относиться последующие

команды. Для того чтобы в процессе рисования сделать текущей текстуру с некоторым идентификатором, достаточно опять вызвать команду **glBindTexture**() с соответствующим значением *target* и *texture*.

При наложении текстуры надо учитывать случай, когда размеры текстуры отличаются от размеров объекта, на который она накладывается. При этом возможно как растяжение, так и сжатие изображения, и то, как будут проводиться эти преобразования, может серьезно повлиять на качество построенного изображения. Для определения положения точки на текстуре используется параметрическая система координат (s,t), причем значения s и t находятся в отрезке $[0,1]$. Для изменения различных параметров текстуры применяются команды:

glTexParameter[if](target, pname, param: GLEnum);

glTexParameter[if]v(target, pname:Glenum, params: ^GLEnum).

При этом *target* имеет аналогичный смысл, что и раньше, *pname* определяет, какое свойство будем менять, а с помощью param или params устанавливается новое значение. Возможные значения pname:

- **GL_TEXTURE_MIN_FILTER** параметр *param* определяет функцию, которая будет использоваться для сжатия текстуры. При значении **GL_NEAREST** будет использоваться один (ближайший), а при значении **GL_LINEAR** четыре ближайших элемента текстуры. Значение по умолчанию: **GL_LINEAR**.
- **GL_TEXTURE_MAG_FILTER** параметр *param* определяет функцию, которая будет использоваться для увеличения (растяжения) текстуры. При значении **GL_NEAREST** будет использоваться один (ближайший), а при значении **GL_LINEAR** четыре ближайших элемента текстуры. Значение по умолчанию: **GL_LINEAR**.
- **GL_TEXTURE_WRAP_S** параметр *param* устанавливает значение координаты s, если оно не входит в отрезок $[0,1]$. При значении **GL_REPEAT** целая часть s отбрасывается, и в результате изображение размножается по поверхности. При значении **GL_CLAMP** используются краевые значения: 0 или 1, что удобно использовать, если на объект накладывается один образ. Значение по умолчанию: **GL_REPEAT**.
- **GL_TEXTURE_WRAP_T** аналогично предыдущему значению, только для координаты t.

Использование режима **GL_NEAREST** значительно повышает скорость наложения текстуры, однако при этом снижается качество, так как в отличие от **GL_LINEAR** интерполяция не производится.

Для того, чтобы определить, как текстура будет взаимодействовать с материалом, из которого сделан объект, используются команды:

glTexEnv[i f](target, pname, param: GLenum);

glTexEnv[i f]v(target, pname: Glenum, params: ^GLtype).

Параметр *target* должен быть равен **GL_TEXTURE_ENV**, а в качестве pname рассмотрим только одно значение **GL_TEXTURE_ENV_MODE**, которое применяется наиболее часто. Параметр *param* может быть равен:

- **GL_MODULATE** конечный цвет находится как произведение цвета точки на поверхности и цвета соответствующей ей точки на текстуре.
- **GL_REPLACE** в качестве конечного цвета используется цвет точки на текстуре.
- **GL_BLEND** конечный цвет находится как сумма цвета точки на поверхности и цвета соответствующей ей точки на текстуре с учетом их яркости.

Перед нанесением текстуры на объект осталось установить соответствие между точками на поверхности объекта и на самой текстуре. Задавать это соответствие можно двумя методами: отдельно для каждой вершины или сразу для всех вершин, задав параметры специальной функции отображения.

Первый метод реализуется с помощью команд:

glTexCoord [1 2 3 4][s i f d](coord: type);

glTexCoord [1 2 3 4][s i f d]v(coord: ^type).

Чаще всего используется команды вида **glTexCoord2**..(s, t: type), задающие текущие координаты текстуры.

6.5.3 Создание эффекта тумана

Туман в OpenGL реализуется путем изменения цвета объектов в сцене в зависимости от их глубины, т. е. расстояния до точки наблюдения. Изменение цвета происходит либо для вершин примитивов, либо для каждого пикселя на этапе растеризации в зависимости от реализации OpenGL.

Для включения тумана необходимо вызвать функцию **glEnable**(**GL_FOG**). Для контроля над туманом существуют функции:

glFog[i f](pname: GLenum, param: Type);

glFog[i f]v(pname: GLenum, param: ^Type).

Аргумент pname может принимать следующие значения:

GL_FOG_MODE аргумент *param* определяет формулу, по которой будет вычисляться интенсивность тумана в точке.

В этом случае *param* может принимать значения:

- **GL_EXP** — Интенсивность вычисляется по формуле $f = exp(-d \cdot z)$.
- **GL_EXP2** — Интенсивность вычисляется по формуле $f = exp\left(-(d \cdot z)^2\right)$.
- **GL_LINEAR** — Интенсивность вычисляется по формуле $f = (e - z)/(e - s)$, где z — расстояние от вершины, в которой вычисляется интенсивность тумана, до точки наблюдения.

Коэффициенты d, e, s задаются с помощью следующих значений аргумента *pname*:

- **GL_FOG_DENSITY** *param* определяет коээфициент d — плотность тумана.
- **GL_FOG_START** *param* определяет коэффициент s — ближайшая граница тумана.
- **GL_FOG_END** *param* определяет коэффициент e — задняя граница тумана.

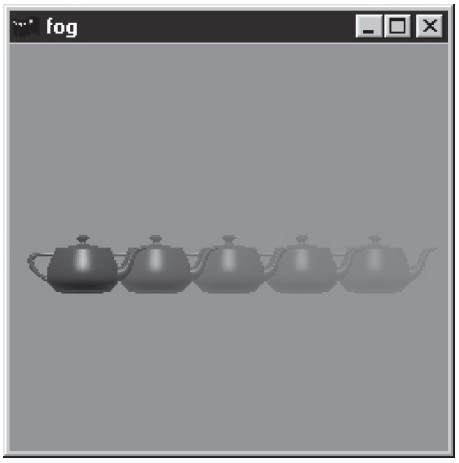


Рис. 6.6 – Эффект тумана в OpenGL

Цвет тумана задается с помощью аргумента *pname*, равного GL_FOG_COLOR, в этом случае *params* — указатель на массив из 4-х компонент цвета.

Пример

Пример использования эффекта тумана (рис. 6.6) :

```
glEnable(GL_FOG);
glFogi(GL_FOG_MODE,GL_LINEAR);
glFogf(GL_FOG_START,20.0);
glFogf(GL_FOG_END,100.0).
```

6.5.4 Использование буфера трафарета

В OpenGL существует буфер трафарета, с его помощью реализуются разнообразные эффекты, начиная от простого вырезания одной фигуры из другой до реализации теней, отражений и прочих нетривиальных функций. Трафарет это двумерный массив целочисленных переменных. Каждому пикселю в окне соответствует один элемент массива. Использование буфера трафарета происходит в два этапа. Сначала его заполняют, потом, основываясь на его содержимом, отображают объекты.

Рассмотрим функции библиотеки OpenGL для работы с трафаретом. Тест трафарета разрешается при помощи функции **glEnable** с параметром **GL_STENCIL_TEST**. Очищается буфер трафарета при помощи функции **glClear** с параметром **GL_STENCIL_BUFFER_BIT**. Заполнение буфера трафарета происходит при помощи следующих двух функций:

glStencilFunc(func: GLenum, ref: GLint, mask: GLuint);

glStencilOp(fail, zfail, zpass: GLenum).

Первая функция задает правило, по которому будет определяться, пройден тест трафарета или нет. Переменная *func* может принимать одно из следующих значений:

- **GL_NEVER** (не проходит);
- **GL_LESS** (проходит, если (ref **and** mask) < (stencil **and** mask);
- **GL_LEQUAL** (проходит, если (ref **and** mask) <= (stencil **and** mask);
- **GL_GREATER** (проходит, если (ref **and** mask) > (stencil **and** mask);
- **GL_GEQUAL** (проходит, если (ref **and** mask) >= (stencil **and** mask);
- **GL_EQUAL** (проходит, если (ref **and** mask) = (stencil **and** mask);
- **GL_NOTEQUAL** (проходит, если (ref **and** mask) <> (stencil **and** mask);
- **GL_ALWAYS** (всегда проходит).

Если тест трафарета не пройден, то фрагменты (пиксели) фигуры не прорисовываются в данном месте, т. е. они не попадают в буфер кадра. Если тест пройден, то фигура рисуется. Вторая функция позволяет задать, как будет инициализироваться буфер трафарета. Параметры *fail* (тест трафарета не пройден), *zfail* (тест трафарета пройден, *Z*-буфера — нет) и *zpass* (пройдены оба теста, либо буфер глубины не используется) могут принимать одно из следующих значений:

- **GL_KEEP** (сохранить текущее значение в буфере трафарета);
- **GL_ZERO** (обнулить);
- **GL_REPLACE** (заменить на ref);
- **GL_INCR** (увеличить на единицу);
- **GL_DECR** (уменьшить на единицу);
- **GL_INVERT** (поразрядно инвертировать).

Например, если мы хотим заполнить область трафарета, где рисуется куб единицами, то можно использовать следующий код:

```
glStencilFunc(GL_NEVER, 1, 0);
glStencilOp(GL_REPLACE, GL_KEEP, GL_KEEP);
glCallList(CUBE);
```

Первая функция говорит о том, что тест трафарета всегда проходит неудачно. Вторая функция задает, что, в случае неудачного теста трафарета, необходимо заменить значение, хранящееся в буфере трафарета, на значение переменной ref, а его мы задали равным единице. В результате, на экране ничего не отобразится, т. к. тест трафарета завершался неудачно, но в буфере трафарета мы получим проекцию куба из единичек, т. е. буфер трафарета заполнен не только нулями. Теперь мы хотим заполнить двойками область, где прорисовывается сфера. Здесь мы уже должны учитывать буфер глубины, иначе мы заполним двойками всю область, где у нас рисуется сфера. Для того чтобы учитывать буфер глубины, тест трафарета должен завершиться положительно. Третий параметр zpass функции **glStencilOp** как раз указывает, что делать, если тест трафарета прошел, а тест глубины нет (сфера выступает из куба). Поэтому код выглядит так:

```
glStencilFunc(GL_ALWAYS, 2, 0);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
glCallList(SPHERE);
```

В результате получаем буфер трафарета, заполненный нулями, где ничего не было, единицами, где виден куб, и двойками, где видна сфера. В последнем примере тест трафарета прошел успешно, поэтому на экране была нарисована сфера. Но это нам не мешает, мы очистим буфер глубины и буфер цвета, но не буфер трафарета.

glClear(GL_COLOR_BUFFER_BIT **or** GL_DEPTH_BUFFER_BIT).

Чтобы нарисовать тот же самый куб без сферы, надо установить, что тест трафарета проходит, если значение, находящееся в буфере трафарета, совпадает со значением второго параметра функции **glStencilFunc**.

glStencilFunc(GL_EQUAL, 1, 255);

glCallList(CUBE);

Здесь отрисовываем лишь ту поверхность куба, которая проходит тест глубины (т. е. ту, которая заполнена единичками) (рис. 6.7).

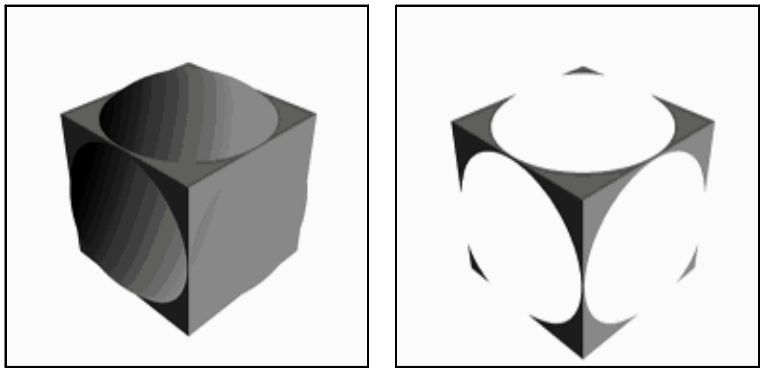


Рис. 6.7 – Использование буфера трафарета в OpenGL

6.5.5 Эффект прозрачности в OpenGl

За прозрачность отображаемой информации отвечает четвертая цветовая компонента — Alpha.

Альфа-компонента *— четвертая компонента значения цвета (кроме красного, зеленого и синего), определяющая прозрачность пикселя.*

В OpenGl Alpha-компонента может быть обработана двумя способами. Это может быть вывод изображений с отсечением пикселей, не проходящих определенного порогового значения Alpha, либо наложение одного изображения на другое с использованием значения Alpha как уровня прозрачности выводимого изображения относительно уже находящегося в буфере либо наоборот. Рассмотрим оба способа.

За разрешения проверки порогового уровня Alpha отвечает команда **glEnable(GL_ALPHA_TEST)**. После разрешения проверки для каждого выводимого пикселя на экране будет выполняться проверка Alpha-компоненты по условию заданному с помощью **glAlphaFunc**(func: GLenum, ref: GLclampf), где *ref* — содержит некоторое пороговое значение, а *func* может иметь значение:

- **GL_NEVER** (не проходит);
- **GL_LESS** (проходит, если ref < alpha);
- **GL_LEQUAL** (проходит, если ref <= alpha);
- **GL_GREATER** (проходит, если ref > alpha);
- **GL_GEQUAL** (проходит, если ref >= alpha);
- **GL_EQUAL** (проходит, если ref = alpha);
- **GL_NOTEQUAL** (проходит, если ref <> alpha);
- **GL_ALWAYS** (всегда проходит).

В конечном результате, на экране будут отображены лишь пиксели, прошедшие тест.

Для включения режима отработки прозрачности нам потребуется команда **glEnable(GL_BLEND)**. Аналогично предыдущему случаю, при включении данного режима в действие вступает функция **glBlendFunc**(sfactor, dfactor: GLenum), где параметры *sfactor* и *dfactor* определяют соответственно способ формирования исходного (входного изображения) и конечного (отображаемой сцены) цветов. Всего существует 11 методов вычисления цветовых компонент.

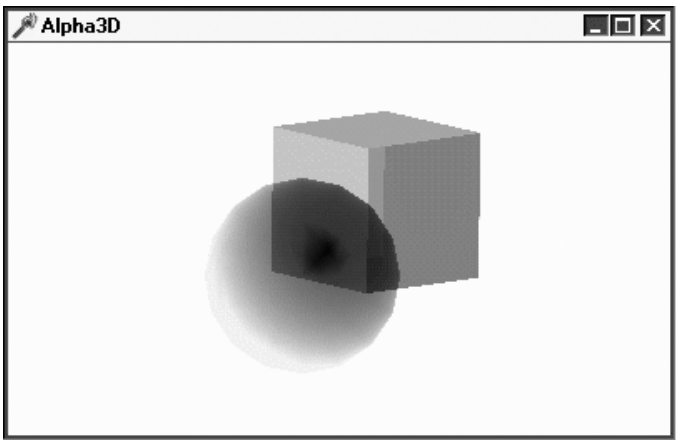


Рис. 6.8 – Реализация прозрачности в OpenGL

В данном случае нас интересуют лишь два значения — **GL_SRC_ALPHA** для *sfactor* и **GL_ONE_MINUS_SRC_ALPHA** для *dfactor*. Этот способ работает при отображении сцены, объекты в которой расположены последовательно, приближаясь к наблюдателю. В таком случае, при отображении очередного объекта мы, не изменив прозрачности уже созданной сцены, наложим на нее объект, учтя его Alpha-компоненту (рис. 6.8).

Контрольные вопросы по главе 6

1) Какие двумерные примитивы реализованы в библиотеке OpenGL?
2) Какие типы матриц используют в OpenGL?
3) Какое правило задает команда **glBegin**(GL_TRIANGLE_FAN)?
4) Какая команда позволяет добавить источник света в сцену?
5) Какие виды проективных преобразований реализованы в OpenGL?

ЗАКЛЮЧЕНИЕ

Компьютерная графика достаточно сложная и объемная дисциплина. В настоящем пособии рассмотрены её базовые понятия и определения, некоторые особенности, возможности и ограничения. В данном учебном пособии вы познакомились с базовыми понятиями вычислительной геометрии, на основе которых построено большинство алгоритмов компьютерной графики.

Используя современные программные средства, вы сможете реализовать многие из рассмотренных алгоритмов, в том числе и в случае решения реальных практических задач.

Автор надеется, что книга поможет Вам в освоении этой интересной и многообразной дисциплины.

ЛИТЕРАТУРА

[1] Амерал Л. Принципы программирования в машинной графике / Л. Амерал. — М. : Сол Систем, 1992. — 224 с.

[2] Амерал Л. Машинная графика на языке С / Л. Амерал. — М. : Мир, 1982. — 184 с.

[3] Васильев В. Е. Компьютерная графика : учеб. пособие / В. Е. Васильев, А. В. Морозов. — СПб. : СЗТУ, 2005. — 101 с.

[4] ГОСТ27817-88 Системы обработки информации. Машинная графика. Функциональное описание ядра графической системы.

[5] Иванов А. П. Трехмерная компьютерная графика / А. П. Иванов, А. С. Батраков. — М. : Радио и связь, 1995. — 224 с.

[6] Казанцев А. В. Основы компьютерной графики : в 2 ч. / А. В. Казанцев. — Казань, 2001. — Ч. 1 : Математический аппарат компьютерной графики. — 62 с.

[7] Краснов М. В. OpenGL. Графика в проектах Delphi / М. В. Краснов. — СПб. : БХВ-Санкт-Петербург, 2000. — 352 с.

[8] Люкшин Б. А. Инженерная и компьютерная графика : учеб. пособие / Б. А. Люкшин. — Томск : Томский межвузовский центр дистанционного образования, 2004. — Ч.1: Вопросы теории компьютерной графики. — 141 с.

[9] Мураховский В. И. Компьютерная графика / В. И. Мураховский ; под ред. С. В. Симоновича. — М. : АСТ-ПРЕСС СКД, 2002. — 640 с.

[10] Никулин Е. А. Компьютерная геометрия и алгоритмы машинной графики / Е. А. Никулин. — СПб. : БХВ-Петербург, 2005. — 576 с.

[11] Ньюмен У. Основы интерактивной машинной графики / У. Ньюмен, Р. Спрулл. — М. : Мир, 1976. — 573 с.

[12] Петров М. Н. Компьютерная графика : учебник для вузов / М. Н. Петров, В. П. Молочков. — СПб. : Питер, 2003. — 736 с.

[13] Порев В. Н. Компьютерная графика / В. Н. Порев. — СПб. : БХВ-Петербург, 2004. — 432 с.

[14] Постнов К. В. Компьютерная графика / К. В. Постнов. — М., 2009. — 247 с.

[15] Резниченко С. В. Аналитическая геометрия в примерах и задачах: алгебраические главы / С. В. Резниченко. — М. : Физматлит, 2002. — 576 с.

[16] Роджерс Д. Алгебраические основы машинной графики / Д. Роджерс, Дж. Адамс. — М. : Мир, 1989. — 512 с.

[17] Роджерс Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс. — М. : Мир, 2001. — 604 с.

[18] Рост Р. Дж. OpenGL. Трехмерная графика и язык программирования шейдеров. Для профессионалов / Р. Дж. Рост. — СПб. : Питер, 2005. — 428 с.

[19] Шикин Е. В. Начала компьютерной графики / Е. В. Шикин, А. В. Боресков, А. А. Зайцев. — М. : Диалог-МИФИ, 1993. — 138 с.

[20] Чириков С. В. Алгоритмы компьютерной графики (методы растрирования кривых) : учеб. пособие / С. В. Чириков. — СПб. : СПб ГИТМО(ТУ), 2001. — 120 с.

[21] Эйнджел Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGl : пер. с англ. / Э. Эйнджел. — 2-изд. — М. : Издательский дом «Вильямс», 2001. — 592 с.

ГЛОССАРИЙ

OpenGL (Open Graphic Library) — библиотека графических функций, интерфейс для графических прикладных программ. Разработана Silicon Graphics.

Аксонометрическая проекция — проекция, у которой проектирующие прямые перпендикулярны картинной плоскости, сама картинная плоскость располагается в пространстве произвольным образом.

Алгебраическая поверхность — поверхность, для которой функция $f(x,y,z)$ есть сумма полиномов трех переменных.

Алиасинг — дефекты изображения, эффект «ступенчатости» изображения.

Антиалисинг — технология, убирающая эффект «ступенчатости» с использованием различных алгоритмов сглаживания.

Альфа-компонента — четвертая компонента значения цвета (кроме красного, зеленого и синего), определяющая прозрачность пикселя.

Аффинное пространство — расширение векторного пространства, в которое включен дополнительный тип объектов — точка.

Аффинная система координат — система в *n*-мерном аффинном пространстве, определяемая совокупностью *n* линейно независимых векторов, исходящих из начала координат.

Буфер глубины (Z-buffer, depth buffer) — дополнительный объем памяти, где хранится значение глубины примитивов передней поверхности (расстояние от наблюдателя до поверхности изображаемого объекта) для каждого пикселя.

Буфер кадра — часть графической памяти для хранения массива кодов, определяющих засветку пикселей на экране.

Визуализация (Rendering) — создание плоских изображений трехмерных (3D) моделей.

Вектор — направленный отрезок прямой линии, характеризуемый только его длиной и направлением.

Видовая матрица определяет преобразования объекта в мировых координатах, такие, как параллельный перенос, изменение масштаба и поворот.

Векторное изображение — тип изображения, которое состоит из геометрических объектов, описанных математически.

Геометрический примитив — элементарный фрагмент изображения, при помощи которого описывается объемный объект.

Глубина буфера кадра — величина, характеризующая количество бит информации, определяющих засветку каждого отдельного пикселя, в частности количество цветов, которое может быть представлено на экране данной системы.

Графический пользовательский интерфейс — обеспечивает возможность управления поведением вычислительной системы через визуальные элементы управления — окна, списки, кнопки, гиперссылки и т. д.

Диффузное отражение — отражение, при котором падающий на поверхность луч рассеивается одинаково по всем направлениям.

Зеркальное отражение — отражение, при котором угол между нормалью и падающим лучом равен углу между нормалью и отраженным лучом.

Интерфейс — совокупность средств и методов обеспечения взаимодействия между элементами системы.

Компьютерная графика — воспроизводит изображение в случае, когда исходной является информация неизобразительной природы.

Косоугольная проекция — такая проекция, у которой проектирующие прямые образуют с плоскостью проекции угол, отличный от 90°.

Линиатура растра — характеризует период сетки и обозначает количество линий растра на единицу длины изображения.

Матрица проекций задает, как будут проецироваться трехмерные объекты на плоскость экрана.

Матрица текстуры определяет наложение текстуры на объект.

Мировая система координат — содержит точку отсчета (начало координат) и линейно независимый базис, благодаря которым становится возможным цифровое описание геометрических свойств любого графического объекта в абсолютных единицах.

Моделирование (modeling) — создание и представление трехмерных (3D) моделей.

Объектная система координат — система координат, связанная с конкретным объектом и совершающая с ним все движения в системе координат сцены или мировой системе координат.

Обработка изображений — рассматривает задачи, в которых и входные, и выходные данные являются изображениями.

Однородные координаты точки — тройка одновременно не равных нулю чисел x_1, x_2, x_3, если $x = x_1/x_3$, $y = x_2/x_3$.

Ортографическая проекция — проекция, в которой картинная плоскость совпадает с одной из координатных плоскостей или параллельна ей.

Отсечение (clipping) — процесс отбрасывания частей изображения, выходящих за границы окна.

Перспективная (центральная) проекция — вид проекции, где лучи проектирования исходят из одного центра (центра проектирования), размещенного на конечном расстоянии от объектов и плоскости проектирования.

Пользовательский интерфейс — элементы и компоненты программы, способные оказывать влияние на взаимодействие пользователя с программным обеспечением.

Проектирование — процесс, в ходе которого создается прототип, прообраз необходимого объекта. Отображение точек, заданных в системе координат с размерностью N, в точки в системе меньшей размерности n, где $N < n$.

Проектирующие лучи — отрезки прямых, идущих из центра проекции через каждую точку объекта до пересечения с картинной плоскостью (плоскостью проекции).

Распознавание образов (система технического зрения) — совокупность методов, позволяющих получить описание изображения, поданного на вход, либо отнести заданное изображение к некоторому классу.

Растрирование (rasterization) — процесс генерации растрового образа геометрического объекта.

Разрешение изображения — количество пикселей на единицу длины.

Растровая графика — способ построения изображений, в котором изображение представляется массивом простейших элементов — пикселей, где каждый пиксель имеет четко заданное положение.

Система координат сцены — описывает положение всех объектов сцены — некоторой части мирового пространства с собственным началом отсчета и базисом, которые используются для описания положения объектов независимо от мировой системы координат.

Сильносвязный путь (4-путь) на плоскости — множество точек A_1, A_2, …, A_n, для которых точки A_i и A_{i+1} являются непосредственными соседями для $i = 1, 2, \ldots, n - 1$.

Слабосвязный путь (8-путь) на плоскости — множество точек A_1, A_2, …, A_n, для которых точки A_i и A_{i+1} являются косвенными соседями для $i = 1, 2, \ldots, n - 1$.

Скаляр (scalaris — ступенчатый) — величина, каждое значение которой может быть выражено одним (действительным) числом. Примерами скаляра являются длина, площадь, время, масса, плотность и т. д.

Сплайн — кривая или поверхность, используемая для представления сложных гладких кривых или поверхностей.

Текстура — стиль заполнения, закрашивание, которое имитирует сложную рельефную объемную поверхность, выполненную из какого-то материала.

Тексель — минимальная единица текстуры.

Трассировка лучей — метод реалистической визуализации, моделирующий движение светового луча в изображаемой сцене; яркость точки экрана (пикселя) определяется интенсивностью проходящего через нее луча.

Фрактал — геометрическая фигура, обладающая свойством самоподобия, то есть составленная из нескольких частей, каждая из которых подобна всей фигуре целиком.

Цветовая модель — с пособ разделения цветового оттенка на составляющие компоненты.

Цветовая модель CMYK (Cyan Magenta Yellow BlacK) — субтрактивная цветовая модель, согласно которой цвет кодируется четырьмя компонентами — голубым, лиловым, желтым и черным.

Цветовая модель HSB (Hue Saturation Brightness) — цветовое пространство, основанное на трех характеристиках цвета: цветовом тоне, насыщенности и яркости.

Цветовая модель RGB (Red Green Blue) — аддитивная цветовая модель, согласно которой цвет кодируется тремя компонентами — красным, зеленым и синим.

Экранная система координат — система координат, в которой задается положение проекций геометрических объектов на экране дисплея.

Ядро графической системы — функциональный интерфейс между прикладной программой и конфигурацией графических устройств ввода и вывода.

Учебное издание

Перемитина Татьяна Олеговна

КОМПЬЮТЕРНАЯ ГРАФИКА

Учебное пособие

Корректор Осипова Е. А.
Компьютерная верстка Хомич С. Л.

Подписано в печать 06.12.12. Формат 60x84/8.
Усл. печ. л. 16,74. Тираж 300 экз. Заказ

Издано в ООО «Эль Контент»
634029, г. Томск, ул. Кузнецова д. 11 оф. 17
Отпечатано в Томском государственном университете
систем управления и радиоэлектроники.
634050, г. Томск, пр. Ленина, 40
Тел. (3822) 533018.